Choreographing the Extended Agent

performance graphics for dance theater

Marc Downie

Bachelor of Arts, University of Cambridge, 1998
Master of Science (Natural Science), University of Cambridge, 1998
Master of Arts, University of Cambridge, 2001
Master of Science, (Media Arts and Sciences), Massachusetts Institute of Technology, 2001

Submitted to the program in Media Arts and Sciences, School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of Doctor of Philosophy
at the Massachusetts Institute of Technology,
September 2005

Author		August 18, 2005
Certified by		Tod Machover Professor of Music and Media MIT Media Lab Thesis Supervisor
Accepted by		Andrew B. Lippman Chairperson, Departmental Committee on Graduate Students MIT Media Lab

Abstract

The marriage of dance and interactive image has been a persistent dream over the past decades, but reality has fallen far short of potential for both technical and conceptual reasons. This thesis proposes a new approach to the problem and lays out the theoretical, technical and aesthetic framework for the innovative art form of digitally augmented human movement. I will use as example works a series of installations, digital projections and compositions each of which contains a choreographic component — either through collaboration with a choreographer directly or by the creation of artworks that automatically organize and understand purely virtual movement. These works lead up to two unprecedented collaborations with two of the greatest choreographers working today; new pieces that combine dance and interactive projected light using real-time motion capture live on stage.

The existing field of "dance technology" is one with many problems. This is a domain with many practitioners, few techniques and almost no theory; a field that is generating "experimental" productions with every passing week, has literally hundreds of citable pieces and no canonical works; a field that is oddly disconnected from modern dance's history, pulled between the practical realities of the body and those of computer art, and has no influence on the prevailing digital art paradigms that it consumes.

This thesis will seek to address each of these problems: by providing techniques and a basis for "practical theory"; by building artworks with resources and people that have never previously been brought together, in theaters and in front of audiences previously inaccessible to the field; and by proving through demonstration that a profitable and important dialogue between digital art and the pioneers of modern dance can in fact occur.

The methodological perspective of this thesis is that of biologically inspired, agent-based artificial intelligence, taken to a high degree of technical depth. The representations, algorithms and techniques behind such agent architectures are extended and pushed into new territory for both interactive art and artificial intelligence. In particular, this thesis will focus on the control structures and the rendering of the extended agents' bodies, the tools for creating complex agent-based artworks in intense collaborative situations, and the creation of agent structures that can span live image and interactive sound production. Each of these parts becomes an element of what it means to "choreograph" an extended agent for live performance.

Choreographing the Extended Agent

performance graphics for dance theater

Doctoral Dissertation Committee

Thesis supervisor		Tod Machover Professor of Music and Media MIT Media Lab Thesis Supervisor
Thesis reader		Bruce Blumberg Director, Advanced Animal Modeling BlueFang Games Inc.
Thesis reader		Mark Goulthorpe Associate Professor MIT Department of Architecture

Acknowledgments

When I was sixteen I was taken by a rather odd idea (it was from Xenakis, I believe, although he is not to blame), that to be a great artist, I should study theoretical physics at the University of Cambridge. I learned a great deal during that time and my approaches to difficult problems, and to the technicalities and potentials of the unknown, have remained unchanged since that time.

But the main thing that I learned is that in the attempt to be even a good artist it's important to work with great people. **Bruce Blumberg** got me over to The Media Lab as part of the then recently formed Synthetic Characters Group; I learnt a lot from Bruce, of course, but above all I learnt how to make things — to start them, finish them, show them to people and then make them better. Throughout that time and after, **Tod Machover**'s vision in general, and interest and encouragement in particular, has proved more valuable and important than I've had an opportunity to admit before. Despite sharing a much shorter relationship, **Mark Goulthorpe** has caused me to rethink the relationship of my work to a broad intellectual community. I look forward to an opportunity to work with each of you again.

Many thanks are due to my good friends and colleagues **Paul Kaiser** and **Shelley Eshkar** who are collaborators on all of my works that involve human motion in any way — *Loops*, *Loops Score*, *22*, and *how long*... Without your minds, eyes, hands and hearts, these pieces and the years spent building them are inconceivable.

To **Trisha Brown** — my monitor is so blank without your points of movement hidden upon it.

To my fellow members of the **Synthetic Characters Group** and my fellow graduate students at **The Media Lab** — in particular **Bill Tomlinson**, **Matt Berlin** and **Ari Benbasat** — you have been the source of the magic material which is the difference between idea and reality.

And finally, to my wife, **Alison James**, who has been with me since the early days in Cambridge — you have been the great reason and the great love that has remained constant.

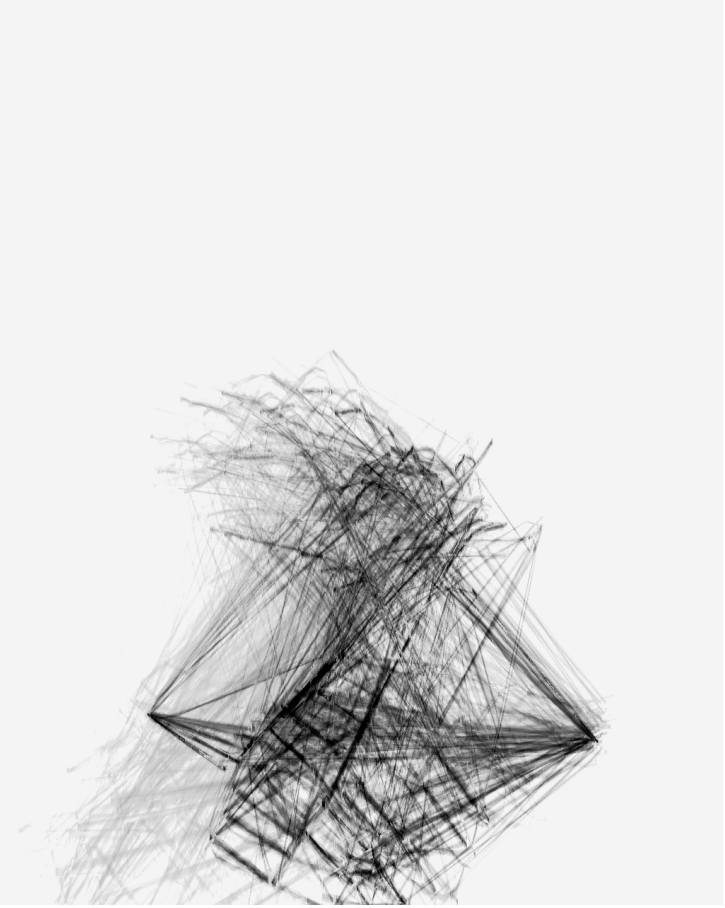
Marc Downie
August 23, 2005 — New York

Table of Contents

Introduction

The marriage of dance and interactive image has been a persistent dream over the past decades, but reality has fallen far short of potential for both technical and conceptual reasons. This thesis proposes a new approach to the problem and lays out the theoretical, technical and aesthetic framework for the innovative art form of digitally augmented human movement. I will use as example works a series of installations, digital projections and compositions each of which contains a choreographic component — either through collaboration with a choreographer directly or by the creation of artworks that automatically organize and understand purely virtual movement. These works lead up to two unprecedented collaborations with two of the greatest choreographers working today; new pieces that combine dance and interactive projected light using real-time motion capture live on stage.

This thesis will achieve its goals because of its methodological perspective — that of biologically inspired, agent-based artificial intelligence — and the technical depth to which this idea is taken. The representations, algorithms and techniques behind such agents are extended and pushed into territory that is new for both interactive art and artificial intelligence. In particular, this thesis will focus on the control structures and the rendering of the these extended

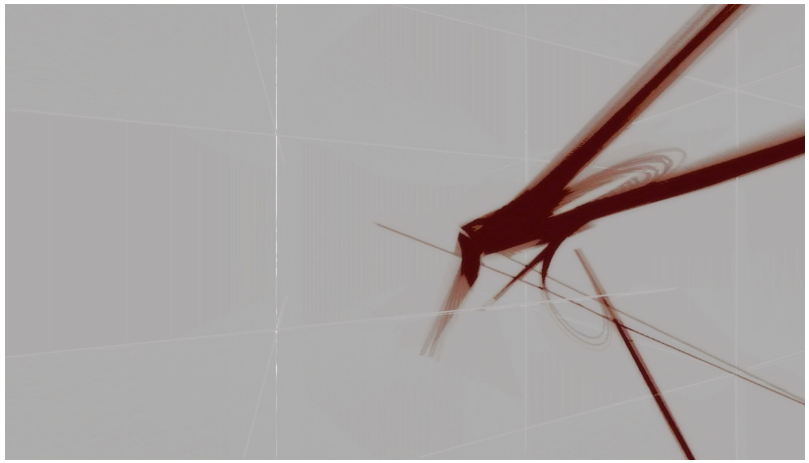


from *Loops* (inverted)

agents' bodies, the tools for creating complex agent-based artworks in intense collaborative situations, and the creation of agent structures that can span live image and interactive sound production — each part an element of what it means to "choreograph" an extended agent for live performance.

In this document I will present five principal artworks developed over a period of four years. The earliest of these is *Loops*, an installation work. A "digital portrait" of choreographer Merce Cunningham, *Loops* takes as its point of departure a "motion-captured" performance of Cunningham performing his 1970s dance for hands of the same name. Since its premiere in Cambridge in 2001, this piece has toured extensively — garnering an honorable mention at the Ars Electronica festival in 2004, installations at the Institute for Contemporary Art in London, and the Acm Siggraph 2002 conference and, as part of a Cunningham "event", a showing at the Festival d'automne, Paris. The piece was created as an interactive work, but is not interactive in its current version. It nevertheless remains a "live" work in the sense that it is computed, that is, made, live. As such, the work never repeats; rather, it is perhaps the first point of contact between Cunningham's "discovery" of the creative potential of chance procedures and artificial intelligence's deployment of probabilistic techniques. Although *Loops* was constructed in collaboration over a short one-month period, it offers many early examples of what I consider to be the creative strengths of my agent-based practice.

Proceeding chronologically, the next principal artwork is *The Music Creatures* — a series of interactive, multi-screen installations. This thesis will focus on the most recent of the series, the 2003 installation commissioned by the Ars Electronica festival. These creatures offer small, "animal-level", musical intelligences; inspired by, but not based directly upon, the acoustic abilities of birds. The creatures come in four varieties, but each creature makes sound solely by manipulating its virtual body, and the growth and appearance of that body is governed by the creature's learnt understanding of its acoustic environment. While

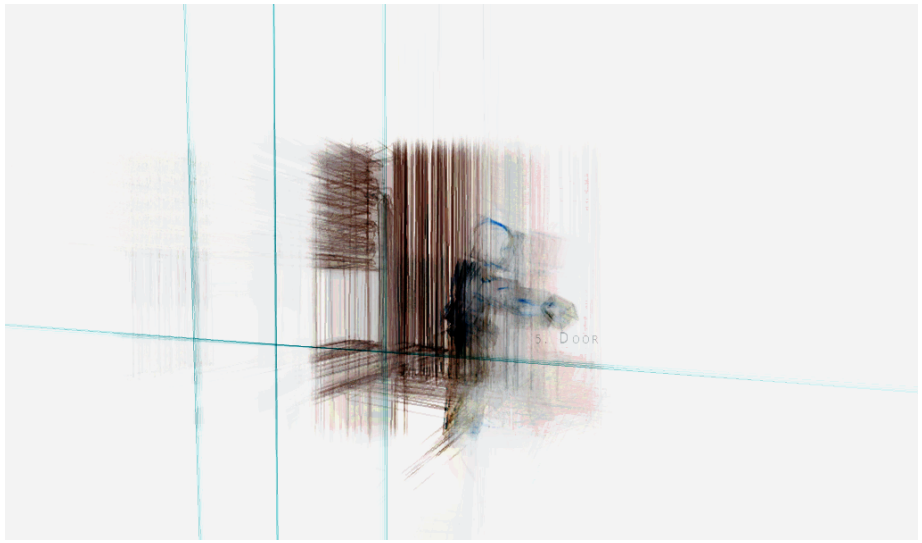


network from *The Music Creatures*

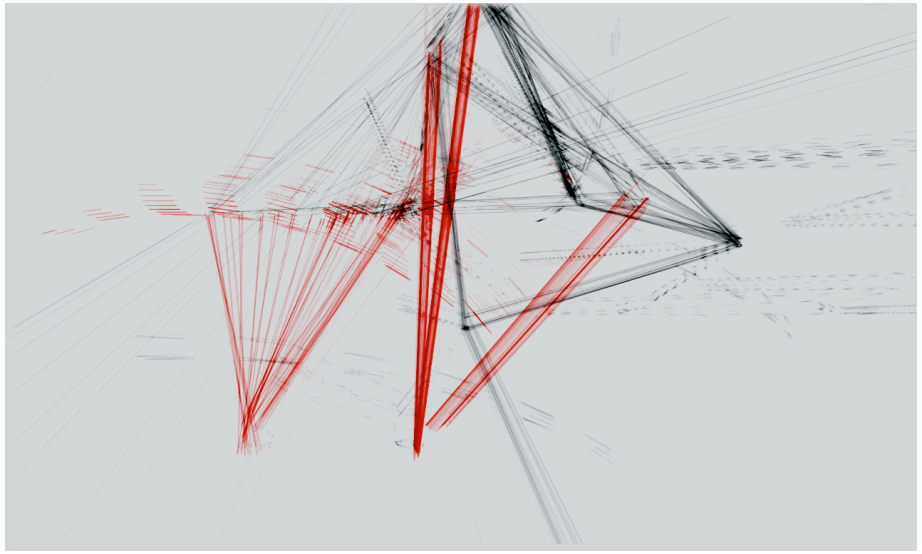
this work does not include human motion, *The Music Creatures*, with their long and multiply versioned development, are the work that is perhaps most responsible for refining my agent-based aesthetics. Further, the creatures in this work utilize a range of AI techniques to maintain a position of "dynamic disequilibrium" with the gallery space and each other, conveying a sense of effort, intention and ultimately transience and instability. This fragmentary and accumulative techniques and aesthetics is fundamental to my approach to interactive imagery in general and human motion in particular.

I revisited the *Loops* installation last year, 2004, with *Loops Score*, a purely musical work to accompany *Loops*. While *Loops* began with Cunningham's performance of his solo for hands, *Loops Score* begins with a narration by Cunningham — reading from his diary, concerning his first visit to New York in 1937. The sound of this narration is recast by a battery of interacting agent processes onto a set of extended prepared pianos, using a high-resolution sample library provided by the John Cage Foundation. While *The Music Creatures* presented an extremely minimal, indeed visual, approach to music, *Loops Score* finds itself closer to the mainstream concerns of computer music. However, *Loops Score* retains the indirection of *The Music Creatures*, deferring the creation of new live music to an autonomous agent. This piece shares with *Loops* a technical focus on the strategies available to "score" such open works, and similar to *The Music Creatures*, *Loops Score* produces music that is at some times startlingly coupled to its source, and at others propelled and sustained by its own oddly inevitable logic. *Loops Score* premiered in 2004 at the Ars Electronica festival.

The final artworks I present in this thesis are my most recent involving choreography, two works entitled *22* and *how long does the subject linger on the edge of the volume ...* The first, *22*, was created in collaboration with choreographer / performer Bill T. Jones, the second, *how long...*, in collaboration with choreographer Trisha Brown. There are several technical accomplishments in these works, for they are live visual imagery for motion-captured dance performance; while



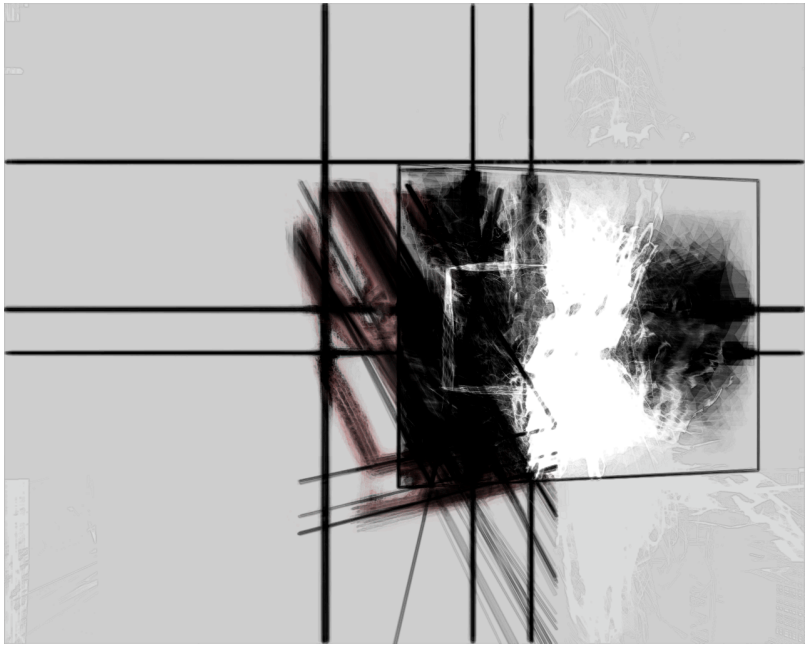
door from *22*



triangle from *how long...*

Loops in 2001 used a carefully recorded, painstakingly hand-cleaned reconstruction of Cunningham's hands, these works in 2005 capture an entire proscenium stage in real-time. These works are some of the first to use this technology in front of an audience, and, to my knowledge, the first to do so on such a scale. Further, while the involvement of Cunningham began and ended with the capturing of his motion, these works were created truly in collaboration with the choreographers. I believe a number of technical and conceptual contributions in this thesis have are the consequence of the need both to keep pace with Brown and Jones in workshop and to meet the challenges of their choreographic practices. These works received their premiere in Arizona in April 2005; *how long...* showed the very next week live at the Lincoln Center for Performing Arts in New York, and the imagery has since received an award of distinction at Ars Electronica. The non-live "touring" version of *22* showed at the opening of the Walker Performing Arts Center in Minneapolis in June. Both are currently on tour.

Along the way there have been other works that will appear less frequently in this thesis. Of most interest is probably the most recent of all my works, *Imagery for Jeux Deux* — the live visual imagery made to accompany Tod Machover's concerto for "hyperpiano", *Jeux Deux*. I will use this work to provide a number of example implementations, and it will often serve to demonstrate the applicability of my techniques to domains outside dance. Other works include *Lifelike* — live, but non-interactive imagery for the Merce Cunningham Dance Company commissioned and premiered by the Barbican Centre, London — and *Weather for an interactive window*, a small work for Joe Paradiso and the MIT Media Lab Responsive Environments Group's "tapper window" — a sensing piece of architectural glass. These too will appear in order to make arguments for experience, breadth, or applicability. This thesis will also make extensive use, especially in the early chapters, of two collaborative pieces by the MIT Media Lab's Synthetic Characters Group directed by Bruce Blumberg, of which I was a member: *Dobie* — an interactive, trainable dog; and *alphaWolf*



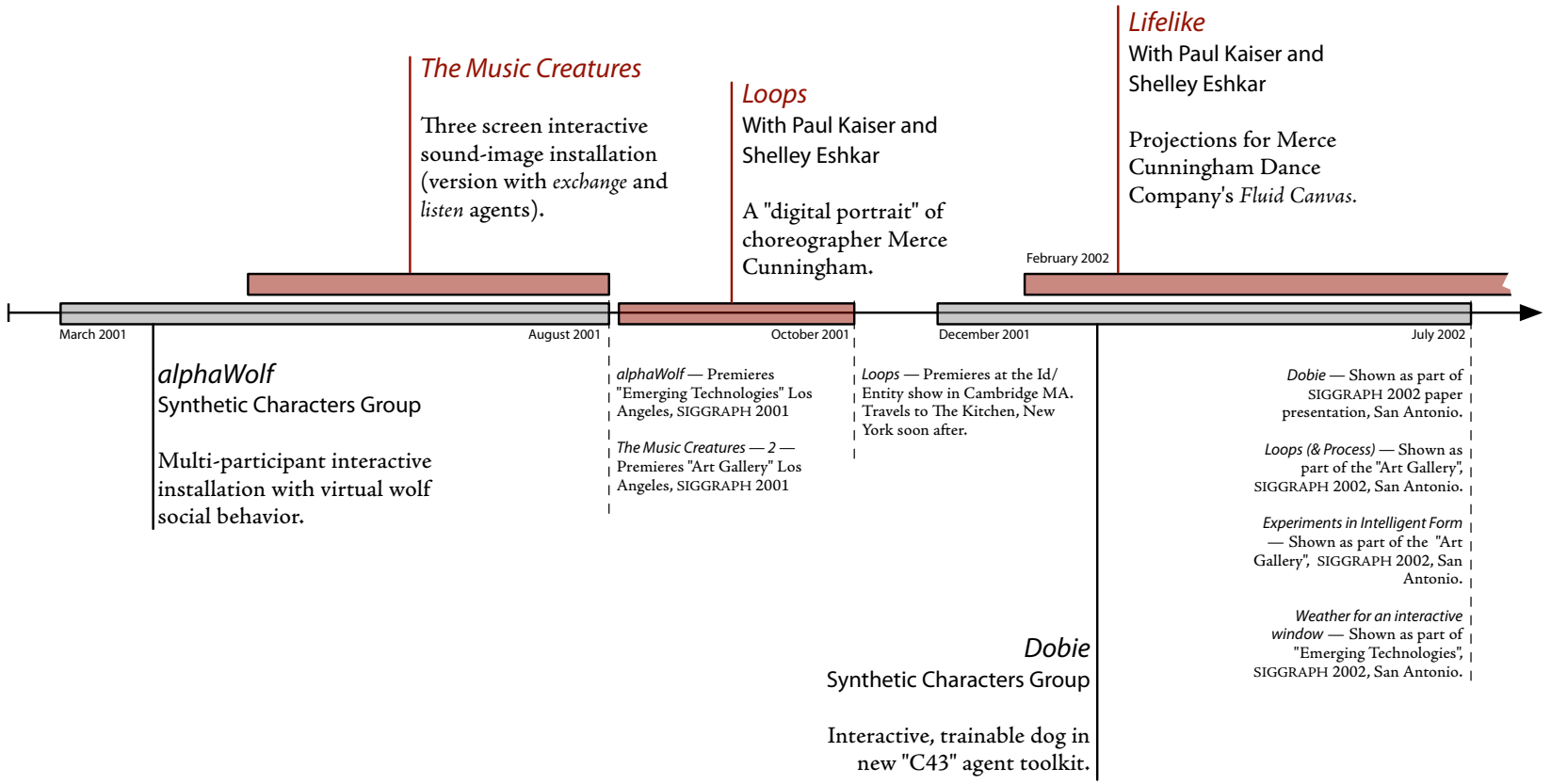
mirror from *Imagery for Jeux Deux*

— a multi-participant interactive simulation of wolf social behavior. *Dobie* is of considerable interest because he represents the high-water mark of the Synthetic Characters development of trainable characters in the toolkit that forms the basis for my subsequent work; *alphaWolf* because it represents the large, complex, multi-programmer collaboration at around the same time.

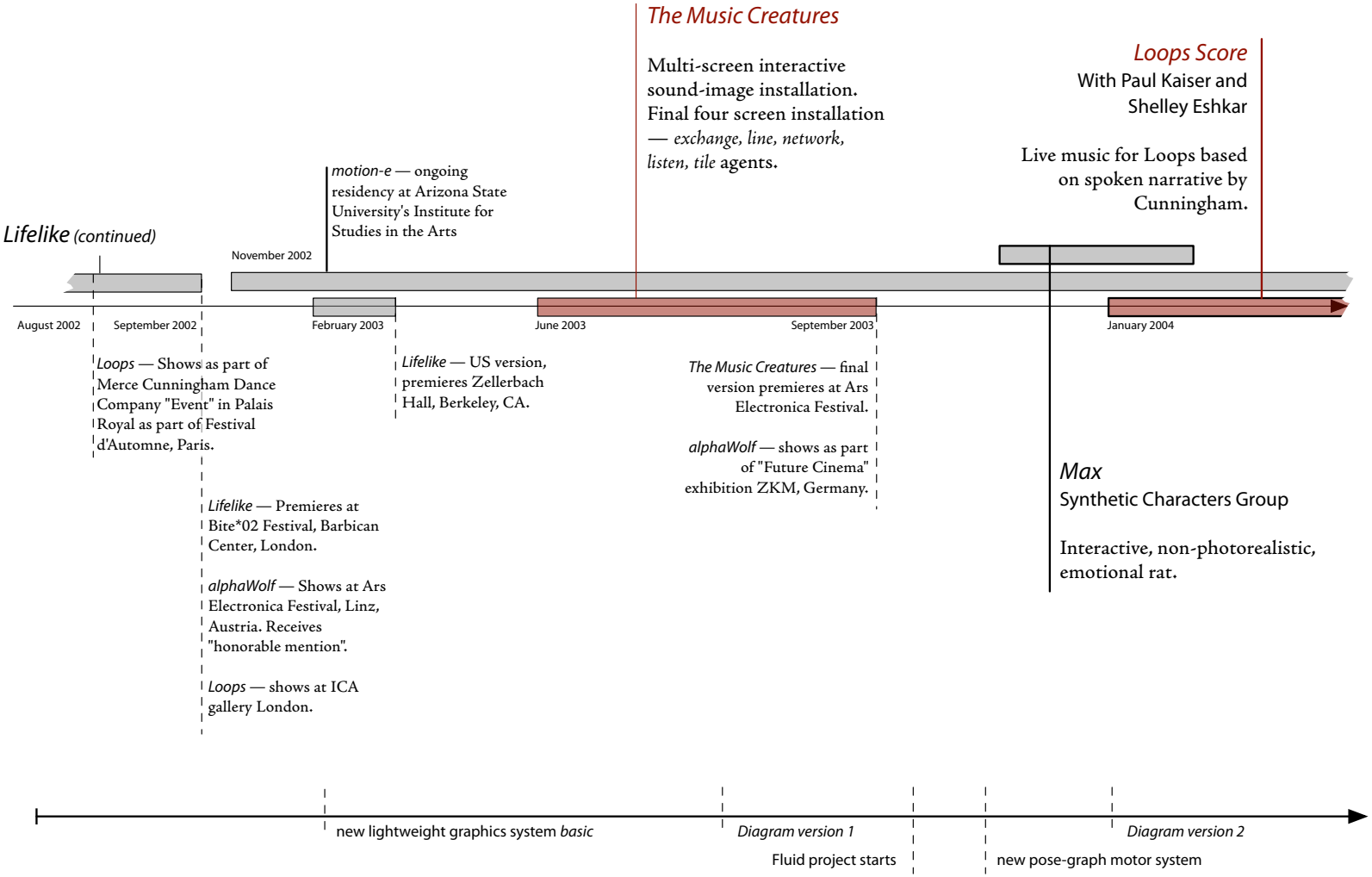
Chapter 1 will contextualize this thesis, locating it between the three areas it is in contact with — choreography, artificial intelligence and computer graphics — and will outline the main arguments both technical and conceptual that will appear in the remainder of the document. The next chapter will lay the groundwork for our agent framework, and survey the particular starting point for the agents constructed for this thesis. It will indicate how the agent-based might fit into an art practice, and what kinds of work AI architectures need to meet the requirements of a practicing artist. Following this chapter will come an overview of my first artwork concerning human motion — *Loops* — that critically develops a response to what I believe to be artificial life's "anti-methodologies" of emergence. Proceeding chronologically, I then present the sound-image installation *The Music Creatures*. This installation, while not drawing upon human movement, helps define several aspects of my agent-based aesthetics and sharpen some of the strategies that it offers in dealing with the uncertainties of interaction. I then pause in chapter 5 to discuss two general frameworks for constructing the perception systems for agents in complex worlds, that will be of specific use in the dance theater works. It is this chapter that contains the most focused technical rebuke of "mapping", a term in widespread use in interactive art. Chapter 6 introduces *Loops Score*, and more importantly collects the extensions to the agent framework, based on the lessons learnt in making *The Music Creatures* and *Loops*, into a new agent toolkit: the Diagram framework. These framework is designed to offer new forms of authorial involvement in the creation and maintenance of agents. Chapter 7 presents *how long...* and *22*, both pieces for interactive dance theater. *22* provides my most focused attempt to reform computer graphics' "non-photoreal" with

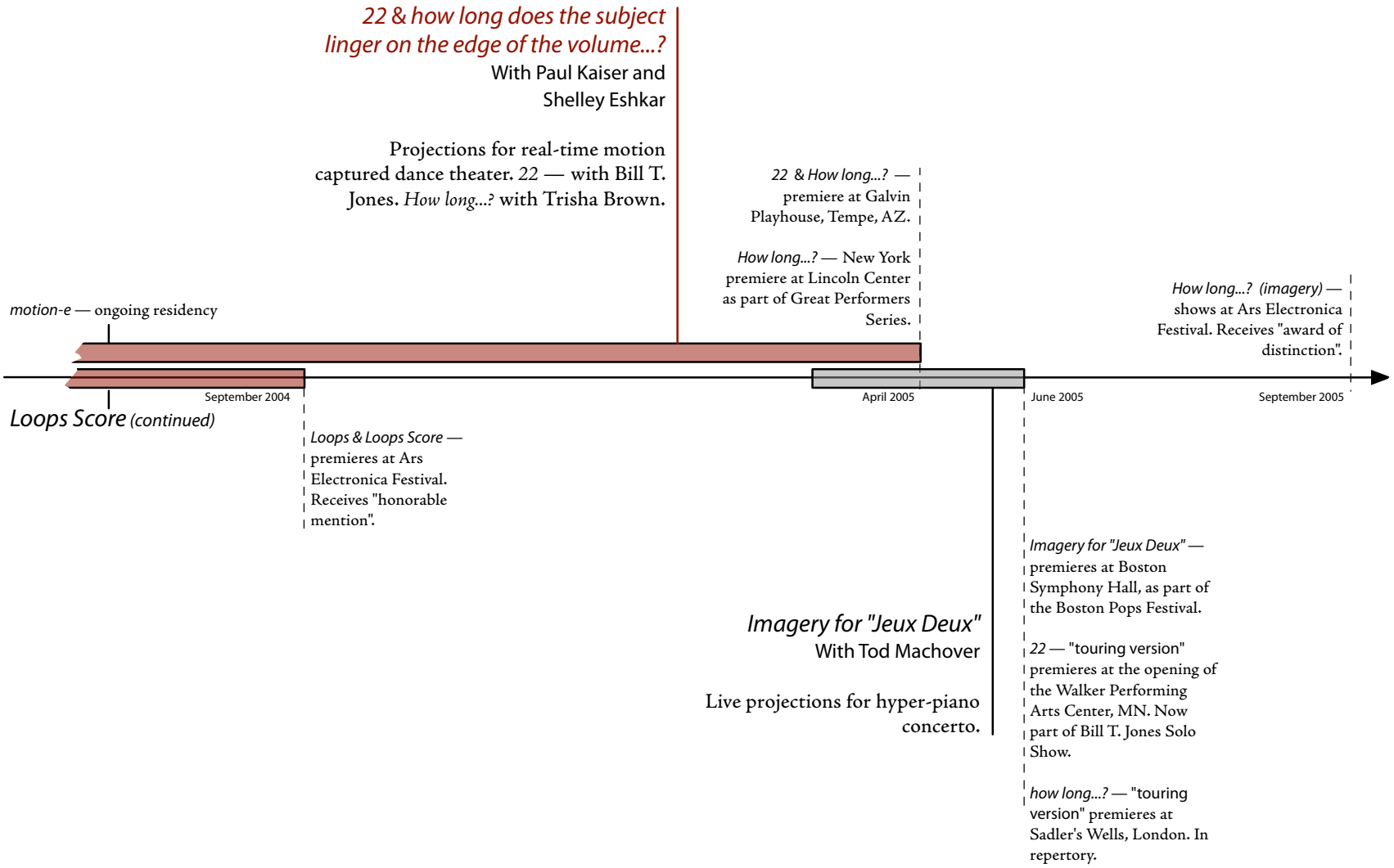
new rendering techniques, while *how long...?* represents my most sustained effort to create a collaboration between digital visual imagery and choreography in a live setting. Chapter 8 concludes the main body of this thesis with a description of a parallel thread — the custom graphical environment that allowed my agent-based approach to meet the realities of collaboration, rehearsal and improvisatory choreographic practice. A section summarizing the technical and aesthetic contributions of this thesis, and indicating the possibilities for future work, follows.

summer 2001 — summer 2002



summer 2002 — winter 2004




22 & how long does the subject linger on the edge of the volume...?
With Paul Kaiser and Shelley Eshkar
Projections for real-time motion captured dance theater. 22 — with Bill T. Jones. How long...? with Trisha Brown.
22 & How long...? — premiere at Galvin Playhouse, Tempe, AZ.
How long...? — New York premiere at Lincoln Center as part of Great Performers Series.
How long...? (imagery) — shows at Ars Electronica Festival. Receives "award of distinction".
motion-e — ongoing residency
September 2004
April 2005
June 2005
September 2005
Loops Score (continued)
Loops & Loops Score — premieres at Ars Electronica Festival. Receives "honorable mention".
Imagery for "Jeux Deux"
With Tod Machover
Live projections for hyper-piano concerto.
Imagery for "Jeux Deux" — premieres at Boston Symphony Hall, as part of the Boston Pops Festival.
22 — "touring version" premieres at the opening of the Walker Performing Arts Center, MN. Now part of Bill T. Jones Solo Show.
how long...? — "touring version" premieres at Sadler's Wells, London. In repertory.

This thesis touches upon many areas of work, many intellectual and artistic fields of endeavor. This chapter reviews some of the previous work in these areas, and sets the stage for the arguments that follow. It concludes with an overview of the remainder of this document.

Chapter 1 — Context

The works presented in this thesis border on three broad subjects — contemporary choreography, both with and without the involvement of computers, artificial intelligence and computer graphics. Parts of its presentation will also touch upon computer music and user-interface design. These are fields with long traditions and many practitioners, so the work that I present and the arguments I develop in this thesis must be contextualized with respect to each of these areas. During this contextualization the central themes behind my artworks will emerge. We will see a new model, a new metaphor, for interactive art-making brought out of artificial intelligence and demonstrated in the context of dance theater; we will see how this model differs from the prevalent synthetic and analytic techniques of interactive art and dance technology; and we will begin to see what fruits this maneuver might have for both artificial intelligence, computer music and computer graphics.

1. On computation and dance

Many of the works presented in my thesis are collaborations involving choreographers. Three are live projections for dance theater — *Lifelike*, *22* and *how long...* — two of which — *22* and the central work *how long...* — are projections

for *interactive* dance theater. These projections are generated in real time, the computers "seeing" the positions and motions of the dancers using state-of-the-art motion capture technology. All three pieces are difficult and expensive, and both this difficulty and expense come directly from the presence of these computers and the equipment required for them to sense the stage so that they can be a live part of the performance.

There is an existing field, positioned between the academy and the arts, of *dance technology*, a field of artists digitally and electronically augmenting dance. This work would seem to fit inside its domain. However, I choose not to draw much grounding context from this field, and I will postpone the contextualization of it until later in this chapter — if my work fits into this tradition, it does so uneasily. Rather I shall look at the recent history of modern dance in the absence of digital "augmentation". The recent line of "interactive" works for dance and computer have failed to make much lasting impact on the dance world or the broader digital arts community. But there remains an uneasy but strong alloy of visual art, dance performance and computation that can be made. I believe the works presented in this thesis do just this, and point toward original ways of continuing this fusion in the future.

I will argue as follows: firstly, that there is surprising common ground between recent choreographic practice and computer graphics (as well as computer science), so much, in fact, that one can identify a "computational sensibility" in the work of many prominent choreographers in the last half century; secondly, that choreographic practice is one where such algorithmic concerns meet the realities, constraints, and meanings of the human body and the eyes of the audience, and as such offers a foil for the worst tendencies of technologically mediated art and a concrete platform for the best tendencies of computer science; and lastly that such a union between digital art and dance is there for the taking — the "dance-technology" work that lays claim to the space where the union would

occur has typically ignored what both computation and choreography could offer to each other.

A computational sensibility — the mechanics of generalization and abstraction, choreography as representation, dance as computation.

One could write a long history of recent dance to separate this computational sensibility out from the more general intellectualization of the art form that has occurred over the last 50 years. But in order to trace the thread of algorithmic concern through 20th-century dance, I'll focus on a set of four central choreographers whose contributions to and impact on dance is unquestionable — Merce Cunningham, Trisha Brown, Bill T. Jones, and William Forsythe. It is my great fortune that three of them — Cunningham, Brown and Jones — are collaborators on works discussed in this thesis.

A key tendency in computer science is the urge towards generalization — the replacements of constants with variables— and abstraction — the re-expression of prototypes as templates. This inheritance from mathematics is powerfully exploited to unmask problems as restatements of previously solved problems, to build generic machines that become the site of confluences of data previously considered disparate, suggesting new computations that can be carried out which in turn make for new frontiers and problems. It lies at both the heart of computer science — in the form of the general Turing machine — and at the periphery — of the everyday activities of the software engineering programmer. The flux of generalizations and abstractions of computer science should, however, not be mistaken for the totalizations of natural science. Rather than seeking a coherent, global predictive and explanatory system, the systems of computer science are forever local, transformative, interconnected.

Works referenced for Merce Cunningham are documented in the comprehensive book:

D. Vaughan, *Merce Cunningham, Fifty Years*. Aperture, New York, 1999.

they are arguably better contextualized for our purposes here in:

R. Copeland, *Merce Cunningham: the Modernizing of Modern Dance*, Routledge, 2004.

For Trisha Brown, the encyclopedic

H. Teicher (ed.) *Trisha Brown: Dance and Art in Dialogue*. MIT Press, Cambridge, MA, 2002.

is indispensable. For Bill T. Jones:

E. Zimmer and **S. Quasha,** *Body against Body: The Dance and other collaborations with Bill T. Jones and Arnie Zane*. Station Hill Press, New York, 1990.

The signature of this tendency is: a recasting of an established formal system in new, more flexible terms, that immediately produces a range of new systems; an often rapid exploration of the outcomes of these systems; a selection and categorization of some of these "instantiations" into new framework; and a resulting framework that is itself ripe for generalization.

This computational sensibility is present at two levels in the work of these choreographers. Firstly, in their choreographic processes — the systems, methods, and notations through which the choreographers create the dance. Secondly, in the finished work itself, as it appears on stage, and is interpreted by the viewer. Of course, it is a defining feature of modern and contemporary dance that the boundary between "process" and "product" is often blurred.

In the choreographic process, we can see this tendency throughout dance in obvious places: the rapidly expanded palettes of modern dance, generalizing the acceptable motion vocabulary to include the everyday, the pedestrian, even the animal. Cunningham's earliest inventions and proclamations — the democracy of the stage space, and the rediscovery of the dancer's back as a point of origin of motion — can be interpreted as generalizations of a kind; any point of a stage can be a "front", and any connected set of joints can be thought of as a limb. What were once specified constants in a rigid description become variables in in a generative framework.

As **Umberto Eco** points out in *The Open Work* (1962) from the second half of the 20th century, artists increasingly became fascinated by indeterminacy, process and open form, establishing a productive dialectic between this openness and the need to produce a "finished" work.

U. Eco, *The Open Work*, **A. Cancogni** (trans.) Harvard, 1999.

See, for example, Cunningham's comments on both "relativity" (as in physics) and animal motion in his work *Beach Birds for Camera* in: **J. Lesschaeve** (ed), *The Dancer and the Dance*, *Merce Cunningham in conversation with Jacqueline Lesschaeve*. Marion Boyars, New York, 1985.

See also: **M. Cunningham** and **D. Vaughan**, *Other animals: Drawings and Journals*. Aperture, New York, 2002.

See the particularly revealing interview with Forsythe by Paul Kaiser in Performance Research, 4 (2), Summer 1999. Available online at:

http://www.kaiserworks.com/ideas/forsythe1.htm

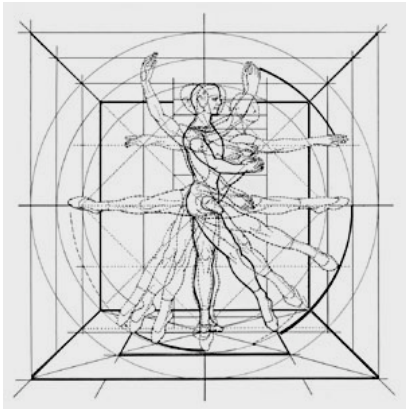


figure 1. A body centered, fixed, "kinesphere".

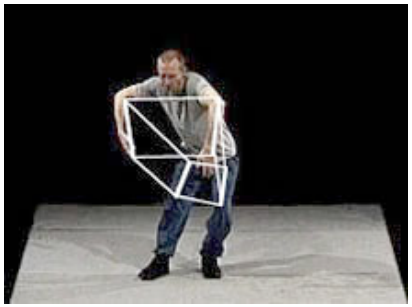


figure 2. A Forsynthian, "generalized", mobile, kine-polyhedron. From *Improvisation Technologies*, below.

But to find the most concentrated and self-contained examples of the generalization–specification cycle we turn to Forsythe's choreographic practice. Many examples drawn from his work have been *described* in the literature on the ideas behind choreography, but seldom are meta-methodological diagnoses attempted. For example, the most commonly mentioned strategy of Forsythe is his decentered kinesphere. Forsythe takes the established kinesphere of movement theorists (most notably Rudolph Laban) — a geometrical framework for the description of limb positions that forms the grounding of Laban's extensive analytical and notational techniques — and frees it from its anchor at the center of the dancer's body. This new, roaming, kinesphere, now centered on an elbow, an ear, or the midpoint between two hands, stands to inherit every analytical use to which Laban put his kinesphere; it is a *generalization* of Laban's analytical framework. Movements, within this framework, now acquire multiple explanations (disparate problems are unmasked and seen as related), new impetuses for moving are rapidly created as the ready-made machinery of Laban can be brought to bear on new joints, limbs and points (the data of dance), the palette radically expanded. Selection, categorization and reformulation then occurs as Forsythe builds new frameworks to deal again with the resulting material — systems of "alphabetization" or hidden geometry. These new representations of dance are in turn ripe for later generalization, an agglomerative cycle that is nothing less than the choreographic process. This is generalization and respecification as a computer scientist would recognize them.

This role of representation within the choreographic *process* indicates more points of connection with computer science. (The problematic issue of representation *on stage* will be addressed later.) The creation of computer programs often turns on representation — the virtual laying out of bits that represent, that stand for, an object. Just as with computer science's *generalization*, the world *representation* should be used with caution. In its basic sense representation occurs when something stands for something else. The relationship between representation and represented may be based on some kind of similarity — in this case this mimetic representation would have an iconic or "natural" relationship with the represented. However, the representative relationship may also be arbitrary — as in the relationship between signifier and signified in language. Computational representations are often arbitrary in this sense (although they are not necessarily experienced as such by the programmer or external viewer). Without the burden of strict imitation, these representations have the freedom to support computation and reconnection — in short, *transformation*. The mimetic/transformable distinction is of course not a binary opposition, but rather two poles of a continuum. Having set up this axis for the purposes of this section we shall see it problematized in later sections.

Walter Benjamin problematizes the idea of the arbitrariness of language in his short essay *On the Mimetic Faculty*, offering an idea of *nonsensuous mimesis* that exploits the mimetic faculty of humans, indeed takes it to a higher level. We might extend this notion to computational representation.

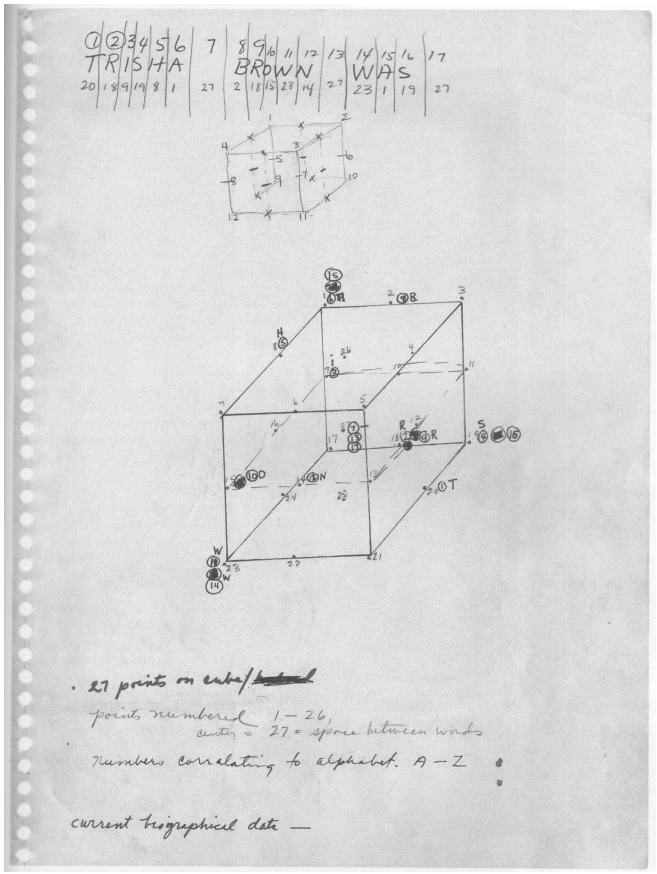
W. Benjamin, *On the Mimetic Faculty*, In: **P. Demetz (ed.)** *Reflections*, Harvest / HBJ, 1978.

Cunningham's earliest investigations of chance procedures had the flavor of computation and transformative representation about them. Motions were broken up, atoms identified, tokenized. These arbitrary tokens rearranged by the toss of a coin, the fall of the I-Ching, and the new lines and tables recast the new motion material for his dancers. Cunningham shares this style with composer and collaborator John Cage at this time, and much of their quest for new compositional strategies could be re-read as a quest for representations that support useful compositional actions.

These tendencies are articulated in: **J. Lesschaeve**, (ed.) *The Dancer and the Dance, Merce Cunningham in conversation with Jacqueline Lesschaeve*. Marion Boyars, New York, 1985.

they are "demonstrated" in: **F. Starr** (ed.) *Changes: Notes on Choreography*, Something Else Press, New York, 1968.

But a particularly simple and effective example can be found in a number of pieces by Trisha Brown in the 1970s. A *transformation* of the dancer's kinesphere



Drawings for *Locus* by Trisha Brown.

The idea of theater as the "imitation of actions" goes back at least to **Aristotle's** *Poetics* (1449b-1450a). Classical dance, with its emphasis on narrative, clearly belongs to this tradition.

into boxes, the arbitrary representation of these boxes by letters of the alphabet, the manipulation of the temporal sequencing of boxes by the creation of words and messages and the *retransformation* of these messages into movement yields a dance, a complex semaphore often intersecting with the representation's mirror — the spoken word. Since the space has been represented as a cube, new transformations (rotations and translations) of the cube suggest themselves, further interrupting this communication. This is the fundamental compositional technique behind *Locus* (1975).

Of course, what is missing inside this alphabetic representation is replaced either by the choreographer while fixing the piece (in the case of many works by Cunningham) or by the intelligence of the performer in the moment (in the case of *Locus*), faced with the almost impossible task of *computing* the results of the choreographic *program*. We as audience are presented with the act of computation itself and its negotiation with the constraints and limits of the human body.

This brings us to the presence of computational aesthetics on the *stage*, and the dance's relationship to the audience's expectations and reactions. And it is here that we should note how radical the relationship of choreographers such as Cunningham, Brown, and Forsythe with dance history has been. While the choreographies of Sergei Diaghelev, Vaslav Nijinsky and George Balanchine all expand the expressive and representational powers of classical ballet from within, many contemporary choreographers, in particular Forsythe, threaten nothing less than the three-thousand-year-old mimetic basis of theater as an imitation of an action, in their displacement of overt mimetic representation by the fruits of covert computational representations.

I think we are trying to create something life-like. a kind of autonomous form. Artificial life, so to speak, but cultural life. Fundamentally traditional arts animate people's expertise. Classical ballet, for instance, lacks its own vital force. Dance is a far more hybrid form of animation. It is like a drawing that is drawn into itself. As in the third act of *Eidos*, it is cellular autonomy; based on the same rules as every other one but each one reacting differently. It is hybrid aesthetic organization, better yet, hybrid aesthetic animation.

William Forsythe, quoted in **T. Ozaki, (P. Vigilio trans.)**, *An Interview with William Forsythe*. (availability as above).

Process spills onto the stage throughout contemporary choreography, and with it the choreographer's computational sensibility. Forsythe is infamous for issuing instructions to ensembles that recast entire choreographies extremely late in the creative process (sometimes moments before first curtain). The results of these manipulations of the systems that produced the choreographies are literally "worked through", computed, on stage in front of the audience, the intelligence of the individual dancers on display as much as their muscular memories. This image of performance as computation leads us to improvisational techniques where the relationships between parallel, often disparate, autonomous machines are negotiated live, where the performer improvises simultaneously inside, and with, a machine of their own making.

No clearer example of this tendency can be found than in choreographer / performer Bill T. Jones's piece *21* (1983) in which a fixed, circular cycle of 21 poses is acted and re-acted out, numbered and named, by the performer. After declaring (quite pedagogically, unlike Brown's *Locus*) almost all of its motion vocabulary, Jones begins to narrate while the numbered poses continue to appear and disappear, his narration, the movement of his body and the declared name of the poses all intersecting under the pressure and the limits of the structure of the piece and the abilities of the performer to negotiate their connections.

Here we reach the physical, rather than formal, limits of this "computational sensibility" in modern dance: the limits of what the human body is capable of performing, the limits of what choreography can be. For all the fecundity of the "computational sensibility" outlined here, none of these techniques or inventions exist independent of a body and a theatre space. Cunningham's proposal that there are no fixed points in space meets the plain fact that the audience sits in one place, and the edge of the stage is all too fixed; his democratic use of the movement resources of the body pushes but goes no further than the limits of bipedal balance. Brown, in the piece *Man walking down the side of a building*

(1970), produces a choreography that disrupts the audience's sense of orientation yet leaves the mechanics needed by humans to defy gravity (the harness and rope) exposed for all to see . In *Accumulation plus talking with water motor* (1978/1986) she shows, while simultaneously talking and dancing, a choreography at the limit of memorization of narrative and of movement. In *Homemade* (1966) the performer has her movements amplified by the film projector she carries on her back in a dance of light, but it is a heavy, obvious and almost domestic burden.

I believe that the creative potentialities of the dialogue between computers and choreographers lie in this shared computational sensibility. Digital artists can connect to, and radically expand, the vocabulary of the choreography that I have outlined. For are they not experts of generalization, representation and, if not computation as performance, surely the performance of computation? In exchange, choreographers and performers are experts of the negotiation between the abstracted, transformed, and mechanical, between the theatrical, human, and perceived. They are the experts of navigating the moments when a cold algorithmic idea meets a warm body, the limits of performance, the limits of an audience, the limits of from. They can offer this crucial expertise in return for the digital artist's computational virtuosity, and the exchange itself ought to impact not just a field of technologically augmented dance, but the creative used of digital tools itself.

Dance technology theorist Scott deLahunta fears that choreographers have been backing away for decades from live performance technologies that are otherwise being integrated into theater. I hope that the work presented in this thesis offers an counterexample.

S. deLahunta, *Virtual Reality and Performance*, PAJ: A Journal of Performance and Art 24.1 (2002) 105-114

Information about *Lifeforms's* intentions can be found in: **T. W. Calvert, A. Bruderlin, S. Mah, T. Schiphorst, C. Welman,** *The Evolution of an interface for choreographers.* Interchi'93 — ACM Press, April 1993.

See, for example, Hotwired's ecstatic reporting of Cunningham's use of the computer:

http://hotwired.wired.com/kino/95/29/feature/index.html

as well as countless post-performance interviews and discussions by Cunningham.

Earlier attempts at this dialogue were often unsatisfying. For example, as has often been repeated in the press, Cunningham has been using the *Lifeforms* software for more than a decade now as part of his choreographic process. Ironically, however, such software is entirely concerned with the appearance of the virtual human figure, its technical concerns imported wholesale from linear key-frame animation rather than offering any computational support to the choreographer. The tools all lead down the path of least technical resistance — the commodity hardware of computer video, the tried and tested hyper-mimetic representations of photorealistic "Hollywood computer graphics". This use of the computer seems oddly disconnected from the underlying computational practice that I have identified in his work. Despite his now expert use of computers to find shapes that he can no longer find on his own body, Cunningham still throws his own dice.

Of course, it would be just as unsatisfying to simply make a computer program that helps Cunningham roll dice — this duplication of the choreographer's role is unnecessary. We have to find some other way to create a programmer-choreographer dialogue. To inform the technologies and practices that we develop for the sake of this dialogue we clearly need to cast a net wider than contemporary choreography or even dance technology. In this regard the tools and techniques of computer music are fundamental to my approach.

Dance notation continues to generate conferences and discussions but little permanent consensus. For a glimpse at the rather more interesting, less academic, personal notations of choreographers:

L. Louppe, B. Holmes, *Traces of Dance*, Dis Voir / DAP Publishers, New York, 1994.

A compendium of mid-century musical notations: **J. Cage**, *Notations*, Something Else Press, 1969.

Over the centuries, composers have developed notational representations that allow the distribution and reproduction, but more importantly the transformation and interpretation of music, without the instantiation of sound. A focus on the experimental transformation of music into a representation, on manipulations within and with this new form, and on subsequent reinterpretation and retransformation back into other sounds or other representations can be found all along the border between computers and music — sound synthesis techniques, compression algorithms, set-theoretic compositional strategies or new instrument design. Computer science's representations and music's notations are not just ways of seeing the world or music but locations for new ways of thinking about how to change it. Since this ground has been so fertile in the past, one of the central techniques in my work is to import the techniques and approaches of computer music into a new "computer dance" domain, *282*

But even outside of computer music *per se*, these exchanges are present throughout the history of music. The ascendancy of the twelve-tone row in Western music was propelled initially, I suspect, by the explosion of formal possibilities that this transformative unit caused. Its mid-century crystallization into an attempted totalization of musical form is opposed by a simultaneous notational explosion amongst experimental composers.

The experimental / avant-garde distinction made by composer and musicologist Michael Nyman in describing this moment of music history, and the computational representations of computer music in general, will also help us calibrate our relationship to the "formalisms" of contemporary choreography. Nyman offers a distinction between "avant-garde" composers (the Boulez of the 1950s) who search for coherent systems, self-contained and self-constraining; and the "experimental" tendency (typified by Cage) concerned not with "prescribing a defined time-object", but rather "outlining a *situation* in which sounds may occur, a *process* of generating action (sounding or otherwise), a *field* delineated by certain compositional 'rules.'"

M. Nyman, *Experimental Music: Cage and Beyond. 2nd ed.* Cambridge, UK: Cambridge Univ. Press, 1999 (p. 4).

In this analysis we see Forsythe, Brown, Jones and Cunningham allied most definitely with the experimental — sharing practices that "work through" a field delineated by temporary "rules", mining the potential latent in algorithmic systems as performed by their dancers, while temporarily protecting the integrity of the system from reproach. Only after the consequences of these computations have been discovered are these tactical formalisms aggressively questioned, toppled, robbed of any governing authority over practice as a whole. We will see a very similar practice developed in this thesis as I explicitly locate techniques that permit the development of algorithmic potential, and computational representations open to the unexpected, that simultaneously permit the navigation and culling of the resulting computational space. In the creation of autonomous "live" digital artworks, this method of working, which is at once ludic and serious, is at the core of my aesthetics.

In this regard we might also look to the "formalisms" of the French literary "workshop" *Oulipo*. In describing the relationship between novelist/mathematician Raymond Queneau and mathematics, author Jacques Roubaud describes the Oulipian imitation of the "axiomatic method" in the writing of literature. Reacting against the surrealist obsession with literary "freedom", Roubaud rejects the "mystical belief according to which freedom may be born from the random elimination of constraints." However, the Oulipian of using constraints to generate texts claims no ultimate authority, since literary rules no longer have any foundation in value.

J. Roubaud, *Mathematics in the Method of Raymond Queneau*, reprinted in: **W. F. Motte Jr** (trans., ed.), *Oulipo a primer of potential literature,* University of Nebraska Press, 1986 (p. 88, 89, 93).

2. On "mapping"

The field of dance technology — the use of computer and electronics in a dance theater context — is undeniably growing today. However, this is a field with many practitioners, few techniques and almost no theory; a field that is gener-

ating "experimental" productions with every passing week, has literally hundreds of citable pieces and no canonical works; a field that is oddly disconnected from modern dance's history, pulled between the practical realities of the body and those of computer art, and that has no influence on the prevailing digital art paradigms — largely taken from computer music — that it consumes.

If there is a term that tries to pass as a central concept in the theory of interactive digital artworks in a dance context it is *mapping*. I shall argue that this word has become vague almost past the point of usefulness, but at its core is a reference to the ability of digital computers to take data from one domain and transform it into another. This transformative core of interactive digital artworks is also the location of its "visualization" and "sonification" tendencies. This thesis proposes and contrasts an alternative point of origin for digital artwork: the interactive agent. But to attempt this contrast, and to contextualize the interactive dance work that is presented later, we will have to spend some time discussing the meanings that "mapping" has for the community.

Despite the term's imprecision, one can hardly cite a technical paper on dance technology without encountering the word. A representative example:

> Each part of the body has its unique limitation in terms of direction, weight, range of motion, speed and force. In addition, actions can be characterized by ease of execution, accuracy, repeatability, fatigue, and response. The underlying physics of movement lends insight into the selection of musical material. Thus, a delicate curling of the fingers should produce a very different sonic result than a violent and dramatic leg kick, since the size, weight and momentum alone would have different physical ramifications. To do this, physical parameters can be appropriately **mapped** to musical parameters, such as weight to density or register, tension to dissonance, or physical space to simulated acoustical space, although such simple one-to-one correspondences are not always musically successful. The composer's job then, is not only to map movement data to musical parameters, but to interpret these numbers to produce musically satisfying results. [emphasis added]

From **T. Winkler**, *Making motion musical: Gesture Mapping Strategies for Interactive Computer Music.* Proceedings of the 1995 International Computer Music Conference. San Francisco, International Computer Music Association, pp. 261-4.

However, more recently: **T. Winkler**, *Live Video and Sound for Dance.* From Video, Technology and Performance Festival, Brown University April 4-5, 2003. Available online at: http://www.brown.edu/Departments/Music/faculty/winkler/papers/

In what is in many ways the parent field of dance-technology, interactive music controller design, researchers talk of *mapping* sensor data to musical parameters, of the *mapping* problem, of classes of *mappings*, of good *mappings* and bad *mappings*, of intuitive *mappings* and unsuccessful *mappings*, of tools for *mappings*. Some 40% of the papers in the 2004 New Interfaces for Musical Expression conference use the term in all sincerity, as part of titles, abstracts, conclusions, problem statements and results. Here and elsewhere, mapping has become a analytical perspective and a methodology, an point of departure and a destination, a field of study, a description of a problem and a place where solutions are to be found.

$$\text{output} = f(\text{input})$$

This then is the core image for a whole branch of interactive music and much of the smaller field of interactive dance technology and, almost as if through infection by the vector of their shared tools, interactive art as a whole.

My highlighting of this term and its problems is in no way meant to detract from the artists and engineers who have struggled to find new sensors, new data and new places for these data in artworks. Rather it is intended to indicate the need for a greater range of vocabulary, for greater nuance, for describing this very struggle and organizing the intellectual field around it.

The step after hardware:

Raw values are received by Max via the VNS object, an object written by Rokeby to handle system configurations. From there, changing values representing the grid are displayed graphically, then scaled, mapped, or otherwise prepared to enter the system's response modules.
T. Winkler, *Motion Sensing Music*, Proceedings of the International Conference on Computer Music 1998, San Francisco, International Computer Music Association.

The mapping had to be both transparent to the user and complex enough to sustain interest if the system were to be used day after day. In our process, we took a top-down approach to mapping [...]
L. Gaye, R. Mazé, L. E. Holmquist, *Sonic City: The Urban Environment as a Musical Interface*, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Or the entire performance problem:

Our work on instrument design and instrumental performance interfaces has led us to consider in detail the mappings from the performer's gesture space to the listener's perceptual space.
G. Garnett, C. Goudeseune. *Performance Factors in Control of High-Dimensional Spaces*. Proceedings of the International Conference on Computer Music. 1999. San Francisco, International Computer Music Association.

Movements are identified and mapped in software to play and process sounds (Max/MSP), or to alter a live video feed using real-time video processing software (NATO). The computer generates most of the material based on the performer's movements, with each performance being a unique realization of the program's many potential responses.
T. Winkler, *Live Video and Sound Processing for Dance*, Video, Technology and Performance Festival,Brown University April 4-5, 2003. Paper available online at: http://www.brown.edu/Departments/Music/faculty/winkler/papers/

A prescription:

... there should be a correspondence between the size of a control gesture and the acoustic result. Although any gesture can be mapped to any sound, instruments are most satisfying both to the performer and the audience when subtle control gestures result in subtle changes to the computer's sound and larger, more forceful gestures result in more dramatic changes to the computer's sound.
D. Hewitt, I. Stevenson, E-mic: Extended Mic-stand interface controller, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

The underlying physics of movement lends insight into the selection of musical material. Thus, a delicate curling of the fingers should produce a very different sonic result than a violent and dramatic leg kick, since the size, weight and momentum alone would have different physical ramifications. To do this, physical parameters can be appropriately mapped to musical parameters, such as weight to density or register, tension to dissonance, or physical space to simulated acoustical space, although such simple one-to-one correspondences are not always musically successful.
From T. Winkler, *Making motion musical: Gesture Mapping Strategies for Interactive Computer Music.* Proceedings of the 1995 International Computer Music Conference. San Francisco, International Computer Music Association.

Or a vista of possibility:

More furious and strenuous activity, for example, could result in quieter sounds and silence. At the same time, a small yet deliberate nod of the head could set off an explosion of sound. Such "unnatural" correlations makes motion all the more meaningful.
ibid.

Objects such as a coffee mug can be instrumented and interactions with them mapped to sounds.
R. Hoskinson, K. van den Doel, S. Fels, *Real-time Adaptive Control of Modal Synthesis*, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Several mapping metaphors were explored; e.g. tongue position was used to play a physical model of the singing voice.
M. J. Lyons, M. Haehnel, N. Tetsutani, *Designing, Playing, and Performing with a Vision-based Mouth Interface*, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

The system must be flexible in respect of providing unlimited mapping arrangements.
D. Hewitt, I. Stevenson, E-mic: Extended Mic-stand interface controller, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

What these principles are meant to address is that the programmability of computer-based musical systems often make them too easy to configure, redefine, remap, etc. For programmers and composers, this provides an infinite landscape for experimentation, creativity, writing papers, wasting time, and never actually completing any art projects or compositions.
P. Cook, *Principles for Designing Computer Music Controllers*, ACM CHI Workshop in New Interfaces for Musical Expression (NIME), Seattle, April, 2001.

The tablet we use allows for simultaneous sensing of two devices, usually one in each hand. This rich, multidimensional control information can be mapped to musical parameters in a variety of interesting ways.
D. Wessel, M. Wright, *Problems and Prospects for Intimate Musical Control of Computers*, Computer Music Journal, 26 (3), MIT Press, 2002.

- The difference between success and failure, for the performer:

  One or two reported on their check for causality between mouth action and aural effect: they found it sometimes easily visible but quite obscure at other times. This appeared to be mainly a function of the mapping.

  M. J. Lyons, M. Haehnel, N. Tetsutani, *Designing, Playing, and Performing with a Vision-based Mouth Interface*, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

- and for the audience:

  The two primary goals of the mapping process are firstly to have a satisfying communicative relationship from an audience perspective and secondly to create a workable relationship from a performers' perspective which meets the requirements for satisfactory control of the sound source and allows high level performance skills to be developed.

  D. Hewitt, I. Stevenson, E-mic: Extended Mic-stand interface controller, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

  While in traditional acoustic instruments the effects of the performer's physical activity on an instrument are already established by the physical properties of the instrument, in electronic instruments this relation must be previously designed. Mapping this relation can be critical for the effectiveness of an electronic instrument. [...]

  The absence of a unique gestural mapping prevents the performer from deeply exploring the system's controlling mechanisms at the same time that it prevents the listener from connecting visual input and music.

  F. Iazzetta, *Meaning in Musical Gesture*, in: *Trends in Gestural Control of Music*, **M. M. Wanderly** and **M. Battier** (eds.) IRCAM, 2000.

- The central problem that the artist faces:

  We emphasise the importance of the mapping between input parameters and system parameters, and claim that this can define the very essence of an instrument [...] Moreover, the psychological and emotional response elicited from the performer is determined to a great degree by the mapping.

  A. Hunt, M. M. Wanderley, M. Paradis, *The importance of parameter mapping in electronic instrument design*. Journal of New Music Research 23(4) 2003.

  Just as the subject of a fugue must be thought out for its potential for future exploration and expansion, here too, the composer is challenged to find musical gestures that serve the dual purpose of creating melodic interest while generating a function applicable to signal processing.

  T. Winkler, *Interactive Signal Processing for Acoustic Instruments*, Proceedings of the 1991 International Computer Music Conference. San Fancisco, International Computer Music Association.

- The solution endlessly deferred as future work:

  The next stage in the process is to develop workable mapping strategies and to implement the compositional process.

  D. Hewitt, I. Stevenson, E-mic: Extended Mic-stand interface controller, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

  The opportunity and challenge of this system is to devise strategies for mapping so very many degrees of freedom into a meaningfully expressive whole.

  C. Dobrian, F. Bevilacqua, *Gestural Control of Music Using the Vicon 8 Motion Capture System*. Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

  Since my sound processing software is in continual development, no definite mapping scheme is in use yet.

  C. Palacio-Quintin, ***The Hyper-Flute,*** Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

  However, even with more meaningful feature extraction,finding compelling mappings for the output of such a system will continue to be a challenge.

  D. Merrill, *Head-Tracking for Gestural and Continuous Control of Parameterized Audio Effects*, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

  Current work with the Metasaxophone involves continued exploration of extended mapping possibilities for physical models.

  M. Burtner, *The Metasaxophone: concept, implementation, and mapping strategies for a new computer music instrument.* Organised Sound 7(2): 2002.

  A lot of the ongoing work on the visual feedback is going to be included into the working prototype in the near future and we have been working intensively on the object design and mapping issues, which will also be reflected in the final instrument design.

  M. Kaltenbrunner, G. Geiger, S. Jordà, *Dynamic Patches for Live Musical Performance*, Proceedings of the 2004 Conference on New Interfaces for Musical Expression.

  Mappings allow for any sound to be mapped to any input arbitrarily, and the extreme freedom and range of possibility makes it hard to construct mappings that look and sound "real" to an audience. It is still not well understood how to construct mappings such that they intuitively map well to an action; this is because interactive music is still an extremely new art form.

  T. Marrin Nakra: *Synthesizing expressive music through the language of conducting.* Journal of New Music Research. 2002, 31 (1). 2001.

Of course, mathematicians have stretched the potential field of meaning of the above equation beyond all the horizons that we can see from here; almost anything could be written as the above definition. But if this definition has no limits it has no use. In practice one can sense in this "function-like" aspect of mapping is a kind of college-level, piecewise linear or otherwise smooth, locally stationary, state-less, typically decomposable relationship between input and output. Such a vision acts as a normative idea of how, in this field, numbers get transformed into numbers. The best work in the field, of course, pushes against this central tendency, but the rules and arena remain fixed.

This term, and the ideas it accretes, spans decades. The source of the lasting power of this description of interactive art is hard to locate exactly. It would be tempting to suggest that it is residue from the early century tropes of synesthesia, and eurythmics interacting with the purely technical possibilities of the "multi-media". Perhaps it is a weak theoretical echo of the color organ or the Theramin. More likely, however, is that it was brought into the field from analogue music synthesizers and never left, reinforced, as we will suggest later in this work, by the tools and environments used and promoted by artists themselves. These tools continue to suggest that the interchangeability and equivalence of the digital signal has in some way a predictive or explanatory power over the relationships between disparate media.

For a survey of synesthesia from both an artistic and biological perspective:

R. E. Cytowic, *Synesthesia: A Unity of the Senses,* New York: Springer-Verlag, 1989.

In dance technology it is not hard to find statements concerning mapping that simply rejoice in the potential of digital tools:

Continued from above: **T. Winkler**, *Making motion musical: Gesture Mapping Strategies for Interactive Computer Music.* Proceedings of the 1995 International Computer Music Conference. San Francisco, International Computer Music Association, pp. 261-4.

> By being aware of these laws, it is possible to alter them for provocative and intriguing artistic effects, creating models of response unique to the computer. More furious and strenuous activity, for example, could result in quieter sounds and silence. At the same time, a small yet deliberate nod of the head could set off an explosion of sound. Such "unnatural" correlations makes motion all the more meaningful.

These articulations are often no more complex than: if the dancers move quickly the music gets louder, or that the bass notes are blue and the treble red. Unless one has unshakeable faith in a broad, universal synesthesia or a natural order of relationships, these function-like statements are equally *meaningful* when inverted: the dancers move quickly and the music gets softer.

The community itself is turning against the term: for example, the New Interfaces for Musical Expression Keynote in 2002:

J. Chadabe, *The Limitations of Mapping as a Structural Descriptive in Electronic Instruments*. Proceedings of New Instruments For Musical Expression, Dublin.

The term "mapping" is clearly outliving its usefulness and its predictive and explanatory power has long left us. If this "map-ism" is deployed as a metaphor, what does it metaphorically connect with? Are there interesting physical systems that are satisfactorily read in this way? Do any of the natural analogues that researchers are also interested in map anything? what part of a flute transforms concrete, quantized measured data? what part of the audience manipulate a stream of readings? If we are interested in *inter*action, why start with a formula that goes only one way? If it is only a metaphor, why then is it embodied directly in *data-flow* interfaces and underling architectures of common digital art tools? The agent metaphor, developed in this thesis in a manner of particular use to art-making, stands directly opposed to mapping in this most banal sense; and I believe it to be of more use than the term in its more diffuse applications.

At the very least it will allow access to interactions that this function-like stance does not. Indeed, the agent's very autonomy acts to prevent a deeply penetrating analysis input to output from having any long term success — in describing the behavior of an agent — or synthetic utility — in thinking about how to build an agent to do something. The complication, and this opposing agent "metaphor", helps illuminate the roots of mapping's troubles. One weakness stems directly from the flattening of detail, inherent in words that populate the descriptions of maps: "move quickly", "louder", "bass", "blue", that comes just prior to tying these surface properties together. This is a *category error* perpetrated by the artist on their own thinking and practice — confusing a way of measuring or a way of controlling something with the thing itself; confusing part of the effect

(appearance) for the totality of the cause (process); confusing a particular control surface (the volume knob) or a particular derived quantity (say, the sum of distances divided by times) with a more internal structure of the process and the context which to my eye is never flat, never just a number waiting be plucked from nature by hardware.

On the one hand this is a particularly surprising mistake for the digital artist to make, for unlike the scientist or engineer who takes nature as they find it, they have at least partial responsibility for both the surfaces — the controls and the viewpoints — and much of the thing being controlled or viewed . Be it the view from psycho-physics, computer science, or digital art itself, these simple numbers and parameters are only byproducts of selected solutions, not the givens of any particular problem domain. On the other hand it is an understandable strategy. In quickly binding "sensor" to "output" inside a digital setting, mapping deflates the awesome *potential* of the algorithmic before it can appear. The space of algorithmic relationships is slowly and safely explored on a scaffolding of one sensor to output thread at a time. The vertiginously parametric opportunities of digital tools are both the object of fascination of the digital art world and its greatest fear. They are, in much of the community's work, collapsed and hidden from view by its very conception of the problem.

Where the connective statements of maps do have an importance is *either* in the micro-scale of the hardware and software that executes a work *or* in the broad strokes of a preliminary sketch. Pieces of hardware and software must and do pass numbers between themselves — but the days have long passed when there were efficiency or protocol problems that put this level of discussion at the fore of this field's theorizing. Developers, collaborators must and will pass general ideas around concerning what might happen and when and will make such broad connective statements — but, again, have we not moved beyond a time when these connections could justifiably mark the end rather than the begin-

ning of a discussion? The field of potential is too large to be explored armed only with these statements and the work is too difficult for them to be of much lasting use. Mapping should be receding in digital art's rearview mirror, not as a solved or exhausted problem, but as an idea either too small or too broad to really fit with the tasks and the opportunities at hand.

Toward the agent

However, underneath this possibly falsely unifying term, there is an interesting and relevant story underway in the literature, which again our competing agent metaphor helps diagnose. I possess the following suspicion about the development of mappings: that as we seek to build better mappings, we are led from the simplicities of "complete specification" — the connecting the wire between input and output — down two divergent routes with inevitable termini. The first of these is better interfaces for complete specification, ones that yield *faster* ways of exploring that space of wires. The second is, sometimes disguised, unsupervised machine learning — to give us *easier* ways of describing the mappings of space. The former path leads to the environments that dominate today — advanced tools for trying out a relationship, discarding it, tuning it, trying another. These are the tools that are commodified, taught in schools and have had to date more permanence than most of the art made with them. They allow the working artist to confront the space of possible mappings, to confront the potential devel-

I believe that many tools today fall into the category of allowing artists to try more things faster. For example, the *Max* series of graphical environments: http://www.cycling74.com.

As for the emerging trend towards using unsupervised techniques for "advanced mapping", a few recent examples: **A. Cont ,T. Coduys , C. Henry**, *Real-time Gesture Mapping in Pd Environment using Neural Networks*.

and, **J. Mandelis** and **P. Husbands**, *Don't Just Play it, Grow it! : Breeding Sound Synthesis and Performance Mappings*

Both are from the Proceedings of the 2004 conference on New Interfaces for Musical Expression, Hammamatsu, Japan.

R. Bencina, *The Metasurface – Applying Natural Neighbour Interpolation to Two-to-Many Mapping*. Both are from the Proceedings of the 2005 conference on New Interfaces for Musical Expression, Vancouver, Canada.

figure 3. A depiction of the trend towards "smarter mapping" found in the academic literature of interactive art. What comes after this piecemeal approach to "smartness" ?

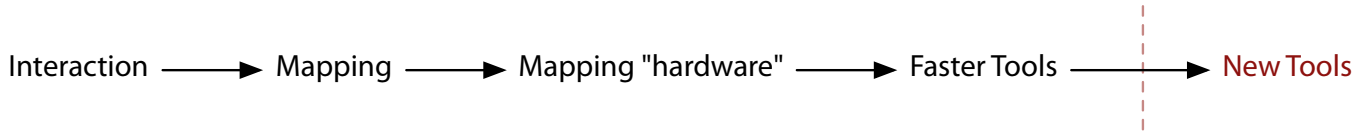


figure 4. A depiction of the trend towards "faster mapping". The dominant tools available today are constructed by analogy with the early "hardware" of interactive art and concentrate on alowing the artist to try a large number of alternative mappings. What comes after the simple speed and flexibility of software over hardware?

oped by digital manipulation by trying out *more things* more quickly. The latter path leads toward nothing less than supervised machine learning — environments for *training* mappings, and *inducing* them out of interactions. These, rather more niche and rather more academic ideas, seek to allow the working artist to confront that unknown space of potential by trying out *smarter things* and by navigating around the space in smarter ways. Ironically, we might say that these tools seek to actually provide a useful map for the space of mappings.

As this latter thread becomes more developed, and its systems meet the realities of rehearsal and distribution as well as the opportunities afforded by complex assemblages of code and more sophisticated artistic intentions, I believe that it will end up squarely in the domain of artificial intelligence.

There is a story here that goes in an altogether different direction from art back towards the biological:

M. Whitelaw, *The Abstract Organism: Towards a Prehistory for A-Life Art*, Leonardo Vol. 34, No. 4, 2001.

and indeed my thesis shares a broad sensibility with this article's primary text:

P. Klee, *The Thinking Eye*, George Wittenborn, NY, 1961.

A more sustained reflection on this artist's relationship to contemporary art practice can be found in:

P. Boulez, *Le Pays Fertile: Paul Klee*, Editions Gallimard, Paris, 1989.

Artificial intelligence, as articulated by its pioneers, is nothing less than the task of getting computers to do the "right thing" — despite our inability to describe in the kinds of ways that computers prefer what the "right thing" is, or at the kinds of detail that computers demand what the operating environment will be. AI is thus the study of and construction of new ways of articulating how systems should behave given a higher level, a more human level, a more convenient description of the desired behavior and the environment in which they will operate.

This thesis starts at the opposite end of this reading of modern interactive digital art. Rather than start with mapping in micro or macro and move toward either art environments or academic artificial intelligence, it starts with both AI and the tools for art-making, and heads towards interactivity, "multimedia" transformation and connection.

Of course there is a reason why this direction is against the flow of the community. Making live interactive programs that are artificially intelligent is a difficult and obscure pursuit, and only recently has it made sense to move away from solely focusing on increasing the scope of what computers can do (the size of the potential field) to devote a little time to considering the practice of making them do it (how that field is navigated). Digital artists need new ways of conceiving their digital methods in order to take advantage of these opportunities. I also believe that while I "demonstrate" this point in a few fields, confined primarily to interactive art, the importance, and potential impact, of this navigation from AI towards meaning-bearing relationships is much broader than this, and indeed might be as broad as digital and algorithmic design itself. This broader context will be indicated most strongly when I focus on the tools that I have constructed partially in response to the difficulties of transferring my chosen approach into my chosen fields.

This engineering, or authorship perspective, is given a broad manifesto in: **R. Brooks**, *Intelligence without Representation*, Artificial Intelligence, 47 (1991) 139-159.

For an emphasis on this human-level *description*, for example:

"Artificial intelligence is the science of making machines do things that would require intelligence if done by men"

from **M. Minsky**, *Semantic Information Processing*, MIT Press, 1968.

This is accompanied by, but not indistinguishable from, an emphasis on *human-level* problems in that work. My thesis work breaks with any remaining AI tradition of human *replacement* and instead focuses on an AI thread of human *augmentation* in the broadest sense — augmentation by both the artifact (the finished artwork is not something that I could create by hand) and the techniques for creating the artifact (the ways in which that "finished" is defined and found are inconceivable without the AI).

By starting where I believe the field is heading, almost inevitably, in a piecemeal fashion and by doing so in a way that is open to influences and problems across both computer music and computer animation, I hope to be able to create new classes of artworks, new classes of experiences, in new ways. To make the above stratagem realizable, both AI and digital tools need significant and careful navigation and revision. That is what this thesis sets out to start.

3. The agent

Work presented in this thesis will therefore take the AI community's concept of *the agent* as its central organizing principle and offers this as a replacement for dance technology and interactive arts *mapping*.

Before sketching its relationship with the history of AI, we should begin with the pioneers of the agent-based. Brooks cites four hallmarks of this *nouvelle AI*, or what we are calling here agent-based style of work: **situated** — the boundary between the agent and the world is porous, with the world directly influencing the system; **embodied** — the agent acts upon the world and immediately senses itself acting; **intelligent** — as acting in the world as far as judged by outside observer; **emergent** — this intelligence is not confined to particular computational engine, nor is responsibility for the external action located in one particular place but arises out of the agent's interaction with the world.

The decomposition is from **R. Brooks**, *Intelligence without Reason*, MIT AI Lab Memo 1293.

The decomposition of action-selection details is from **R. Brooks**, *Challenges for Complete Creature Architectures*, In Proceedings of the first international conference on simulation of adaptive behavior, Paris, France, 1991.

This is similar to the set of concerns given by: **P. Maes**, *Modeling Adaptive Autonomous Agents*, Artificial Life, 1 (1), 1994. pp. 135-62.

Elsewhere in the literature we see the terms *behavior-based* and *interactionist* to refer to this style of artificial intelligence practice. For our purposes here these terms are indistinguishable and merely serve to re-indicate the emphasis on finding an AI practice that is focused on the how the agent acts in, on *and* with the world.

In *software* agents, which are the only ones discussed in this work, we expand upon the definition of the "world" to include software worlds (although for interactive systems these worlds are in turn connected to ours), and we expand upon the definition of "body" to include software bodies — control structures that operate on material that can be rendered graphically. Indeed, part of the contribution of my work is to push the agent-based approach into new worlds (dance theater, and to a lesser extent computer music) and new bodies — musical bodies and non-constant, non-figurative bodies.

We will need some more terms, and a more concrete model of an agent to proceed through this. One segmentation that I propose, which is both generic and useful, decomposes the software agent into three coupled systems, which we shall label as the **perception system**, the **action system** and the **motor system**. I believe this picture can be read into, if not read *from*, much agent-based work — it is a generically descriptive decomposition of many practitioners' agent architectures.

Describing the "contents" or the fields of competences for each of these systems will be a task undertaken throughout this document. However, some starting points are of use.

The **perception system** of an agent is the area that takes the world as it finds it and begins to transform it into a form more convenient for the creature. Often this is where measurements become symbols, where raw sensations given in what form hardware and the world can offer become categorized, or at the very least scaled, filtered and perhaps fused. In *Dobie*, a synthetic dog that can be trained in many of same the ways that a real dog can, the perception system holds onto and adapts models of spoken commands ready to classify the incoming speech from the trainer; in *how long...* many agents' perception systems try to follow dancers from the stage despite the presence of noise and missing information. The flow of control, update

Note that in: **M. Minksy**, *Society of Mind*, Simon & Schuster, New York, 1988, We see a broader, more inclusive and more abstract use of the term agent — his *Society of Mind* is a radically heterogeneous society of mindless agents, none of which have the scale of, or the same structure as, the normative agent model described here. We shall start with this large, monolithic agent description and move towards the heterogeneity of Minsky's society, but we will keep the term agent for the "creature-as-a-whole"

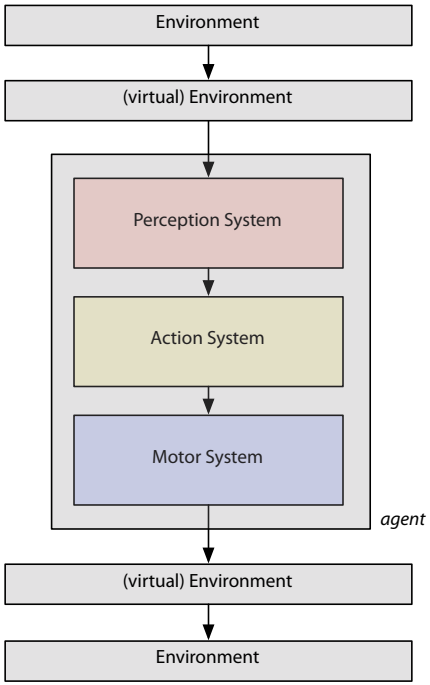


figure 5.
The agent, for our definition here, is decomposed into three parts — perception, action and motor systems.

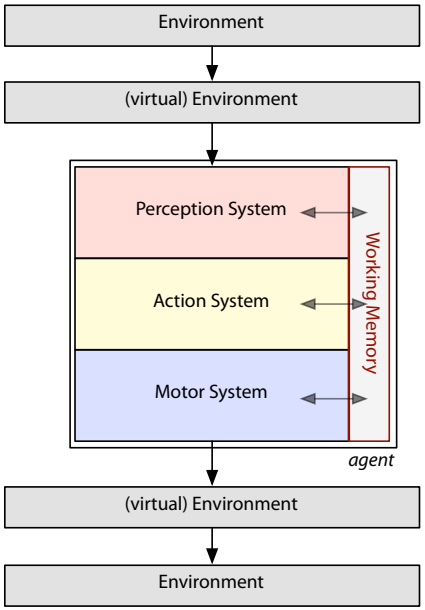


figure 6.
Additionally, in the systems developed for this thesis, we add an additional infrastructural "system" for the purposes of internal communication

and activation of a perception system is only partially under the "control" or the autonomy of the agent, and it is necessarily shared with the moment-to-moment changes in the world.

The **motor system** of an agent is the area that coordinates the body's relationship to the world. Often, this is where the commands of the action system meet the constraints of animation and the constraints of the world. It is most often the site of expectations about how the body should move in and interact with the world, and ongoing monitoring of how progress is being made. In the case of graphical (and musical) agents it is where premade material, often on its own terms, enters the agent. This material is spliced, blended and layered to synthesize the manipulation of the agent's body. In *Dobie*, the motor system splices, blends and layers from a library of pre-made, hand-made dog animations; in *how long...* agents with no natural analogue often find and integrate their motion material by sampling movement from the stage. The flow of control, update and activation inside a motor system is only partially under the command of the action system of an agent, and only partially under the command of imperatively written code made ahead of time. This control is necessarily shared with constraints of the agent's material and the uncertainties of the body which it controls and world in which it acts.

The **action system** selects the actions to perform based on the perceptual state and the state of the motor system and articulates these selections to the motor system. Using the language of Brooks an action system is usually judged by three criteria: **salience** — are the actions appropriate and relevant to the context?; **coherence** — do the actions make sense to an observer over time?; and **adequacy** — are the actions *in toto* sufficient to get the creature to achieve its goals?. We can fine-tune these criteria from the point of view of an author of an agent: salience — has the creature, in integrating its perceptual world, taken advantage of the correct aspects of

> world, or responded to the unexpected in the correct way?; coherence — does the temporal patterning of the actions chosen amount to something?; and adequacy — has the creature enough actions, and enough competence to explore and modify its actions to yield the desired long-term behavior?.

This diagram also hints at the "execution cycle" of each of these systems. Often it is translated directly into a sequential update of perceptual, action and motor systems in order to complete one "evaluation" of the agent. It is further, but rather confusingly in this presentation, the place where the terms "bottom-up" and "top-up" act. In our case here the sense of gravity is reversed, and bottom-up, or the data-driven, enters through the perception system from the top, and the top-down, or agent-driven, is exerted from the core outwards.

Although the predominant flow in this diagram is from top to bottom, there is much, in a complex creature, that goes the other way. Perceptual states can be created from proprioception of the state of the action system, the state of the motor system or the state of the body itself. In action systems that support the learning of new relationships, it is the action system that guides the perceptual development. Such is the confusion, at this level of discussion, of the flow of communication between these systems that we draw an alternative diagram, with an additional box, labeled "working memory". This area in the systems that are built for this thesis is where much of the complex communication occurs, and forms a better description of the systems as implemented than the alternative tangle of arrow.

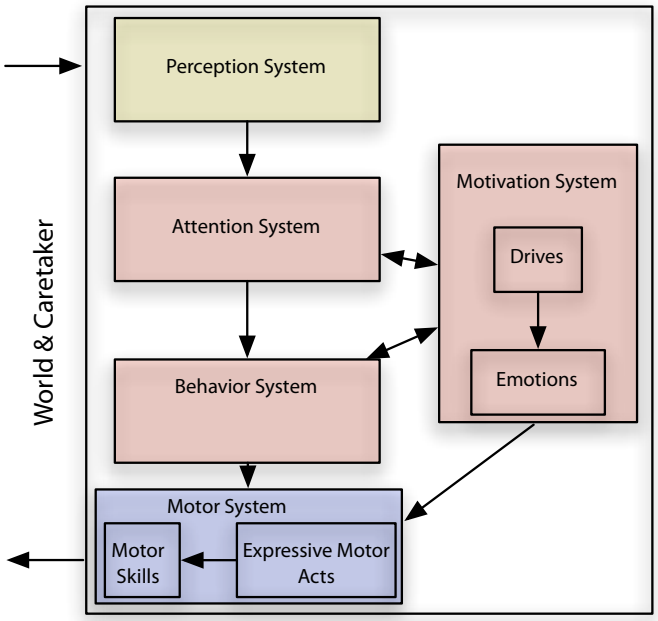


figure 7.
In the interactive robot, Kismet, Breazeal et al. divide the internal mechanisms as above. Coloring as per *figures 1* and *2*. From **C. Breazeal**, *A Motivational System for Regulating Human-Robot Interaction*, Proceedings of AAAI-98.

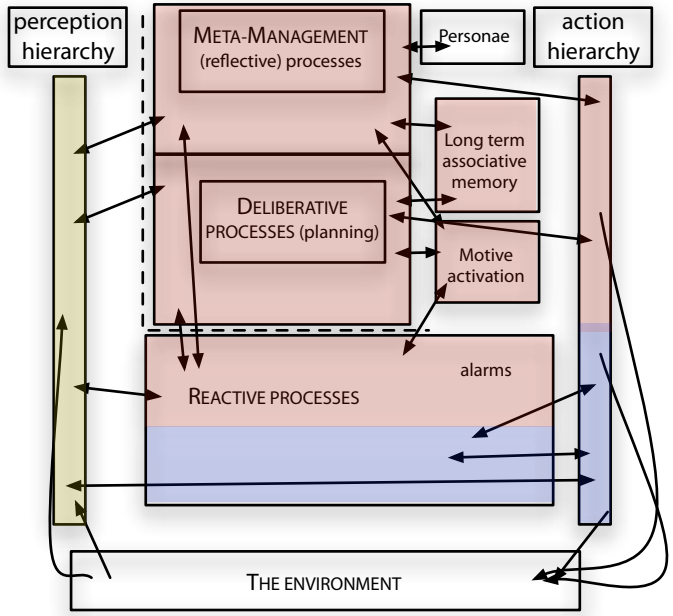


figure 8.
Further afield, Aaron Sloman's more "human level" artificial intelligence framework still permits a similar decomposition. Note that in this work, as is typical with such architectures, the space devoted to motor system issues is vastly reduced. Coloring as per *figures 1* and *2*. From **M. Minsky, P. Singh, A. Sloman,** *The St. Thomas common sense symposium: designing architectures for human-level intelligence.* The AI Magazine, Summer 2004.

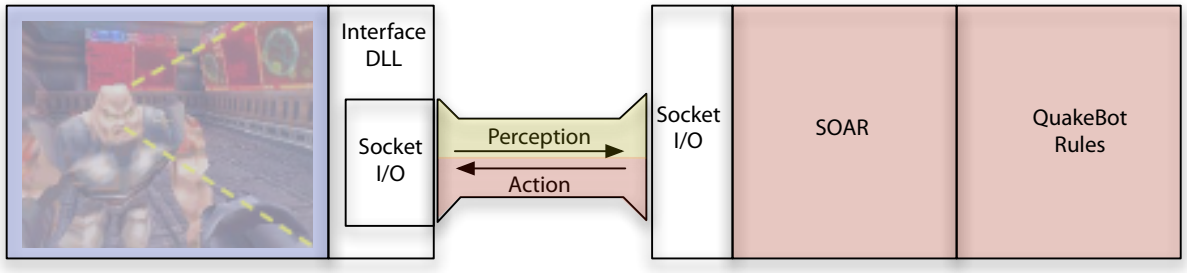


figure 9.
In this work the Soar general purpose AI architecture has been coupled to the computer game Quake. In this diagram, the "motor system" of the agents is almost entirely located inside the computer game itself. From **J. E. Laird,** *It knows what you're going to do: adding anticipation to a Quakebot*, International Conference on Autonomous Agents, Proceedings of the Fifth International Conference on Autonomous Agents, 2001.

This division between perception, action, and motor systems has been often hidden, suppressed or marginalized in the literature. For example, Brooks's early "subsumption architecture" work: where each subsumption layer might either inhibit or suppress elements of the layer below it, Brooks typically builds his own vertical architecture through sensor inputs, an augmented finite state machine "action selection" and motor outputs. The complexities of the sensing (which in the case of the early robotic vision work were significant) or ordering movement (which on wheel robots are insignificant) are left out of the diagram. However, as the bodies of robots (and later graphical characters) grow more complex and as the perceptual worlds of the robots also grow more complex, "sensor input" as a description necessarily yields to something that is worth labeling perception system, and "motor output" similarly yields to "motor system". By the time we arrive at humanoid robots this decomposition into perception, action and motor competencies is evident and fundamental.

This decomposition will serve as a useful starting point for our further complications and recastings of the agent metaphor throughout this thesis.

Authorship and AI

The agent concept itself offers a way of navigating AI literature. But there is another perspective on this body of work that is relevant to this thesis. This I shall call the authorship stance — the descriptive and critical stance toward an AI system that is based on what it like to make things inside it.

Descriptions of the creative process allowed or encouraged by practitioners' systems are surprising hard to find in the extant literature, and they are a hidden subtext to the papers, a secret currency between researchers. Classical, or non-agent-based, AI research often approaches this stance obliquely and narrowly; one can detect only hints of the engineering reality of the *practice of AI* under-

A famous example of a robot implemented within the subsumption architecture that collects soda cans is given in : **J. H. Connell,** *A colony architecture for an artificial creature*, MIT Ph.D. Thesis in Electrical Engineering and Computer Science, MIT AI Lab Tech Report 1151 (June 1989).

By the time we arrive at:

R. Brooks, *Elephants don't play chess*, Robotics and Autonomous Systems 6 (1990) 3-15,

the problems of fusing sensor signals prior to entry into the subsumption architecture is becoming apparent. The commencement of *expressive* robotics:

R. Brooks, C. Breazeal , R. Irie, C. C. Kemp, M. Marjanović, Brian Scassellati, Matthew M. Williamson, *Alternative Essences of Inteligence*, Proceedings of the American Association of Artificial Intelligence 1998, AAAI Press, CA.

states this decomposition directly, as does: **C. Breazeal,** *Sociable Machines: Expressive Social Exchange Between Robot and Human*. Artificial Intelligence Laboratory. Cambridge, MA, MIT. 2000.

neath a kind of academic politeness. Some of this trace manifests itself in the questions commonly asked of published systems — does the system scale? is it robust? These questions might be taken to be "can a person reasonably add to the system without it collapsing?", "is it possible to debug?", "how does it fail?". These questions are typically the hardest for scientific method to reach, but of significant relevance to this thesis — this is, after all, an argument for and an articulation of an agent-based *practice*.

Of course what is reflected in the history of the field and the level of critique and discussion in the field's papers, what one might call the *science of* AI, is only partially related to the practice of AI. It remains to be argued elsewhere whether this disconnect, indeed, this failure of academic AI discourse to integrate the use of AI into the discussion, shares any blame for the growing unease with the progress made by the field and the ambiguity of the relationship between a core academic AI and the more clearly engineering pursuits of either computer games or statistical machine learning. However, this peculiarity of the field is unavoidable if one approaches it to press its developments and techniques towards the service and synthesis of one's own art. For both broad-reaching frameworks and individual algorithms stand or fall in this foreign domain based not on their performance in the chosen micro-world or standard dataset, but on the story that emerges when they are turned loose within another micro-world — my micro-world — or on another dataset — the one that I'm faced with in a theater or a gallery.

Some of the motivation for the agent-based — and other distinct but related trends in the 80s and 90s such as connectionism and artificial life — came from an often open and explicit authorship twist: a belief that reactive, connective, adaptive or behavior-based systems avoid the burden of knowledge engineering (i.e. knowledge authorship) and exploit a far closer relationship with statistical

machine-learning techniques to avoid the hand-tuning, assembly or even creation of systems altogether.

On "repairing" the subsumption architecture to remove general purpose computation: **R. Brooks**, *How To Build Complete Creatures Rather Than Isolated Cognitive Simulators*, in: *Architectures for Intelligence*, K. VanLehn (ed), Erlbaum, Hillsdale, NJ, Fall 1989, pp. 225–239.

For example, we can use this position to reread Brooks's general appeal for simple natural analogs first, his structuring of layered behavior systems and his desire to limit the complexity of each layer to something much less than a general Turing machine, as authorship prescriptions — the attempt to create systems that survive in complex worlds without the unconstrained complexity that characterized previous approaches.

From **R. Brooks**, *Elephants don't play chess*, Robotics and Autonomous Systems 6 (1990) 3-15,

> In our experience debugging the subsumption programs used to control our physically grounded systems has not been a great source of frustration or difficulty. This is not due to any particularly helpful debugging tools or any natural superiority of the subsumption architecture. Rather, we believe it is true because the world is its own best model (as usual). When running a physically grounded system in the real world, one can see at a glance how it is interacting. It is right before your eyes. There are no layers of abstraction to obfuscate the dynamics of the interactions between the system and the world. This is an elegant aspect of physically grounded systems.

Elsewhere the explosion of energy surrounding the related fields of neural networks, genetic programming and artificial life in general in the 1980s and 90s was fueled by the promise that these techniques seemed to have to dodge the whole question of system authorship.

P. Maes, *Modeling Adaptive Autonomous Agents*, Artificial Life, 1 (1), 1994. pp. 135-62.

However, Pattie Maes follows Brooks's lead and articulates the basis for considering the software agent away from the world of robotics, virtually embodied in our computers as user interface or online as acting on our behalf. In this work she explicitly judges action-selection strategies based not on their mathematical qualities, experimental results or, in our terms, the field of potential developed, but based on how easy or hard it is to make agents out of them.

Overview from: **P. Maes**, *Modeling Adaptive Autonomous Agents*, Artificial Life, 1 (1), 1994. pp. 135-62.

L.P. Kaelbling and **S. Rosenschein**, *Action and Planning in Embedded Agents*, In: *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, edited by P. Maes, MIT Press/Bradford Books, 1990.

B. Blumberg, *Action-selection in hamsterdam: lessons from ethology*. Proceedings of the Third International Conference on Simulation of Adaptive Behavior, Brighton UK, 1994.

Maes plots a line through the "hand-assembled, flat structures" of early Brooks, the early work of Leslie Kaelbling and Stanley Rosenschein on agents with explicit goals, and Maes's own "compiled" flat behavior networks towards the hand-assembled hierarchical structures of Bruce Blumberg. Again, by the time we reach Blumberg the demands of authorship, and the complexities of the micro-worlds in which the agents are put to work, are necessitating new, more authorable, action-selection mechanisms as well as a clearer statement of the perception / action / motor system decomposition of the (in this case) graphical embodied agent. This is less of a shift of emphasis than a clarification of what agent-based AI has really been trying to find since its conception — a way of making things.

Concern for how AI agents are authored takes one directly toward another field very related to the work presented here — graphically embodied interactive agents. This field has always had a little more concern for the techniques and difficulties of actually authoring characters, being closer to interaction design, digital entertainment, and computer game production.

Evidence of this courtship includes a spate of "practical" books, including the AI Game Programming Wisdom series:

S. Rabin, *AI Game Programming Wisdom 1-2*, Charles River Media, Cambridge MA, 2002-3.

Even more conservative, the AI sections of the game programming gems series: **M. Deloura**, *Game Progamming Gems 1-3*, Charles River Media, Cambridge, MA, 2000-2.

and, **M. Buckland**, *Programming Game AI by Example*, Wordware Publishing , 2004.

Here of course, there are a number of examples of successful computer games that incorporate artificially intelligent characters and mainstream academic AI. The computer game industry is in the middle of an ongoing courtship limited on the one side, I believe, by the willingness to articulate a useful authorship position by academic AI and on the other by an willingness to create new game genres that truly require and exploit artificial intelligence.

Of the most notable recent successes *The Sims* for example succeeded in dominating if not forging a genre based on manipulatable synthetic people — dodging many of the more complex issues of making these characters smart by successfully basing their smartness on a vast array of objects and events with which the characters can interact. Earlier the successful *Creatures* got quite far with a very interesting agent framework — I believe it is telling that one of the most popular extensions to that series was the "creature science kit" that enabled players to directly manipulate (author?) aspects of the creatures. It is perhaps also telling that there are striking parallels between the online communities that went up surrounding *Creatures'* success and the craze for the altogether unintelligent Tamagotchi that came much later. Perhaps the success of these AI-based games has less to do with crafting a genuine AI-based genre that it initially appears. The recently interesting game *Black & White* — based on a central, learning agent — was certainly criticized by some as failing to find a stable genre — opting to combine instead world-building, role-playing and straight out fighting elements. The promised AI revolution of computer gaming has yet to take hold.

However it is possible, in the field of graphical characters, both to remain inside academia and yet stray too far from our AI roots. Ken Perlin and Athomas Goldberg's classic Improv architecture appears at first to solve *the whole problem* — that of authoring interactive graphical characters or actors with personality — through the creation of simple, hierarchical scripts. Indeed, in reading these papers one might be forgiven for thinking that the problem — creating live interactive creatures with shallow but broad artificial intelligence — never existed in the first place, but rather was a cruel hoax perpetrated by AI researchers (and programming language researchers) on the animation, game design and artistic community at large.

Maxis. *The Sims*, published by Electronic Arts. 2000-.

Millennium Interactive Ltd. *Creatures*, 1996.
— however, see **S. Grand, D. Cliff, A. Malhotra,** *Creatures: artificial life autonomous software agents for home entertainment*. Proceedings of the first international conference on Autonomous agents, ACM, 1997.

Bandai, *Tamagotchi*, 1996-7.

Lionhead Studios, *Black & White*, published by Electronic Arts. 2002.

K. Perlin and **A. Goldberg,** *Improv: a system for scripting interactive actors in virtual worlds*. In: SIGGRAPH 1996 International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM, 1996.

More detail concerning the motor level issues confronted in this work is given in: **K. Perlin,** *Real Time Responsive Animation with Personality*, IEEE Transactions on Visualization and Computer Graphics, 1 (1), 1995.

A closer look at these papers hints at the underlying problem that is not solved. That Perlin and Goldberg's architecture diagram omits any role for perception is an important clue. It is true that the interaction between the "behavior system" (the nest of scripts) and the "motor system" is a strong, indeed a seminal, contribution. That their motor system for character, with all of its kinematic, physical and content-level constraints survives when connected to its rather unpredictable action-system scripts is an important success. However, the interaction between this script-driven "behavior system" and the dynamic, unpredictable world is absent. Not a single one of the example action scripts in Goldberg's Lecture Notes in Artificial Intelligence paper talks about an external perception, influence or input, let alone one that can change by itself at an inopportune moment in a script's ballistic unfolding. This architecture, as applied to the problem of action selection in even simple, unscripted worlds, offers a outsider at least, little in the way of authorship strategies or tactics. As this line of work emphasizes the problems faced by the meeting of animation and action, these papers offer a lasting contribution. However, the "AI authorship problem" remains.

A. Goldberg, *Improv: A System for Real-Time Animation of Behavior-Based Interactive Synthetic Actors.* In Lecture Notes in Artificial Intelligence, Vol 1195, R. Trappl P. Petta (Eds.), 1997.

Similar criticisms can be levied against other animation-based work that appears to take on the problem of creating complete characters — for example: **J. Cassell, T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjálmsson, H. Yan,** *Embodiment in Conversational Interfaces: Rea.* Proceedings of the CHI'99 Conference. 1999.

J. Cassell, H. H Vilhjálmsson, T. Bickmore, *BEAT: The Behavior Expression Animation Toolkit.* In: SIGGRAPH 2001, International Conference on Computer Graphics and Interactive Techniques. Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, ACM, 2001.

Emergence, artificial life and digital art

It might come as a surprise, then, that it is at the very point when artificial intelligence's anti-authorial positioning is at its greatest that the fusion of art and AI reach their apogee, as art approaches the excitement surrounding *artificial life* and *emergent systems*.

C. G. Langton (ed.), *Artificial Life*, Addison-Wesley, CA, 1989.

Artificial Life's point of origin is usually given as a workshop organized by Chris Langton around the study of living systems without biological structures — a field concerned with "life-as-it-could-be", the formal basis for life, rather than "life-as-it-is", the material basis of life. Unlike artificial intelligence the focus is very much on evolution, morphogenesis, and metabolism and in particular, on emergent structures.

Emergence is characterized by systems of prodigious yet ultimately rather uncontrollable and unengineerable production. The academic fields that momentarily looked set to fuse to create a stable artificial life alloy have apparently moved apart and onward, yet contemporary artificial intelligence, indeed, any interdisciplinary academic field that occurs after "A-Life", has failed to get comparable traction in the field of digital art — either in art theory or in art practice. We must revisit this power that emergent systems had over digital art if we are to construct a new analysis of AI's potential to provide tools for digital artists.

On defining emergence: **E. M. A. Ronald, M. Sipper, M. S. Capcarrère,** *Design, Observation, Surprise! A Test of Emergence. Artificial Life 5: 225–239 (1999).*

ALife reconsiders its progress and its lack of progress at the turn of the millennium:

M. Bedau, *Artificial Life VII: Looking Backward, Looking Forward. Artificial Life 6: 261–264 (2000)*

Emergence itself is a difficult term to define, although a number of technical definitions exist. A most useful definition — seriously proposed at a time when Artificial Life was reconsidering its history — ties "genuine emergence" to a "lasting surprise" at a whole given an "apparently adequate description of its parts". This definition neatly captures the relationship between the captivating "coolness" of emergent phenomena, explaining some of the prevalence of "emergent art" over the last two decades, and their utter un-engineerability (or unauthorability). If an emergent phenomena stands or falls on the excitement of getting more out than one puts in, it conversely offers little advice on how one should go about getting anything in particular out of such a system. Artificial Life's emergence then stands more as an anti-methodology than a constructive practice and we should be as suspicious of "emergence" as we are of "mapping" as a point of origin for an art-making.

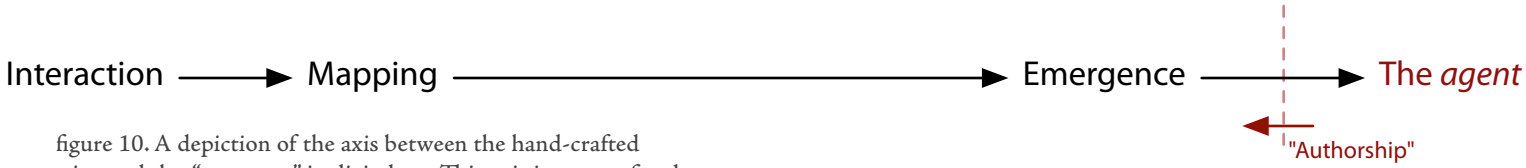


figure 10. A depiction of the axis between the hand-crafted mapping and the "emergent" in digital art. This axis is not confined to artworks with an explicit Artificial Life referent, we can detect a more general trend in interactive art; a belief that by the construction of complex systems, artists will gain access to new and interesting meaning-bearing forms. But how can one author an emergent system when the definition of emergence is the very surprise surrounding its unforeseen appearance?

However, there is a long history of the biological entering digital art through an artificial life context and we should pause to contextualize this thesis with respect to this work. Indeed, Artificial-Life-based art seems an ideal context to locate work, such as this thesis, that seeks to handle the creation of complex, biologically inspired art.

There are a number of reviews of the still growing body of ALife/Art works. One review is **K. Rinaldo**, *Technology Recapitulates Phylogeny: Artificial Life Art*, Leonardo 31, No. 5,371–376 (1998). A more comprehensive compendium is **M. Whitelaw**, *Metacreation: Art and Artificial Life*, MIT Press, Cambridge, MA, 2004.

A retrospective review of the artists' work is: **C. Sommerer** and **L. Mignonneau**, *Art as a Living System: Interactive Computer Artworks*. Leonardo, Vol. 32, No. 3, pp. 165–173, 1999.

More recent work is included in:

L. Mignonneau and **C. Sommerer**, *Creating Artificial Life for Interactive Art and Entertainment*. Leonardo, Vol. 34, No. 4, pp. 303–307, 2001.

In particular, the forms evolved in: **K. Sims**, *Galapagos*. Installation.

There is much precedent: for example, the highly influential work of Christa Sommerer and Lauraunt Mignaux spans a number of installations that are structured by allowing gallery-goers to meddle in the evolution of simulated creatures — staying close to a natural analogue in both appearance and process — in *Interactive Plant Growing* 1993, *A-volve* 1994, *GEMMA* 1996. Yet at the same time these works contain profound, convenient and presumably deliberate misreadings of genetics, eliding phenotype and genotype, morphology and embryology. After all, in reality, manipulating individual base-pairs just wouldn't make much *sense*. It is not that a more faithful instantiated biological system would make for better art, but rather it's important to note that this line of biology avoids the trickier problems of agent-based artificiality — phylogeny, adaptation, behavior. From this perspective the equally influential work of Karl Sims is both less ambitious and more complete, in that it tries more simply to evolve creatures without genomic authorship on the part of the viewer and yet

achieves graphics with astonishing apparent intentionality, strategy and even character.

Indeed, since Artificial Life is characterized by anonymity and group dynamics rather than personality and behavior, Artificial-Life-based art stands or falls on its ability to guide the interaction away from individual effort, intention and adaptation of the lives it purports to synthesize. This sleight of hand has resulted in a disconnect between theorizing and rhetoric surrounding interactive, Artificial-Life-based artwork. In fact, this literature reads like an argument for *artificial-intelligence*-based artworks. Strictly in terms of artificial life as a powerful meaning-bearing principle, gallery-goers are undeniably more familiar with a dog than with a fungus, a genome or a population distribution. No matter how artificial life bends or reprojects its biological inspirations and aspirations, it will miss these relationships and readings.

M. Bedau, *Artificial Life VII: Looking Backward, Looking Forward. Artificial Life 6: 261–264 (2000)*

Stepping back, we might ask how artificial life itself is doing as a scientific field entering its middle age. According to recent reviews, one fundamental question that is posing considerable difficulty concerns creating example simulations that show multi-scale, and multi-level emergent properties. We know from nature that extremely deep chains of self-organizing and self-regulative structures are one of the hallmarks of biological systems.

This far on in the history of A-Life based art, the lack of plentiful systems that produce higher-order emergent structures at this point reads more as an obituary than a call-to-arms for the intersection of artificial life and digital art. As artists we are typically interested in structures that have more than one level. Indeed, we typically assume that any rich starting place has more than one level. If I cannot "emergently" get to someplace where one long time-scale governs a shorter time-scale, where one space overlaps another, where a complex of small

objects coalesce into a complex large object, then it is time to reconsider the excitement around the emergent.

As I have argued in my critique of mapping, the central problem of digital art is not *generating* potential, it is working with it and within it— it is navigating it; it is drawing an atlas with your collaborators and agreeing on the names of the continents; it is remembering where you have been in the space; it is turning this potential field into a work. And if the problem isn't generating potential, there is no need to be excited should it turn up or rather *emerge* without much effort on our part. Such easy possibility is not an omen of good art but an harbinger of effort to come.

Other intersections between art and artificial intelligence often slice AI too thinly — to glean potential without organization — building systems with specific deep but narrow competences. This is a classic criticism levied against main-stream artificial intelligence by both Brooks and Minsky at various points. Of the recent compendiums of articles on the use of artificial intelligence in music, the interdisciplinary corner of "art-making and AI" that has seen the most activity, two aspects stand out: firstly, almost all of the introductory descriptions of artificial intelligence fail to include the concept of an agent in their ubiquitous opening survey (opting instead to find a neat binary opposition between symbolic AI and, say, connectionism).

For Brooks, see references above, for Minsky, this is his central attack on mainstream AI in his upcoming *Emotion Machine*.

For example, many instances of this blindness to the agent are to be found in: E. R. Miranda (ed.), *Readings in Music and Artificial Intelligence.* Routledge, 1999.

Music and connectionism — collected in: **P.M. Todd** and **D.G. Loy** (Eds.) *Music and Connectionism*. Cambridge, MA, MIT Press 1991.
However, the field continues, music and (recurrent) neural networks: **D. Eck** and **J. Schmidhuber,** *Finding Temporal Structure in Music: Blues Improvisation with LSTM Recurrent Networks.* **H. Boulard**, editor, *Neural Networks for Signal Processing XII*, Proceedings of the 2002 IEEE Workshop. 747{756, New York, IEEE, 2002.

Music and generative grammars — **E. R. Miranda,** *Regarding Music, Machines, Intelligence and the Brain: An Introduction to Music and AI*. In **E.R. Miranda** (ed.) Readings in music and artificial intelligence, Hardwood Academic Publishers, 2000.
And, of course, the much more sustained analytic work of **F. Lerdhal** and **R. Jackendoff**, *A Generative Theory of Tonal Music*, Cambridge, MA, MIT Press, 1983.

Music and genetic algorithms — one thread of research is concluded in: **P. M. Todd, G.M. Werner,** *Frankensteinian methods for evolutionary music composition*. In: **N. Griffith** and **P.M. Todd** (eds.), *Musical networks: Parallel distributed perception and performance* Cambridge, MA: MIT Press/Bradford Books 1999.

Music and Markov models: **L. Hiller** and **L. Isaacson**, *Musical Composition with a High-Speed Digital Computer*. Journal of the Audio Engineering Society. 1958. **M. Farbood** and **B. Schoner.** *Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains"* Proceedings of International Computer Music Conference. Havana, Cuba. 2001.

Finally, Minsky's classic manifesto for why the border between music and AI should be much longer and more intricate: **M. Minsky**, *Music, Mind, and Meaning*, Computer Music Journal, Vol. 5, Number 3. 1981.

Robert Rowe's earlier work, *Cypher*, **R. Rowe**, *Interactive Music Systems: Machine Listening and Composing*, MIT Press, Cambridge MA, 1992.

David Cope's well known "Experiments in Musical Intelligence" project(s), surveyed in: **D. Cope,** *Virtual Music, Computer Synthesis of Musical Style*, MIT Press, Cambridge MA. 2001.

George Lewis's Voyager systems: **G. Lewis**, *Interacting with Latter-Day Musical Automata.* Contemporary Music Review 18/3 (1995): 99–112.

Secondly, what practitioners mean when they write "music and artificial intelligence" is almost always "music and something that some AI has found useful". Thus, we have admittedly fascinating work in music and the neural network, music and genetic programming, music and Markov models (hidden or not), music and self-organizing maps. Research that is a called "music and AI" is generally missing a strong "music and complex AI systems" vein. On the one hand, this is unsurprising: artificial intelligence has always been encroached on by machine learning, artificial life, and exploratory statistics. On the other hand this is rather unexpected, given general interactive tendencies in this research (both as tool-for-composer and instrument/partner for performance) and the widespread acknowledgment of music as an art that, if it has a definition at all, is defined by the broad range of faculties that it draws upon and synthesizes.

However, the standout exceptions, I believe, represent some of the best and lasting work in the field of computer music. Robert Rowe, George Lewis and David Cope have all built systems that have reached the openness, the mass and heterogeneity that are hallmarks of actual "artificial intelligence" systems. The former two have exploited in different ways the software agent metaphor and framework; all three have at least addressed if not looked closely at the languages , tool-sets, and representations needed for their work from an authorship perspective. We shall see shades of both Rowe and Lewis in the areas of this work that are most resolutely computer-music directed, *Loops Score* and parts of *The Music Creatures*.

In the visual arts we see a similar pattern, and when an agent-based framework is claimed we should eye it as closely as we inspect artificial-life-based art's biological inspiration. This is not to critique the success of the art itself, but to inspect the strength and practical use of the metaphor. Flavia Sparacino constructs an "intentional agent"-based interactive graphical and music dance space but the metaphor seems thin — the agent is missing a body of any complexity to control and the work is much closer to the principles of mapping perceived movement than the description immediately reveals. Claudio Pinandez sustains a longer agent narrative within the field of interactive drama, using real actors and live graphics, and his agents represent an important early perceptual contribution to understanding the drama realm live. Some of the fruits of Carnegie-Mellon's "Oz" project are also of relevance, including the "Façade" project and more recent work, which has tried to broaden the possible cultural application of AI considerably.

However, as the interactionist or agent-based reaches the fields of drama and narrative, the possibility of an interaction between cultural production and artificial intelligence, as constructed by actual AI / art practitioners, is now being seriously written about. Most related to this work is Michael Mateas's contributions to locating the classical AI versus interactionist AI debate with respect to the working artist's cultural production. Out of the ashes of this rhetorical bonfire he fuses a third way, an alternative "expressionist AI".

In his writings Mateas explicitly warns artists against aligning themselves too strongly with "interactionist techniques" which might result in them "missing out on a rich field of alternative strategies for situating AI within culture". This rejection is deemed necessary to license the *Oz* project's use of "traditional AI" structures for its natural language- and narrative-based works.

Flavia Sparacino reviews her work with an agent-based bent in: **F. Sparacino, G. Davenport, A. Pentland,** *Media in performance: Interactive spaces for dance, theater, circus, and museum exhibits*. IBM Systems Journal, 39 (3&4), 2000.

Claudio Pinandez's work in "computer theater": C. S. Pinhanez, *Computer theater*. Technical Report 378, M.I.T. Media Laboratory Perceptual Computing Section, May 1996.

The "Oz Project" is reviewed at its beginning: **J. Bates,** *The Nature of Character in Interactive Worlds and The Oz Project*, Technical Report CMU-CS-92-200, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. October 1992.

And near its end: **M. Mateas,** *An Oz-Centric Review of Interactive Drama and Believable Agents*. Technical Report CMU-CS-97-156, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. June 1997.

Its "successor" the "Façade project": **M. Mateas** and **A. Stern,** *Architecture, Authorial Idioms and Early Observations of the Interactive Drama Facade*. Technical Report CMU-CS-02-198, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. December 2002.

M. Mateas, *Interactive Drama, Art, and Artificial Intelligence. Ph.D. Thesis*. Carnegie-Mellon University, December, 2002.

The root of this criticism appears to be a misunderstanding about the role of the agent metaphor in contemporary AI practice and the relationship of the agent as used today with the agent as used in early Brooks. The rejection of the *prescriptive* power of the agent-as-metaphor arises from a disagreement with an imagined *proscriptive* thrust of the early rhetoric that surrounded its birth. It is as if the organization of the debate perpetuated in brief historical capsules (and thesis context sections such as this) froze in the early 90s. The agent functions today as an organizing principle, and as such organizes extremely hybrid structures that press into service representations and algorithms that in the 90s might have been perceived as heretically classical, and can do so without becoming "vacuous". The agent-as-metaphor is not a position that rejects materials, but it does structure them strongly. What one "misses out on" with a rejection of the agent as a poetic metaphor is a set of ideas about how one might go about structuring a complex system that interacts with a dynamic, unpredictable world of which it itself is a part. While stance neutrality on any particular AI debate might initially seem an appealing non-position for an artist (one who wishes to remain external to the field of AI), to my mind this position simply recapitulates the narrative of endless potential and new possibilities that Mateas justifiably finds suspect in early interactionist AI.

Rather, I argue it is more productive for the AI / artist to choose some broad and useful organizing principles over a practice of ensuring the perpetual blankness of their slate. I further disagree with Mateas's insistence on the novelty of the artist's focus on authorship issues within AI or even the transformative possibilities of artists engaging in an AI practice. AI, and in particular agent-based AI, has always has a profoundly important engineering tendency which has maintained an interest in authorship problems. To set up an opposition — that art focuses on the negotiation of meaning as mediated by the object, while AI focuses on the internal structure and its interaction — that is, to make a clean separation between art's art and AI's science, one has to first strip

AI of its engineering core. As much as I appreciate the difference of the value structures apparent in both literatures, this opposition is not is not clear-cut in practice. That this core has been poorly expressed in AI's relationship within the workings of scientific culture, publication and discourse is, I feel, undeniable. But absence of evidence here is not evidence of absence. The artists who exploit the AI tradition and literature are certainly not the first to make things using AI techniques.

However, what is true is that this line of work takes the agent-based directly toward the parts of human activity that AI has found so hard to reach — the use of language and complex human narrative. In contrast, the work presented in this thesis, which deals exclusively with non-linguistic, non-narrative domains, is somewhat dislocated from the few examples of uses of AI in theater and drama. Behind this dislocation is a significant difference of approach.

The story of Classical AI versus Nouvelle AI has often been presented as story of the narrow-and-deep (the classical chess-playing computer) versus the small-but-complete (Brooks's intelligent insect). The *Oz* project, *Façade* and the work, for example, of Justine Cassell and Norman Badler and the general tradition of language- and gesture-based intelligent agents all stretch the small-but-complete of the agent to a broad-and-shallow — an "aspect ratio" unenvisaged and unintended by Brooks; a breadth that goes all the way out to human language.

A review of much of this work can be found in: **J. Gratch, J. Rickel, E. Andre, N. Badler, J. Cassell** and **E. Petajan.** *Creating interactive virtual humans: Some assembly required*. IEEE Intelligent Systems, 2002.

And a point of origin: **N. Badler, C. W. Phillips** and **B. L. Webber,** *Simulating humans: computer graphics animation and control*, Oxford University Press, 1993.

From the broad-and-shallow-agent researchers behind the Oz Project:

"It has been suggested to us that it may be impossible to build broad, shallow agents. Perhaps breadth can only arise when each component is itself modeled sufficiently deeply. In contrast to the case with broad, deep agents (such as people) we have no *a priori* proof of the existence of broad, shallow agents."

from: **J. Bates, A. B. Loyall, W. Scott Reilly**, *An Architecture for Action, Emotion and Social Behavior*. Technical Report CMU-CS-92-144, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. May 1992.

There is little *a priori* evidence to suggest that convincing broad and shallow agents are possible. And despite the constructive work that has occurred in this area was made I find the work in general has failed to coalesce a generally useful set of core techniques and ideas for practicing different classes of "breadths". This is, of course, rather unsurprising — if there were a small set of reusable ideas that enabled a host of ultra-broad-yet-shallow agents over a comprehensive set of human-level domains then there would be no need for any "depth" and AI would have simply imagined its core troubles.

The work presented here follows a more conservative and consolidating path. When an agent is described as "broad" we mean as broad as the world that we put the agent to use in (and no broader, and no narrower). And when it is said to be shallow, it means supple, and not over-committed to a particular problem domain ahead of time. The frameworks that back the agents developed in this thesis have all found use and instantiation in multiple agents, often designed by multiple people. That these agents do not extend their apparent breadth all the way out to human-level competencies is part of the cost at this time of developing more fundamental and reusable frameworks: *Dobie* looks to the training of real dogs in order to work through, to an unprecedented level, the details and needs of trainable computer systems; *alphaWolf* looks to wolf behavior for its core interaction.

Toward an aesthetics and a practice of the agent-based

Signs of this tactical compromise are to be found directly in the artworks presented in this thesis. *The Music Creatures* offers up multiple fragments of human-musical competency rather than an attempt at a totalizing human whole, looking instead to the proto-musical competencies of animals; *how long...* similarly offers a sequence of overlapping agents that slice through the choreography in different ways and different times, while each attempt by the agents is

left radically incomplete. *Loops Score* takes a narration as its score but plays with aspects that lie halfway between language and meaning, sound and music.

The perpetual inadequacy of these agents can be compared to the nonexistent "human-level" artificial intelligences that they refuse to fake — an "automatic music-generating system" (in the case of *The Music Creatures*); a "live choreography-notating system" (for *how long...*) or a "meaning-to-music converter" (for *Loops Score*).

This rejection of the directly human results in a series of works that are continually trying to catch up with their material, constantly off-balance, perturbed by — rather than at equilibrium with — the gallery or stage that they share. The resulting artworks develop an agent-based aesthetics of intention, effort and transience. Every other aspect of these works — from the gestural, hand-drawn qualities of their imagery to their attitude to human motion as it evaporates in front of their gaze — gathers around this unstable core.

This privileging of disequilibrium in my gallery works may seem no more than a matter of personal style, a mere reaction against the well-trodden paths of computer graphics and interactive art. However, I believe it also to be nothing less than the condition of contemporary dance — and a symptom of the computational sensibility in modern choreography in particular. Is not this disequilibrium implied by the gap between the audience's inevitable search for mimetic representation and the transformative calculus hidden in this choreography? Is it not this transience that Forsythe points to when he calls dance the poetry or architecture "of disappearance"?

c.f. **R. Sulcas**, *William Forsythe: The poetry of disappearance and the great tradition*. previously available online:

http://frankfurt-ballett.de/articles2.html

however, since the dissolution of the Ballett Frankfurt, this paper is currently available through the internet archive:

http://web.archive.org

It is like parallel computing. In the old days, to make a dance in which nothing changes, people used their perception to make a rigid structure, we use our perception now to make a very complex structure. If I have 8 people figure out a dance from the inside, I have 8 people looking at 8 different things, from which they make connections. So basically what you are seeing in the third act of *Eidos* is a huge connection machine, using the human being as the original machine. It is primitive and wonderful, like a game; always the same game, but each time played differently.

William Forsythe, quoted in **T. Ozaki, (P. Vigilio trans.)**, *An Interview with William Forsythe*. (availability as above).

Further, the agent metaphor, and the technologies to which it will lead us, offer fertile ground for the growing of small algorithmic ideas, hypothetical enabling constraints, and game-like forms that are open to the possibilities of chance and interaction. But more importantly, unlike the cold computation that computers find so effortlessly, but like the "parallel computers" of Forsythe's dancers, artificial intelligence's technologies of learning and adaptation, and its structuring of the problems of perception and movement, allow artists to work through the consequences of these tactical formalisms.

4. "Non-photorealism" and computer graphics

One, inspirational, *post hoc* synthesis of photorealistic computer graphics is Andrew Glassner's formulation of the rendering equation in **A. Glassner,** *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers, Inc, San Francisco, 1995.

The goals and metrics of photorealistic computer graphics are relatively well understood. This synthesis of the primary algorithms, of forward and inverse ray-tracing, radiosity and surface-modeling techniques came at a time when practice had not irretrievably outstripped theory. Drawing deeply on our understanding of the physics of the world it is possible to interpret the wide variety of photorealistic techniques in a single unifying framework which can in turn be used to derive, or at the very least locate, approximations that are inside the current real-time envelope accessible to contemporary graphics hardware.

Any theory of non-photorealistic computer graphics is in immeasurably worse shape. Having benefited from, or at the very least been able to co-opt, the hardware created for these immersive, realistic graphics, the field of non-photorealistic works seems too large to unify, too diverse to theorize over and its boundaries too poorly defined for us to artificially construct a set of natural kinds to examine and reason with.

One might, however, start finding some orienting landmarks, if not in the aesthetics of the resulting works, in the techniques or technical styles deployed therein. It is a field, after all, wedged between the predominantly photorealistic

theoretical legacy of military virtual reality and Hollywood special effects and the, again, predominantly photorealistic technical affordances offered by the commodity hardware necessitated by computer games.

While we might feel that all kinds of non-photorealistic life has taken root in this space, I believe that we'll find a limited number of survival strategies at work. A comprehensive survey of all of non-photorealism in computer graphics is a demanding task — it is after all a field that is defined principally in terms of what it is *not* — and I shall limit the discussion here to trying to organize one corner of this space that works in real-time, that is concerned with live and interactive settings and is not devoted solely to duplicating early twentieth-century painting or nineteenth-century engraving. We will try to find these landmarks or axes throughout this document to locate technical styles, to examine the tensions navigated by certain technical approaches or to look at possible groupings of the aesthetic results of these practices.

One pole I shall refer to as "textural"; it has a corresponding anti-pole "geometric". These terms come from a purely technological distinction: these places embody the parallel paths of real-time computer-graphics architectures that always must incorporate manipulations at the level of the individual pixels and at the level of points, lines and planar elements. This is the perennial separation between bitmap image-manipulation software (e.g. Adobe Photoshop) and the vector-based (e.g. Adobe Illustrator); this is the difference between "fill-rate" and "vertex operations" that graphics-accelerating hardware performs; between the "fragment shader" and the "vertex program" of the modern, reinvented core OpenGL, or the "imaging" and "primitive" pipelines of OpenGL's previous core. Similar cleavages appear elsewhere: the sample level (textural) of the Music N / CSound family of musical programming languages is encoded in a different language from the score (geometry); the blocks of musical signal (textural) are distinct from the world of lists and bangs (geometric) of Max/MSP.

An excellent online resource covering this area is to be found at: **C. Reynolds**, http://www.red3d.com/cwr/npr/

In print, a review: **T. Strothotte, S. Schlechtweg**, *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Morgan Kauffman, 2002.

Despite more than a decade of dominance, development and convergence **Adobe corporation's**, Photoshop and Illustrator (www.adobe.com) remain separate, distinct and normative on their application domains.

A review of "modern" versus "classical" OpenGL (www.opengl.org): **R. J. Rost**, *OpenGL Shading Language*, Addison-Wesley, 2002.

The "score / orchestra" fissure is discussed in **R. Boulanger** (ed.) *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing and Programming*, MIT Press, 2000.

Max/MSP is available at: http://cycling74.com

At present I believe we are a point where the textural is on the ascendancy in computer graphics in general and in interactive digital art in particular. The real-time realism of recent computer games owes more to the textural complexity afforded of normal-mapping than it does to the geometric complexity of the rather low-polygon models that carry those static textures. And of course, interactive graphics is dominated today by the processing of video (texture). The rise of "image-based" rendering techniques that work either mainly or solely with 2D images exploiting the convenience of video sampling compounds this trend.

This is not due to technical opportunity; in the case of interactive art the decompression and manipulation of multiple video streams still taxes modern hardware and the gap between the resolution of interactive, textural, video-based work and its audience's high definition televisions shows no signs of narrowing quickly. Rather, a full diagnosis of the fascination with video in particular and the textural in general requires a little more disentangling.

In any case, the best, and most interesting work in the field of non-photorealism has been some work to bridge this texture / geometry segmentation, marrying the controllability of geometry with the fluidity of texture. Often, however, this work has focused on developing one particular style, taking photo-realism's geometry and texturing it appropriately, rather than finding a broader, experimental framework. I know of no work to date that provides a generic framework or principles for synthesis of an animated form that lies between the purely abstract and abstractions from sampled material. The purpose of the *re-projection rendering* techniques developed in this thesis is to provide an instance of such a framework.

Image-based rendering, early work includes: **P. Debevec, C. J. Taylor, J. Malik,** *Modeling and Rendering Architecture from Photographs:A hybrid geometry- and image-based approach*. In: SIGGRAPH 1996 International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. ACM, 1996

An overview: **C. Zhang** and **T. Chen** *A survey on image-based rendering-representation, sampling and compression*. In :*Signal Processing: Image Communication*, January 2004, 19, (1), pp. 1-28

For the technique of normal mapping, an introduction can be found in any "game graphics" textbook. For a more detailed practical discussion: **D. H. Eberly,** *3D Game Engine Architecture : Engineering Real-Time Applications with Wild Magic,* Morgan Kaufmann, 2004.

To cite just two influential examples of geometrically controllable texture: **S. Strassmann.** *Hairy Brushes*. In SIGGRAPH 1986 International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques. ACM, 1986.

and:

L. Markosian, M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, and **J. F. Hughes.** *Real-Time Nonphotorealistic Rendering*, In: SIGGRAPH 1997 International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 24rd Annual Conference on Computer Graphics and Interactive Techniques. ACM, 1997.

The textural / geometry pole is of use in analyzing the contemporary use of live graphics in a dance theater setting. Increasingly today the ascendancy of textural computer systems is echoed by those working in dance technology who are turning to video technology to "sense the stage". Where does this come from? and where is it going?

See, *Improvisation Technologies* (CDROM), ZKM (Center for Art and Media Technology) / Ballett Frankfurt, 1993.

In the early 1990s, Forsythe published many of his choreographic techniques in a seminal pedagogical work — the CDROM *Improvisation Technologies* (1993). Crude, but effective, hand-rotoscoped annotations of his inventions — generalized kinespheres, hidden representations, and obscuring constraints — overlaid Forsythe's own dancing and were accompanied by examples from complete choreographies made for the Ballett Frankfurt. Effective, this tool was still in use for training new members of the company up until its dissolution. This work could have acted as a "call-to-arms" for both choreographers (showing that a powerful articulation of their ideas was possible and useful using new media) and would-be digital dance specialists (showing that a relationship could be made with dance on a level deeper than the visual appearance of the dancer), but instead *Improvisation Technologies* indicates a path not taken by the dance technology community at large. There are a number of reasons why the field did not develop and unfold in this way. I shall focus on diagnosing part of the problem using only one symptom, which will also be useful in highlighting some of the differences between the work conducted for this thesis and the prevalent field.

Part of the impasse is due, I believe, to the interactive dance community's use of video as the computer's way of seeing the stage. Video has a number of apparent conveniences: it is cheaper than other sensing technologies, the interfaces with computers have benefited considerably from recent consumer demand, a video frame or sequences of frames seem to offer a large amount of data, and finally it is, of course, easy to visualize how computation acts upon video. There is even a sub-field of dance concerned with dance on camera, which offers a veil of precedent and theory.

Unsurprisingly, then, the number of dance performances using live and processed video has been growing for a decade; the tools to support these works are being standardized, distributed and sometimes even supported. None of these things are true of the technology used for many of the works described in this thesis — motion capture. Not only can't one buy in the store the tools used to build the pieces described in this thesis, but my principal pieces (*how long...* and *22*) are the first use of real-time motion capture in a major dance work. The underlying hardware technologies are expensive, specialized, obscure, rare and precious.

Video, despite its apparent mimetic transparency, is a poor *computational representation* of human *movement*. It is poor because it is not clear what transformations of pixel-level data have to do with transformations of human motion, much less a choreographic dialogue; poor because capturing it constrains the existing stage picture, the lighting and the set elements that have a longer tradition of relating to dance than video; poor because it is a fragile representation — slow it down, zoom in, or recast it and quickly video's own materiality comes to the fore, leaking onto the surface of the works created with it and pinning the level of dialogue between the collaborators to a negotiation of appearances.

Some recent examples of the ongoing trend towards video-based sensing and/or projection in dance include the work by the dance company **Troika Ranch** *Future of Memory*, 2003, by media artist **Klaus Obermaier**, *Apparition*, 2004 and *Vivisector*, 2002, **Wayne McGregor** / Random Dance's *AtaXia*, 2004.

In each of these works, the primary sensing technologies (for both the real-time works and the pre-prepared video materials) and the primary body representation used digitally is the video frame. The apparent (in)ability for video to allow computational "access" to human in these works often stands remarkably at odds with the stated intentions of the artists involved.

here is a little history of motion-capturing dance, much of it is reflected in the work of Paul Kaiser and Shelley Eshkar (collaborators on *Loops, Loops Score, how long...* and *22*). Information available online: http://www.openendedgroup.com

For a review of a broader range of work:

S. deLahunta *Dialogues on Motion Capture* Proceedings of IDAT 1999.

While it may be technically accurate to say that "video is used to sense the stage" during a typical interactive dance work, little transformative representation of the stage actually takes place: the stage is sensed by video but not perceived by video, and little representation of the stage survives the transportation through video into the computer. Rather, this video input leads directly to a predominantly textural computational methodology, an aesthetic of image-based abstraction turning around limited mimetic representations inside the computer.

The computational sensibility outlined in this argument suggests starting a little closer to what it is that choreographers care about — human motion. Yet human motion is hard for a computer to see in the sea of pixels that is video. Modern dance is perhaps the worst-case scenario for the already challenging pursuits of the field of computer vision; the sophisticated and experimental techniques that would be required to bring human dance motion out of the video frame are not yet stable enough for a sophisticated and experimental piece to be constructed upon them.

In that variable lighting, camera placement, multiple overlapping bodies, a variety of motion with a large dynamic range, and even costume conventions seem to be exactly configured to thwart much of the seated, upper-torso focus of practical computer vision research.

Motion capture offers such a representation while also having the "benefit" that, having never captured the appearance of a dancer through a camera there is no easy way to duplicate it — there is no path of least resistance leading to duplication and repetition. Motion capture is the basis for a hybrid representation: one that lies between the purely computational and the dedicatedly mimetic.

This has been shown in a series of studies in the 70s and 80s, starting with **G. Johansson**, *Visual perception of biological motion and a model for its analysis*. Perception and Psychophysics, 14: pp. 201-211. 1973.

Johansson's work, and the work of people who followed, showed that when presented with a few points of motion (effectively the "dots" of motion capture) humans are capable of correctly labeling body parts in the absence of all of the points, recognizing gesture, differentiating gender, recognizing familiar people, and even recognizing their own motion.

A more recent review of this matter may be found in **J. K. Hodgins, J. F. O'Brien, J. Tumblin**, *Judgments of Human Motion with Different Geometric Models*, Transactions on Visualization and Computer Graphics, 4 (4), December 1998

This double aspect of motion capture is clear: it is computational because computers can clearly manipulate it with considerable ease; yet a direct presentation of the data is shockingly readable. This, then, is representation that supports transformation and recasting — and it is the place that we will meet the computational sensibility of contemporary choreography.

This, then, is surely the place to begin generalizations from which abstractions can be made and on which algorithms can act. These actions, made visible, might yet be related more, in the eye of the audience, to human movement than they are to the technology that captured it and the representation that stored it. In the language developed above motion capture necessitates a geometric force and a point of departure from the abstract to re-project motion back onto the stage. This is the technical point of contact around which a dialogue between programmer and choreographer can occur, an intermediate point that is neither automating say, the dice rolling of Cunningham, nor simply duplicating the appearances of the performer.

Toward ambiguous computational graphics

Armed with the textural / geometric opposition it is possible to project artworks onto it. Karl Sim's genetic programming / artificial-life-based work (for example, the Galapagos installation) leans toward the "geometric", while his earlier, arguably less lasting work is textural.

For example, the 2D image-based work in **K. Sims**, *Artificial Evolution for Computer Graphics*. in Computer Graphics, 25(4), July 1991, or the *Genetic Images* installation, with **K.Sims** *Evolving 3D Morphology and Behavior by Competition*. In: *Artificial Life IV Proceedings*, R. Brooks & P. Maes (eds.) , MIT Press, 1994.

G. Levin and **Z. Liberman,** *The manual input sessions*. 2004. Performance.

K. Obermaier, *Apparition*, 2004. and *Vivisector*, 2001-2.

"I think we are in a very curious position today because, when there is no tradition at all, there are two extreme ends. There is direct reporting that is like something that is very near to a police report. And there is only the attempt to make great art. And what is called the in-between art really, in a time like ours, doesn't exist. [...] ... with these marvelous mechanical means of recording fact, what can you do than go to a very much more extreme thing where you are recording fact not as simple fact but on many levels, where you unlock the areas of feeling which lead to a deeper sense of the reality of the image, where you attempt to make the construction by which this thing will be caught ..." (p. 66).

Francis Bacon, in **D. Sylvester,** *Interviews with Francis Bacon*, Thames-Hudson, 1975.

Few works are nimble enough to play between these poles. Golan Levin and Zach Liberman's performance work for hands, projectors and cameras, "The Manual Input Sessions", draws its motivating humor from the dramatization of the crossing point between shapes made from hands (the abstract, made through a most resolutely geometric use of video) and hands made from shapes (the abstracted). In Klaus Obermaier's influential work for projections over dance theater *Apparition* or the earlier *Vivisector*, it is not the play of figure / ground that is of lasting interest; rather it is the threat of video unusually born of transformation — human body parts re-projected onto other body parts — rather than as bearer of mimetic intent. Of course, in these works (the latter of which uses video to capture the silhouette of the dancers) this is achieved, if it is achieved at all, through careful rehearsal rather than clever digital representation — but that they are succeeding, to my eye, exactly at the point where they are pushing against the affordances offered by the image-making technologies that they use is critical to the argument here.

In each of these performance-based works it is the vertiginous places during the performance where the imagery loses its definite location between textural and geometric that are most lasting to my eye. I argue that non-photorealistic graphics, if it is to move beyond technical demonstrations of mimicry, or mere novelty, will only find its contact with a lasting use in lasting digital art in these technically difficult, ambiguous, middle grounds.

The "real world" seems to be stacked in favor of the textural over the geometric. The "reality" of photo-realistic graphics remains heavily dependent on sampled or procedurally generated textures; the physicality of the drawn line is revealed as much in the surface qualities of the paper and the interaction between layers of graphite as in the geometric shape of the drawing; both the softness of the organic and the abrasiveness of its patina explode geometric complexity. Given these challenges, it is no wonder that the most recent increases in graphical

processing power and machine learning have given birth to a sub-field of "image-based" off-line computer graphics.

There is a textural bias to the aesthetics of all this processing power. The apparent "softness" of the organic form was at one point the most sought-after and precious commodity of computer graphics, photo-real or not. The "gaussian blur" was a popular benchmark of processing muscle; the smooth curve of the non-uniform rational b-spline a hallmark of sophisticated and expensive non-polygonal manipulation. But more often than not the texturality of computer graphics fumbles its physical, or even biological, referent in real-time transformation. For its coveted smoothness are the result not of an unspeakable multitude of past processes acting on the geometric, they are rather anti-patina: an erasure of history, a hiding of the lack of process by which the rendering came about. The gaussian blur *deletes* high frequency information; the smooth interpolate surfaces are skins stretched through a *hidden* 3-dimensional lattice.

The aesthetics of the work I wish to develop reveals process. Thus in computer graphics, photo-real or not I begin at an unconventional and inconvenient place. The works presented in this thesis stem from a common search: can we find a non-photoreal, geometrically controllable textural aesthetic? Can their texture be the trace of a process rather than an additional decorative layer? Can geometry interact on a textural surface? Can these interactions be metaphorically physical or biological but geometrically rather than texturally faked? It is from these principles that the work, by the time we are ready to enter dance theater, readies itself to accept motion-capture material and play with the mimetic and transformative possibilities offered.

Concluding remarks

In surveying some of the context of this thesis this chapter has sketched its main arguments. From the narrowest to the broadest they are:

- that with its recent focus on the "textural", live graphics in dance theater fails to allow either computer graphics or simply computer algorithms sufficient access to the very stuff of dance and choreography — human movement; that new rendering techniques and new sensing methodologies must be developed and incorporated into the field of dance technology for genuinely computational interactive artworks to be created; that, paradoxically, prominent choreographers today are outpacing the dance-technologist's deployment of computational ideas.

- that it is time for an articulation of the agent-based in digital art; that in particular, this should be seen as a platform from which to critique the prevalent synthetic and analytic approaches of interactive art in general and dance technology in particular. The agent-based offers a radically different path for creating and navigating the *potential* developed by computation processes in comparison with either interactive art's "mapping" or artificial life's "emergence", and an alternative point of origin for the development of digital art-making tools. This metaphor provokes and accepts choreography's use of "tactical formalisms" as a working practice.

- that the time is ripe for algorithmic art and contemporary choreographic practice to enter into a genuine, constructive dialogue; that for too long the interactive digital arts have largely ignored the precedent of and opportunity offered by dance.

These arguments will be fleshed out in the chapters that follow, and all of them will be responded to in the final works for dance theater that I present.

This chapter is necessarily more technical than the chapter that preceded; it articulates the specific technical context and technical origin of my work. It introduces the agent toolkit developed by the Synthetic Characters Group in 2001, and includes an overview of the c43/c5 action-selection approach, as an important example of the agent as an "organizing framework". And it ends with a critique of the largest, most complex installation from that period — alphaWolf — *indicating how the contributions of this thesis respond to the weaknesses that appeared during the development of this work.*

Chapter 2 — Beginnings

In 1999-2003 the Synthetic Characters Group, of which I was a part, proposed a new line of action-selection algorithms and an action representation that is used in, or at least relevant to, many of the works described in this thesis. The agent toolkits that embedded these techniques were referred to as the 'c' series, c2 all the way through to c43 and c5, and these are as good a name as any to refer to the successive developments of the approach. This work perhaps reached its apogee with *Dobie*, an interactive synthetic dog character, trainable in many of the ways that real dogs are. Inside *Dobie* there is an elaborated version of the action selection structure used in the projects *alphaWolf*, *Loops* and an early version of *The Music Creatures*, and this is the action-selection approach that motivates the later discussion of the Diagram system.

The 'c' series of work is described in the following two papers:

R. Burke, D. Isla, M. Downie, Y. Ivanov, and **B. Blumberg,** *CreatureSmarts: The Art and Architecture of a Virtual Brain*. Proceedings of the Game Developers Conference: 147-166. 2001.

D. Isla, R. Burke, M. Downie, and **B. Blumberg.** *A Layered Brain Architecture for Synthetic Creatures*. Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI). 2001.

Issues of learning and adaptation inside this series are discussed in:

B. Blumberg, M. Downie, Y. Ivanov, M. Berlin, M. P. Johnson, B. Tomlinson, *Integrated learning for interactive synthetic characters*. SIGGRAPH 2002. Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, ACM, 2002.

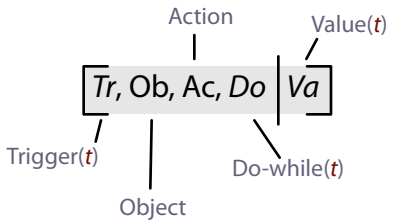


figure 11. The action-tuple consists of up to five parts. The trigger, do-while and value can each appear as a time-varying scalar value.

The c5 action system begins with two structures, the action-tuple and the action group. The action-tuple, which is either active or inactive at any given moment of time, is a unit of action that aggregates the following components: each tuple has at least a **trigger** — an instantaneous scalar value that indicates how relevant or important the activation of action-tuple is to the current situation, in the broadest sense; a **do-while** — an instantaneous scalar value that indicates whether or not this action, should it be active now, is still important (high value) or is "finished" in some way (low value), and thus should become inactive; an **action** "**payload**" — the code that, while the action-tuple is active, should be executed; a **value** — a scalar representation of the value that this action has to the creature, this may be set by hand or learnt. Additionally, many action-tuples possess an **object** — code that can scan the perceptual system to provide an object for the action-payload to operate upon. An action-tuple can be read as a (fuzzy) rule, in the case of *Dobie* for example: when the perception system hears the trainer say "sit" (trigger), tell the motor system to sit (action-payload) near the trainer (object), and keep telling the motor system to sit until it has done so (do-while).

In this way, triggers, do-whiles and objects connect the perception system to action-tuples; the action-tuples when applied send commands to the motor system. The communication for each of these connections is supported by the "working memory" blackboard introduced earlier.

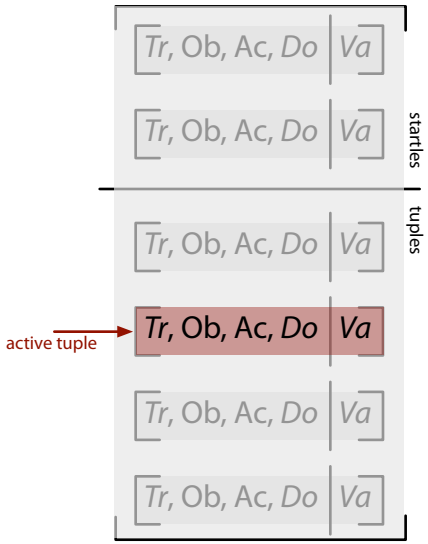


figure 12. Action-tuples compete for expression inside action groups. At any given time an action group has an active tuple.

Action-tuples compete for expression (activation) in action groups based on their *expected* values. An expected value is a combination (a multiplication) of the value of a tuple and its trigger, if it is not active; or its value and its do-while if it is. This captures a sense that a tuple should be both relevant (high trigger) and valuable (high value).

Each action-tuple is inside one and only one action group, and each action group has only one and always one active action-tuple. An action group is responsible for reading the triggers and values of the actions, and the do-while of the active action, and deciding whether to keep the current active action or switch to another one. It is possible to implement an action group in a number of ways, but a good action-group implementation shares the responsibility with the action-tuples it maintains for the behavioral relevance, persistence and coherence of the creature — balancing the short-term reactivity and opportunism of a creature with the medium-term need to keep at an action active for long enough to see some payoff and the long-term need to demonstrate goal directness and exploration of the behavior space. An initial action-selection algorithm, that forms the basis for most c43 / c5 characters, is as follows →

The c43 Action Selection Algorithm

state

startles — a list of action-tuples that get special privilege to interrupt others.

tuples — a list of action-tuples inside this group

currentlyActive — the currently active tuple

lastValues — a mapping from tuple to real number

algorithm

if the maximum *tuple*.expectedValueOf() over all of *startles* is greater than zero then the greatest becomes *nextActive*.

▸ otherwise,

construct the new map *nextValues*[*tuple*] = *tuple*.expectedValueOf() for all *tuples*

if any tuple that isn't *currentlyActive* has *nextValue*[*tuple*]>*lastValues*[*tuple*] and 2* *nextValue*[*tuple*] > *currentlyActive*.expectedValueOf() then select a new action

if *currentlyActive*.expectedValueOf()=0 then select a new action

▸ if we need to select a new action:

if all of *nextValues*[...] = 0 then nothing is done, and *nextActive* = *currentlyActive*

▸ otherwise,

sample *nextActive* from a normalized version of *nextValues*[...]

and set *lastValues*[...] = *nextValues*[...]

For example, compare the Scoot Framework: **S-Y. Yoon, B. Blumberg, G. Schneider,** *Motivation Driven Learning for Interactive Synthetic Characters*. Proceedings of 4th International Conference on Autonomous Agents. 2000.

or the work in **P. Maes,** *How To Do The Right Thing*, MIT AI Lab, Memo 1180, December 1998.

This action-selection mechanism was born of three goals — each a desire for a certain simplicity. The first is to actually reduce the amount of action selection that takes place. Unlike other, more continuous strategies, the whole action system isn't exposed to a re-selection with every iteration; a new action is chosen only in two cases: that the current one has decided itself that it is done (and thus reduced its do-while to zero or a low value), or another action has become much more important than it was the last time selection occurred. Indeed this choice moves to increase the ease with which temporal patterns can be constructed inside this action-selection system. Therefore, this dramatic thinning-out of the temporal complexity of the action system is not for computational efficiency — it is for pragmatic clarity.

This is an authorship position. A behavior author can think through the behavior system they are creating for a longer period of time if it only changes at a small number of well-defined situations. And, in principle, since the short-term temporal dynamics of triggers and do-whiles matter much less in this framework than in others, the author is freed from the uncertainty associated with the degrees of "freedom" afforded by time-constants and gains. Some of the "emergent" behavior that would have been indirectly specified by these filter-constants is set directly, most notably by the structure of the do-while. The do-while clause enables individual action-tuples to finesse their long term likely-hood of preemption by other actions in the system, offering again a more local, and more explicit, control over the temporal dynamics of the group.

The second impetus for the c43 action-selection algorithm is an acknowledgment that some actions need to be handled differently: specifically, the short, transient, but always important actions are set aside into a separate group and have the opportunity to override the current action at any time. In *Dobie*, these *startles* control the initial introductory sequence and the reward marker; in *alphaWolf*, user interaction and reactions to pain; in an early iteration of *The Mu-*

sic Creatures, unexpected sound. This two-level approach decouples the trigger of these transient actions, which often form the fundamentals from which the interaction with the character is constructed, from the rest of the group, preventing an "arms-race" between the values and triggers of these two species of tuples.

The third motivation for c43 is a realization that the actions themselves, if given a stable, long duration life-cycle and a supple motor system can form quite large blocks and that there is little to be gained, other than a certain academic cachet, by making an action system out of completely homogeneous material.

There are a number of extensions to this selection framework that are important, and a number of ways that it directly or indirectly supports learning and adaptation of its parameters.

Hierarchical structure

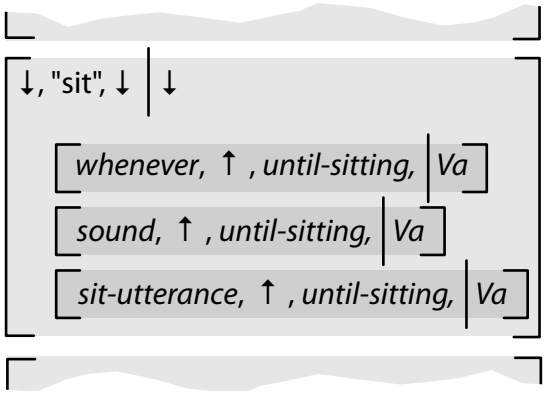


figure 13. Action-tuples can contain action groups. In *Dobie*, functional groupings are developed. Here is a group of actions inside a tuple that share the same action.

With a little care, we can create an action-tuple that has, as its action payload, an action-group. Care is required because it is not initially obvious which one of the many ways of choosing how to generate the external interface — trigger, value — of an action-tuple (from the external interfaces of all of the action-tuples inside a group) one should pick, if any. Synthesizing some mean combination of triggers and values to present to the layer above endangers two advantages that a hierarchical action system might have — a more complete functional separation between hierarchical descendants that do not share an ancestor, and computational efficiency through partial evaluation of the hierarchical action tree.

These advantages can be restored if triggers for whole groups can be specified ahead of time — that these triggers have something to do with the functional grouping expressed through the hierarchy is not too onerous a requirement —

and that values for groups can be adapted on the same time-scales as their children.

That said, there are characters constructed within the c5 thread that exploit this hierarchical flexibility in both of these ways. In *Dobie*, action-groups collate actions together according to action-payload and present triggers and values to parenting groups by computing the maximum of its descendants. *Loops* chooses the second path and groups behaviors together for the purposes of scripting control, creating a semi-modal action-selection structure. *alphaWolf* runs multiple action-groups simultaneously, and contains a few actions that themselves do arbitration, using this second approach.

Complex value

The value part of the action-tuple offers one hook for adaptation to occur. In classic reinforcement learning we find a decomposition of the action selection problem that is not unlike the action-tuple representation. We can draw a table with rows representing actions and columns (perceptual) states. If we were to instantiate this complete table inside an action-tuple setting we would have one action-tuple for each cell in the table, and this action-tuple would have a trigger connected to the perceptual state and an action-payload corresponding to the action. Reinforcement algorithms exist that explore this table and learn the value of each of the cells.

The classic overview of reinforcement learning remains **R. S. Sutton, A. G. Barto,** *Reinforcement Learning,* MIT Press, 1998.

For example, in **q-learning**, for a transition taken by performing action $a \in \{a_i\}$ in state $s \in \{s_j\}$ that results in being in state s' we update the cell $\{a,s\}$ as follows:

$$q_{\{a,s\}} \leftarrow q_{\{a,s\}} + \beta\left(r + \gamma \cdot \max_j \left[q_{\{a_j,s'\}} - q_{\{a,s\}}\right]\right)$$

with β the learning rate and γ a "discount factor" and r the reward signal.

This cell is perhaps seeded with certain actions that are of high value (those that satisfy the motivations of the creature, for example receiving food). The intuition here is that action/state pairings that result in the opportunity to take high value actions are themselves valuable, and during action/state transitions, during, that is, credit assignment, these action/state pairings have some fraction of the expected reward propagated back to them.

Inside the action-tuple-based c43 architecture, this table is forever implicit, and we are free to partially instantiate it; the temporal dynamics of the action selection mechanism is significantly richer. More significantly, for the purposes of more biologically plausible and more authorable learning, we reject q-learning's proposal to continuously update a scalar value for each cell. Rather in *Dobie* we represent "value" with a slightly more complex structure (that, for the purposes of action selection may be turned into a scalar at any moment). This structure keeps track of an expected rate of reward — it remembers that a highly rewarding action is taken after a particular action-tuple at a certain *rate*. If the value of subsequent action is in keeping with this *expectation*, then no value update or propagation need take place.

The motivation for this simplification is in keeping with c43's rejection of individual update cycle dynamics — value propagation does not occur unless something changes in the world, no more than action selection occurs unless something (triggers or do-whiles) changes inside the action-group.

The specific details of these three forms of learning in the case of *Dobie* are well described in the paper summarizing this collaborative work:

B. Blumberg, M. Downie, Y. Ivanov, M. Berlin, M. P. Johnson, B. Tomlinson, *Integrated learning for interactive synthetic characters*. SIGGRAPH 2002. Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, ACM, 2002.

In *Loops* the values of action-tuples are still scalar, but the value interface becomes a window onto a looping script which modifies the values of the action-tuples that read from it (and provides a mechanism for the storage of active tuples), *page 114*. In *alphaWolf* the values of certain actions are coupled to a social-learning memory system developed by Synthetic Characters group-member Bill Tomlinson.

B. Tomlinson, *Synthetic Social Relationships for Computational Entities*. PhD Thesis, MIT, June 2002.

Dynamic structure

figure 14.
Dobie, 2002-3,
The Synthetic Characters Group.

Finally, we look at a powerful species of learning that characters can perform by dynamically extending its action system during the life of the creature in response to training. In the reinforcement-learning example above, the rows (actions) and columns (perceptual states) of the "table" were set initially and the learning algorithm converged on a set of entries for the cells. This solution works quickly only if there are small number of rows and columns, otherwise the time to convergence quickly becomes prohibitively large. Using the sparse nature of the action-tuple representation, *Dobie* hierarchically explores both his state space (the number and organization of columns), his action space (the number and contents of the rows) and the set of pairings worth exploring (whether or not there even is a cell, or here an action-tuple).

In *Dobie*, action-tuples, which start with very general rules that have constant (and thus completely non-informative) triggers, maintain statistics about what finer-grained models in the perception system have better predictive power over obtaining reward. Eventually, new action-tuples, grouped automatically with their more general hypotheses, are created which represent these less general, but more reliable statements of the "rules of the world" for the creature. In this way, new state-action pairs are created through reliability-based hierarchical descent of the perception system structures. *Dobie* might join "sit" (action)

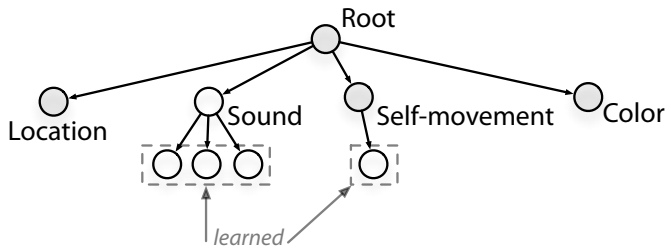


figure 15. A c43/c5's hierarchical decomposition of the perceptual world can be dynamically grown in response to reward signals or other forms of learning.

"whenever" (trigger) with a more specific tuple of "sit" (action) "whenever there is sound".

Similarly, in response to reward signals inside the action system, the perception systems can begin to create finer models of parts of the perceptual world — children of "whenever there is sound" might be specific models of speech uttered by the trainer, labeled by the actions that potentially make use of these models. These new perceptual states become candidates for further state-action pairing in the action-system. *Dobie* might end up with a tuple "sit" (action) "whenever the trainer says 'sit'", having started out knowing how to sit (and the unfading joys of food). Sometimes this modeling of the world is less driven by reward and more by the perceptions themselves: in *The Music Creatures* we shall see a few creatures that opportunistically construct other models of heard sound and — when they do so they store this "knowledge" (which is inherently procedural, it is the ability to *recognize again*) in the perception tree.

Finally, we can build the equivalent of perceptual models of the results of performing by actions that are parameterized in some way — for example if there is an action which corresponds to a complex selection and blending of motor-system material that is directed towards the goal of getting *Dobie*'s nose as close to a point in space as possible (perhaps some food). By keeping track of the specific animation patterns that *Dobie* ends up using to execute this action, if this action results in reward then we can generate new actions that correspond to the performance of these animations. *Dobie* might end up with a tuple " 'roll over' " (action) "whenever" (trigger), having started out with the motor competence, but not the action-system representation for rolling over. Similar learning, in a simpler domain, takes place in *Loops*: the learning of blended rendering parameters. In this case the new perceptual models are saved for later as part of the authorship story — during the "performance" of *Loops* as a work, no learn-

ing takes place, *page 123*. *The Music Creatures* store new perceptual models of action directly inside the motor system themselves, *page 161*.

2. The generic pose-graph motor system

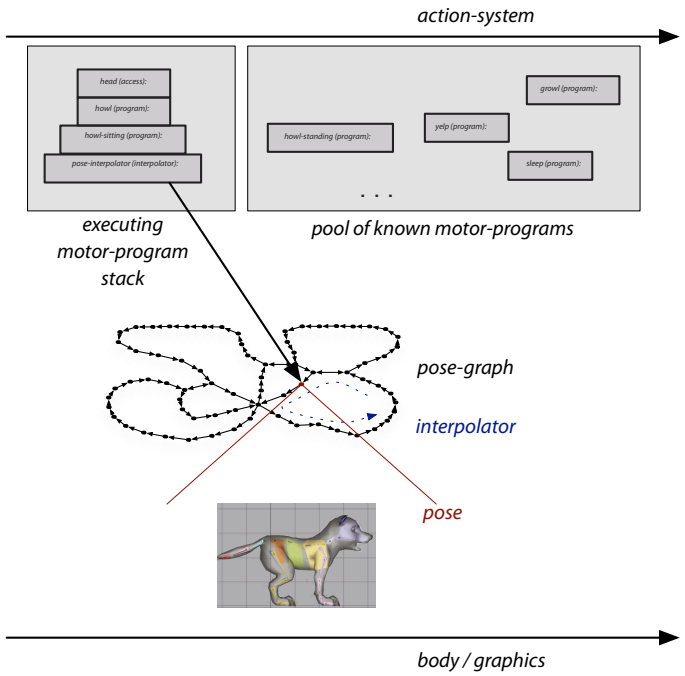


figure 16. The pose-graph provides much structure for the motor systems of agents. The action system communicates with a pool of "motor programs"; in the most basic case this produces a stack of executable programs. These programs in turn call upon interpolators that play back paths through the pose-graph structure.

If the perception system of an agent is where the demands, time-scales and structures of the world first meet the needs, flows and internal preferences of the agent, the agent's motor system occupies a similar, but reversed role: interpreting the selections made by an action system into a control structure, perhaps a control *plan*, executing it, and monitoring and reporting on its progress.

An action system may ask a virtual dog to walk over to an object in the world and "sit"; it's a motor system that will be responsible for the *sequencing* of animation material onto the body representation of the creature in order to cause movement and ultimately sitting. Lower levels of the motor system ought to be able to answer questions concerning whether another step or walk cycle should be taken — knowledge about the *constraints* of the control of the body are stored here. And, as we shall see, the motor system as a whole ought to be able to answer questions concerning the outward appearance of the character — does our dog appear to be sitting or is it "standing up" — knowledge about the *contents* of the body representation are also exposed at this location.

My master's thesis: **M. Downie,** *behavior, animation, music: the music and movement of synthetic characters*. S. M. Thesis, January, 2000.

Graph structures in computer graphics are by no means unique in motor-system issues. Prior to my thesis:

C. Rose, *Verbs and Adverbs: Multidimensional Motion Interpolation Using Radial-Basis Functions.* Department of Computer Science. Princeton, NJ, Princeton University. 1999.

And more recently, **L. Kovar, M. Gleicher, F. Pighin,** *Motion Graphs.* 2002. Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, 2002.

and on the automatic generation of graphs: **L. Kovar** and **M. Gleicher,** *Automated Extraction and Parameterization of Motions in Large Data Sets.* Transactions on Graphics, 23, 3. SIGGRAPH 2004.

The italicized words above — planning, sequencing, constraints and contents — are the essence of the motor system's task. We shall see throughout this thesis a wide range of body representations and control structures, indeed, one of the contributions of this work is a demonstration of this range. Almost all of these bodies and control structures have been implemented within a graph based motor system framework that we will call here the pose-graph. This structure was first constructed in 2000 and, although it has been re-implemented twice since then, its generality allowed the idea to survive intact through collaborative and solo work over the last four years. A brief review of the original contribution is supplied here for completeness and to ground the subsequent recent extensions and revisions to the framework.

A pose is a fundamental atom of a pose-graph motor system. In a representational creature it might be equivalent to a keyframe of an animation — a complete description of the position of a classical, hierarchical, joint-angle based digital figure. This dealing in terms of poses rather than key-frames, animation tracks, or joint angles is the first abstraction that the pose-graph structure offers. Additionally, poses may be parameterized — for example, a single pose might represent a key-frame bundle of a frame of a walk-animation that is itself re-computed based on blending several walk-animations together.

In the pose-graph motor system, poses, as you might expect, are arranged in graphs — specifically graphs that are directed and strictly cyclic (where all nodes are accessible from all others). Paths through these graphs are, in the case of a classical digital figure, the raw material from which animations can be created. In order to build an animation from a path through a graph of poses we need one additional piece of information — the durations between the key-frames. We call this the time-metric.

This graph structure becomes useful for two main reasons: because there are well-known algorithms for manipulating and inspecting graphs (in particular searching them, but also comparing them and editing them) and because we can specify operations at the level of the abstract-pose or chain of poses rather than at the joint level — in the simplest case, at any given time the configuration of the body of the creature is completely specified as either being at a pose graph vertex or at some point along a pose-graph edge. These two abilities allow some efficient abstract reasoning about and manipulation of poses to be constructed without committing to a particular pose representation.

For example, the standard search algorithm used for generating animations from pose-graph poses is the A*-search, for similar uses: **J.C. Latombe**, *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

A pose-graph motor system spends much of its time producing animation material by searching for poses from the graph that match certain criteria, then searching for paths to those poses from the current body configuration and then finally building "interpolators" that execute that path. To search for a path between two nodes in a graph we need another metric (which might be related to time, but is more likely related to an estimation of "effort") which we shall call the distance-metric. Two related distance metrics are possible (and in general needed for fast searching) — metrics between pose-graph nodes that share an edge and a distance between any two poses. To search for nodes of interest in the graph we need labels and other information stored in the graph. Sometimes these labels are simply names of poses of particular interest ("sitting" for example), sometimes these labels are the positions of part of the creature's body — the dog's nose. This decomposition of searching into two unrelated and relatively efficient searches — a search for a node, and a search for a path — works well for a number of problems.

both these metrics are used in the A*-search algorithm that the pose-graph uses to find shortest paths between nodes; it is a sufficient condition for the admissibility of the pose-pose metric as a search heuristic for the node-node metric for the node-node distances to never be less than the pose-pose of their respective poses.

Thus the pose-graph motor system seems to have something to offer the problem of *planning* and *sequencing* movement and its graph structure is one representation of the *constraints* on motion placed on a body by the availability of

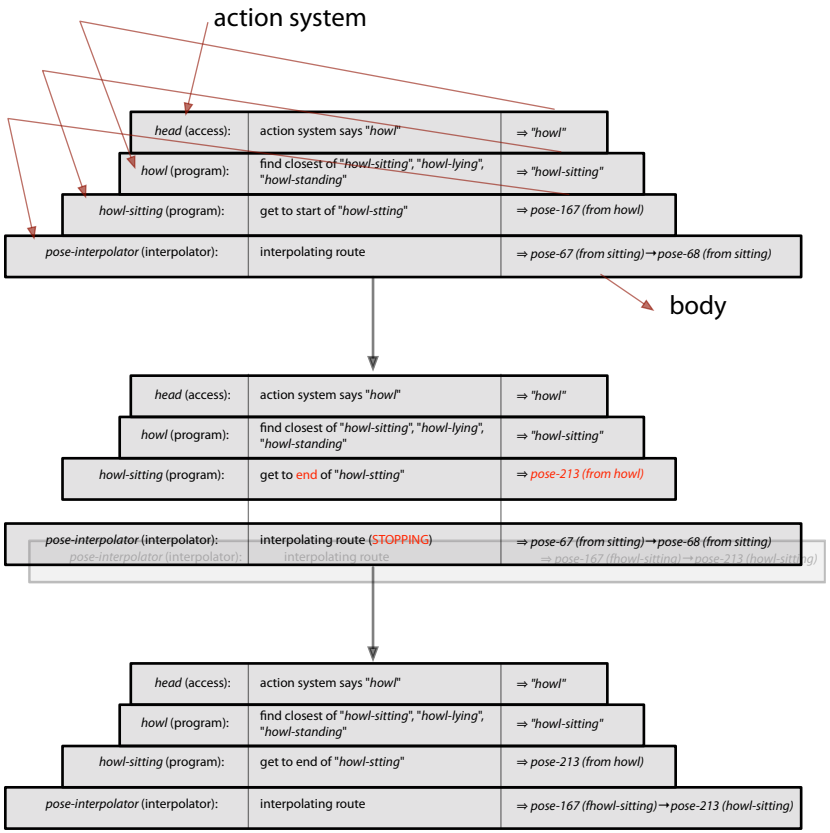


figure 17. Initially, the stack consists of a "head" element that is connected to the action system. Each program executes and requests that the pose-graph move to a particular object — which might be another program (in which case this is placed on the stack beneath) or a pose (in which case an interpolator is created, and the stack ends). The stack of executing motor programs undergoes a complex tear-down process whenever an element of the stack changes its output.

animation material *content*. To complete the basic pose-graph framework, a few more ingredients are required.

Firstly there needs to be a **control language**, situated between the desires of an action system and the constraints of the pose interpolators. One particular, "stack-based" language certainly found some amount of traction in the collaborative works presented here. This virtual machine for motor programs consisted of a running stack of control elements — the "output" of element n is resolvable into a request for a particular element $n+1$. Should element n ask for something and never change its mind, element $n+1$ continues to execute. Should element n ask for something else, the stack elements $> n$ are torn down, and a new set of stack elements are recursively constructed. Stack element 0 is a connection to the requests of an action system and at the top of the stack, element N, is a pose-graph interpolator. This construction worked well for a wide range of creatures — allowing the creation of modular, reusable, programs that shielded the action system from the current contents of the motor system — but began to show its age and its simplicities with more complex configurations.

In particular, for most creatures the identity between a single pose edge in a single pose-graph and a complete description of the creature's body is just too simple. Most representational creatures require the creation of several pose-graphs, which execute in parallel and in general control various layers of the body's motion. There might be a "tail layer" or a "face layer". The description of a body configuration is then, at the very least, the pose-graph edge positions of all of the pose-graphs, the parameters to the poses either side of those edges and some description of how the independent pose-graph layers blend. The analytic power of the pose-graph is not lost if, for example, there is one main pose-graph that controls the bulk motion of the creature, if there is one place to go to ask if the creature is sitting or standing. But the expressive power of the stack-based

motor programs are compromised if there are constraints between various layers. One critique and replacement is offered for *The Music Creatures, page 152*, and for agents where the decomposition into layers offers insufficient granularity, where motor programs that access different resources and target different bodily degrees of freedom need to run in parallel rather than series, this, more sophisticated motor-program "language" seems vital. The agents of *how long...* operate in this domain.

Secondly, we need some algorithms that can supply some analytic, or **perceptual representations** of what the pose-graph is doing. One is offered in *Dobie* — an algorithm that can construct a data-driven model from several, multiple pose-paths. This means that *Dobie* can learn that a particular flow through the graph, one that might have no specific motor-program to create it, is important enough to be named and represented as a possible action in the action-system.

Another is to use the pose-graph as a place to store learning information that can later be recalled in order to provide an analytic view of the motor system. This approach has many uses and the pose-graph, in the agents that use it, seems an ideal representation in which to store information about the relationship between motion and something else. One satisfying result is in the integrated learning of *Dobie* where, by incrementally labeling poses with the actions that are running when they are executed, an agent can reflect upon the appearance of its own body — in effect, averaging over the complexities of motor-program stacks that interface between action and movement. Such a capacity is one of the differences between a machine learning problem and a creature-*training* problem — how the agent *appears* is all a trainer gets to see. In this case it's extremely important that the trainer's reward information is propagated to the correct actions — not to the actions that are currently executing but rather to the actions that are generally responsible for making the creature look that

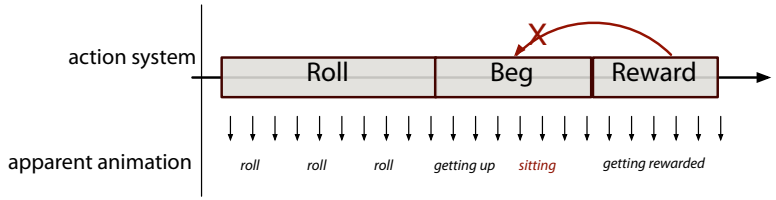


figure 18. The body does not always appear to be the same as the running action. Fundamental to the problem of building motor systems is having them be able to provide information about how the body *appears* to the outside. In the case illustrated above, the action system tells the motor system to perform the action "beg" — however, *Dobie*, in transitioning from "roll over" to "beg" appears to be sitting for a moment. It is the action that causes "sitting" that ought to be rewarded.

way. A trainer rewarding *Dobie* the virtual dog for sitting is rewarding the appearance of sitting — and the fact that a complex stack of motor programs is at this instant preparing to make *Dobie* stand is hidden and thus meaningless as far as the trainer is concerned.

We can store this information in each pose as a simple distribution $p(\text{action})$ and we can compute the entropy of this distribution which indicates whether this is a pose that is strongly associated with the particular action or not. We can improve these histograms, decreasing their sensitivity to precise system timings, by smoothing these labels along the graph edges.

In *The Music Creatures* we have examples of storing sound in the pose-graph — the *line* agent, *page 141* — and of learning the results of more control-like poses — in the *exchange* agent, *page 135*.

The usefulness of storing this information, and propagating it around the graph structure hints at a more general extension of the pose-graph into hierarchical structures, that allow the storing of label information, and perhaps even pose information at a variety of resolutions. Indeed, as the richness of the information stored in the poses increase, so does the chance that our search problems will not decompose into node-finding and then path finding. Searching for a node in the graph that has certain apparent properties that are dependent on the path that the agent would have to take to get there. To find these kinds of nodes takes, at the very least a full $O(n^2)$ search of the graph (a full, carefully ordered, breadth first search) with $O(n^2)$ storage — although the A*-search algorithm also scales as $O(n^2)$ its effective leading constant is much, much smaller given the good pose-pose heuristic information. Hierarchical graphs

support a downward, heuristic beam search that reduce this search to $O(n\log(n))$.

There are a variety of ways of creating low-resolution versions of connected graphs. Those most useful for searching maintain the same connectivity information at lower-resolution views as found in higher-resolution views, looking at the connectivity of nodes as well as their contents when deciding to merge nodes. Since these graphs are completely cyclic anyway, this is not a hard constraint, but a heuristic. We should expect to find that the higher-resolution children of two adjacent low-resolution nodes are also reasonable closely connected. In the hierarchical pose-graph we collapse chains of singly connected nodes — common in graphs created from long animations — into fewer nodes by unsupervised clustering while converting nodes that are centers of star configurations into larger models centered on them. At any given time, the agent's body is located now not only on a pose-graph edge or node but on a whole set of increasingly lower-resolution pose-graph edges or nodes. Poses that fit some description and are "close-by" in terms of the shortest path from the current position can now be found with standard hierarchical graph searching techniques. These techniques are used in the motor learning of the *exchange* creature in *The Music Creatures*, *page 141*, and also form the basis for some of the motion identification work described.

A review of hierarchical search methods in such ad hoc generated hierarchies is: **J-A. Fernández-Madrigal** and **J. González**, *Multihierarchical Graph Search*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (1), January 2002.

For a use of the hierarchical motor system work described here:

C. L. Breazeal, D. Buchsbaum, J. Gray, D. Gatenby, B. Blumberg, *Learning From and About Others: Towards Using Imitation to Bootstrap the Social Understanding of Others by Robots*. Artificial Life, 11 (1), 2005.

The generic *pose-graph*

It is worthwhile to stand back and look at the more general implications of pose-graph-based motor systems in a more digital-art rather than digital-agent context. The triple space navigated by the motor system — action selection, virtual body and animation material — is particularly interesting. The pose-graph was originally constructed as a structure that could take *content* in the most banal sense of the word — unstructured animation produced by hired

animators — and reuse it, live. This repurposes and exploits the animations, the animator's talents and the extremely polished and researched tools that animators use. As a site of content the pose-graph explicit representation of material seems inherently collaborative. In *Loops* the pose-graph structure became the representation used to manipulate not *movement* (the underlying motion used in that work simply played out on a loop) but *choices* — rendering parameters and action-system parameters named by the collaborators. The explicit graph structure — its graph of statements that "this" is a pose and it has "this" name — became the sketchpad for the solidification of the collaboration, the memory of where we were. As such it became a site of versioning information and database techniques, *page 107* and later, *page 225*.

3. Critiquing c5, *alphaWolf*

AlphaWolf was the vision of Bill Tomlinson and is described in detail in: **B. Tomlinson,** *Synthetic Social Relationships for Computational Entities*. PhD Thesis, MIT, June 2002.

The c5 action-selection mechanism is an example of an organizing structure rather than a prescription for action selection in general. Each "slot" in this template-like description can be filled by code of incremental complexity; the abstraction afforded by c5's action groups has survived for a number years and a number of projects because it offers the right blend of structure and openness for a number of problems.

Much of this approach's gentle innovativeness was devoted to controlling the temporal patterning of the resulting action selections. But it is exactly here that there was more work to be done. The overview of the c5 action-selection mechanism given above was an overview because it omitted two complications. The full algorithm, as ultimately deployed in *alphaWolf* and *Dobie* (for *Loops* the difference is immaterial) is given here →

Modified C5 Action Selection Algorithm

state

startles — a list of action-tuples that get special privilege to interrupt others.

tuples — a list of action-tuples inside this group

currentlyActive — the currently active tuple

lastValues — a mapping from tuple to real number

define expectedValueOf(*tuple*) to be *tuple*.trigger()*tuple*.value() if tuple is *currentlyActive* or *tuple*.doWhile()*tuple*.value() otherwise.

algorithm

if the maximum expectedValueOf(...) over all of startles is greater than zero then the greatest becomes *nextActive*.

otherwise, if the *currentlyActive* is a startle, and expectedValueOf(*currentlyActive*)=0, and *tuple*.trigger()*tuple*.value() is greater than zero, *currentlyActive* remains active and *nextActive=currentlyActive*.

▸ otherwise,

construct the new map *nextValues*[*tuple*] = expectedValueOf(*tuple*) for all *tuples*

if any tuple that isn't *currentlyActive* has *nextValue*[*tuple*]>*lastValues*[*tuple*] and 2* *nextValue*[*tuple*] > expectedValueOf(*currentlyActive*) then select a new action

if expectedValueOf(*currentlyActive*)=0 then select a new action

▸ if we need to select a new action:

if all of *nextValues*[...] = 0 then nothing is done, and *nextActive* = *currentlyActive*

▸ otherwise,

sample *nextActive* from a normalized version of *nextValues*[...]

if we selected because expectedValueOf(*currentlyActive*) was 0 set *nextValues*[*currentlyActive*] = *tuple*.trigger()*tuple*.value()

◂ finally, *lastValues*[...] = *nextValues*[...]

Note the three highlighted lines in the description of the action selection mechanism. These, rather explicitly, avoid a common but troubling scenario — when the do-while of an action-tuple goes to zero, but the trigger of an action-tuple is non-zero, the action-selection mechanism is likely to dither (swap back and forth) for exactly one iteration. Why? because the action-tuple appears to the action selection mechanism to go from zero to a non-zero value in one iteration, this causes a "surprise" based reselect. The first line prevents this phenomenon from occurring in the case of startle action-tuples, the second in the case of normal action-tuples. In order for these "workarounds" to be written at the action-group level the action-tuple must present all the pieces of its expected value function, coupling its internal value structure directly to the group. (There is an inverse situation, which was always solved at the tuple level, where a high trigger leads to a zero do-while, that again causes a one-iteration dither.)

However, these corrections do not remove all temporal pathologies from the action group formulation. If an action's trigger depends on the currently running action of the group, or if it depends on another action in another group which in turn connects back to this action, multi-iteration dithers and even freezes are possible. Such cases have appeared a number of times — during the development of *alphaWolf*, where the creatures' action systems were split into two separate, but very coupled groups. This coupling is perfectly reasonable on paper — sometimes, the "attention" group controlled the palette of options for the the main "action" group, sometimes the what the wolf was doing controlled what it could pay attention to.

Other aspects of the extant c43/c5 creatures are already pushing along this path of temporal complexity. Traditional reinforcement learning systems propagate value between state-action pairings at each state-action transition. We have seen that c43/c5 potentially drops this credit-assignment stage altogether should the upcoming value be expected. However, in agents with complex bodies and mo-

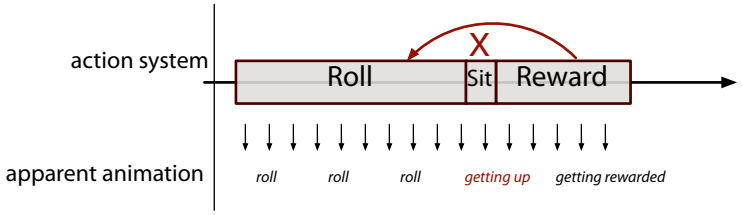


figure 19.Sometimes actions should not participate in credit assignment. In this case the action "sit" is too short to have an effect on the body of the character — "Roll" should receive the reward instead.

tor systems this credit-assignment stage is further complicated. It is necessary (in the sense that it is often the difference between a "trainable" character and a non-trainable one) for creatures with complex bodies to defer this decision to propagate value to a later point in time. Since the action system necessarily runs ahead of the motor system, issuing commands that will take a number of frames to have an eventual effect on the body (due to animation constraints and the constraints of realistic motion) a creature must take care to allow only actions that had a shaping influence on the body of the character to participate in the value propagation.

The reason for this descriptive detail (in addition to the benefits of accurately recording the action selection algorithm for *alphaWolf*, *Dobie*, *Loops* and an early music creature for the first time) is to illustrate where the tensions lie in taking the simple intuition behind the c43 action selection mechanism and turning it into an algorithm. The general lesson: that unexpectedly difficult intricacies involving the patterning of time result from surprisingly simple systems. The specific lessons: that if one really thinks of the action selection problem in terms of not only relevancy and persistence but *patterning* — the order and timing of actions that arise from an action system — then c5 appears deficient. As we move from what one would call patterning, towards actions systems that distribute and control multiple interacting actions simultaneously, what I might be tempted to call *choreographing* actions, then one might need an additional level of description, additional state and additional control mechanisms than those offered by c5's action-group.

Locating c5

Before proceeding to the artworks created with c5, it's worth pausing to locate c5 in the space of possible action-selection solutions, considering in more general terms the problems that it solves well and the kinds of problems it has little to offer. We'll reuse the axes later to work out where some of the contributions made in this thesis fit. These axes are judged from the point of view of the author of an agent, when they are trying to think through or think over the behavior of the system. For the purposes of comparison we'll also include the popular graphical environment Max, which will be the subject of considerable discussion in the last chapter; and a hypothetical placeholder entitled xFSM that notates a variety of small, finite-state machine techniques that appear in scripting contexts or simple computer games. Finally, Maes89 refers to Maes's influential "How to do the right thing" architecture; and Scoot, an older, hierarchical Synthetic Characters Group architecture.

P. Maes, *How to do the right thing,* MIT AI Lab, Memo 1180, December 1998.

Scoot: **S-Y. Yoon, B. Blumberg, G. Schneider,** *Motivation Driven Learning for Interactive Synthetic Characters*. Proceedings of 4th International Conference on Autonomous Agents, 2000.

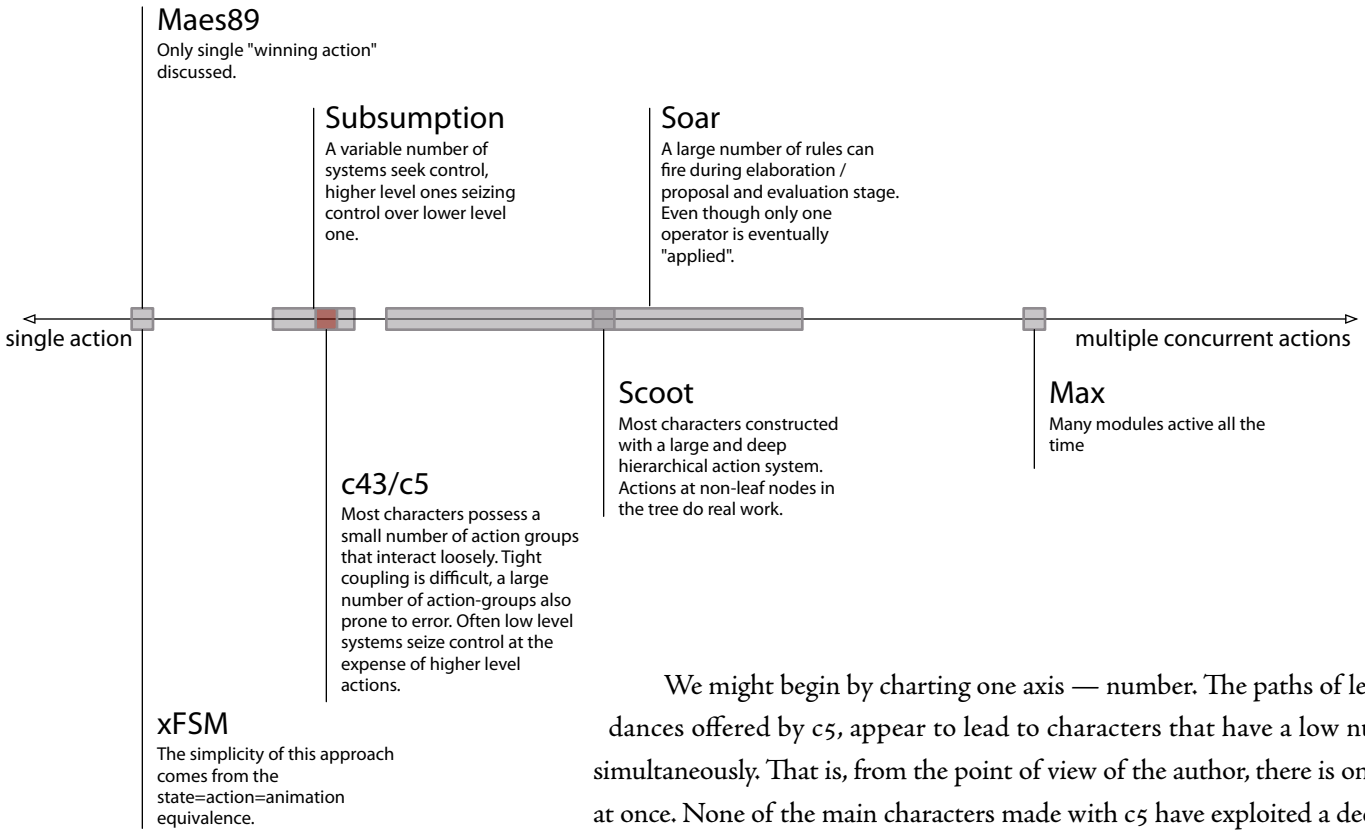


figure 20.
Single-actions versus multiple actions.

We might begin by charting one axis — number. The paths of least resistance, the affordances offered by c5, appear to lead to characters that have a low number of actions active simultaneously. That is, from the point of view of the author, there is only a few things going on at once. None of the main characters made with c5 have exploited a deeply hierarchical action-system, and this is probably because c5 says little about, and potentially has trouble over, action-tuples that interact with each other "behind the scenes". Towards the opposite end of this axis we locate hierarchical action structures (for example, Scoot) and other structures where the composition of simultaneous actions and finally the composition of simultaneous, interacting actions leads to the expressive power of the action-group. As we proceed through this thesis we will move from left to right — as we meet, as artists, the duration of a dance, or the installation of an artwork, we will require more complex, more self-generating temporal structures based on multiple overlapping actions.

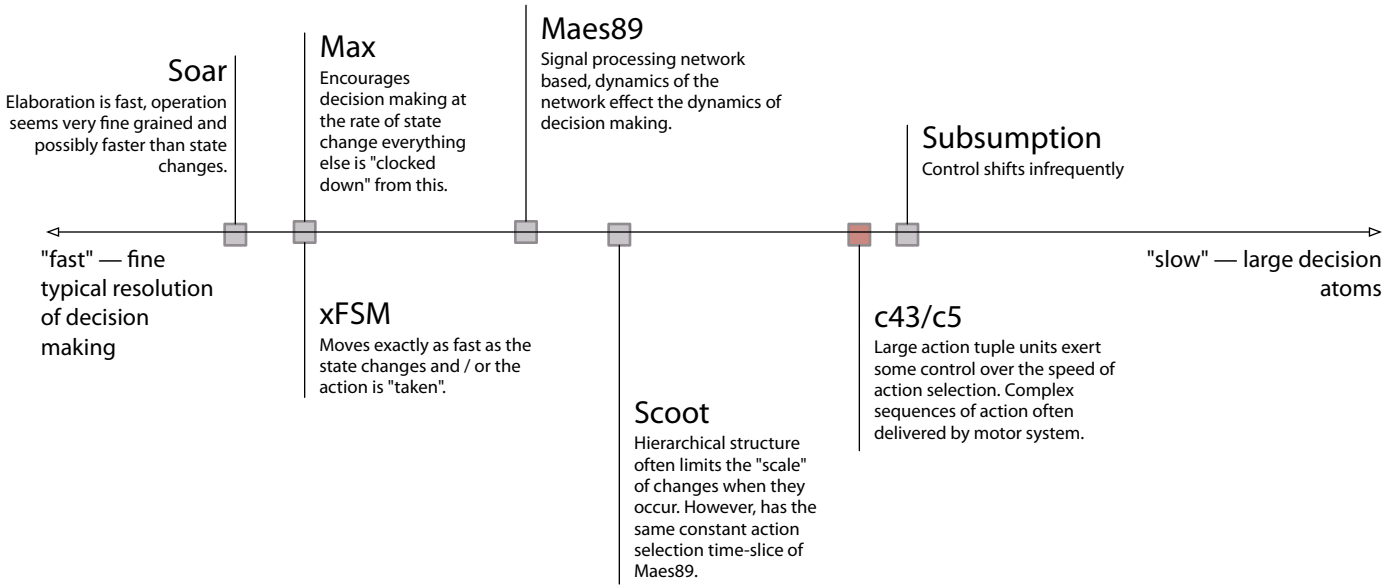


figure 21.
High resolution versus low resolution action selection.

Another axis — **speed**. What is the typical granularity of decision-making inside the action system that the author is actually working with (or against)? Systems that make decisions based on elaborate filter networks count here as fast; Planning-based systems must potentially discard a large amount of computation to change their decision count as slow. C5, as we have seen, picks a middle ground, selecting infrequently and allowing both active actions and inactive actions to semi-explicitly recommend re-selection. As we proceed through this thesis we will fight to maintain the correct granularity of decision-making and augment c5 with both long and shorter time-structures.

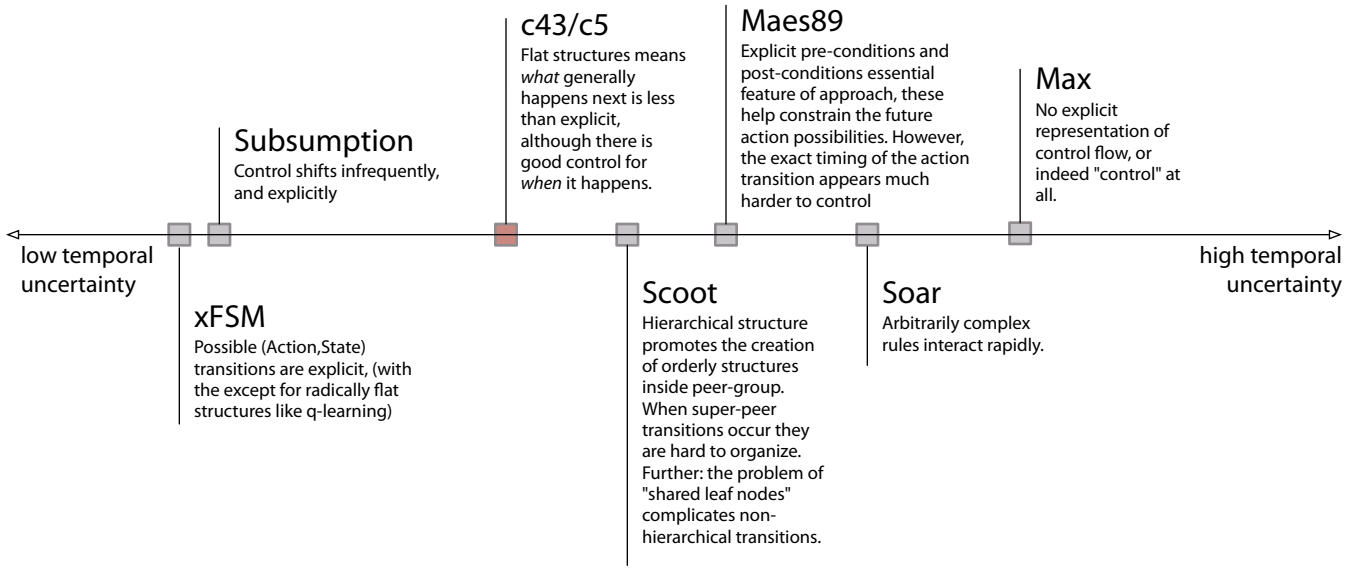


figure 22.
Low versus high "temporal uncertainty". What is the range of things that can happen when?

The final axis — **temporal uncertainty**. What is the typical potential temporal complexity of the results of action selection — both its "texture" in time and its scope? Here scripting systems and finite-state machines count as low — these are often the kinds of systems that are used in computer games particularly because they are considered simple, and "safe" to use. C5 is significantly further along than this, with its broad, flat action-groups (that have a lot of scope) and its fairly unconstrained transition dynamics. Without taking extra steps patterning sequences of c5 action-tuples is left entirely to the author, often delegated to the motor system which has significantly stronger language for talking about constraint and ordering. I will describe ways of simultaneously increasing the temporal complexity of agents and their action selection techniques while reducing this "temporal uncertainty" from the point of view of their authors.

Some of the work of the latter half of this thesis is to broaden the range of these axes that is accessible from c5-like systems. Some artworks — *Loops* and *Loops Score* — will need more actions ongoing simultaneously and interacting with each other, and I'll present two ways of achieving access to this part of that axis. Before moving onto those artworks, there is an installation that will draw the advantages and disadvantages of c5 into sharper focus.

alphaWolf, *a large c5 installation*

figure 23. *alphaWolf*, 2001, The Synthetic Characters Group.

alphaWolf was a project by the Synthetic Characters Group during the year 2001, completed during an intense collaborative period during the summer, premiering at the SIGGRAPH computer graphics exhibition. It is by any metric a success, multi-participant interactive work. It later toured to a number of venues, including the international Ars Electronica festival and German ZKM institute. Since in terms of the number of people actively contributing code, and the numerical size of the perception, action and motor systems it contains, it is second only to *how long...* it is important to take this opportunity to examine a concrete example of a complex work created within the c5 agent toolkit and to learn the lessons latent in this unquestionably successful installation.

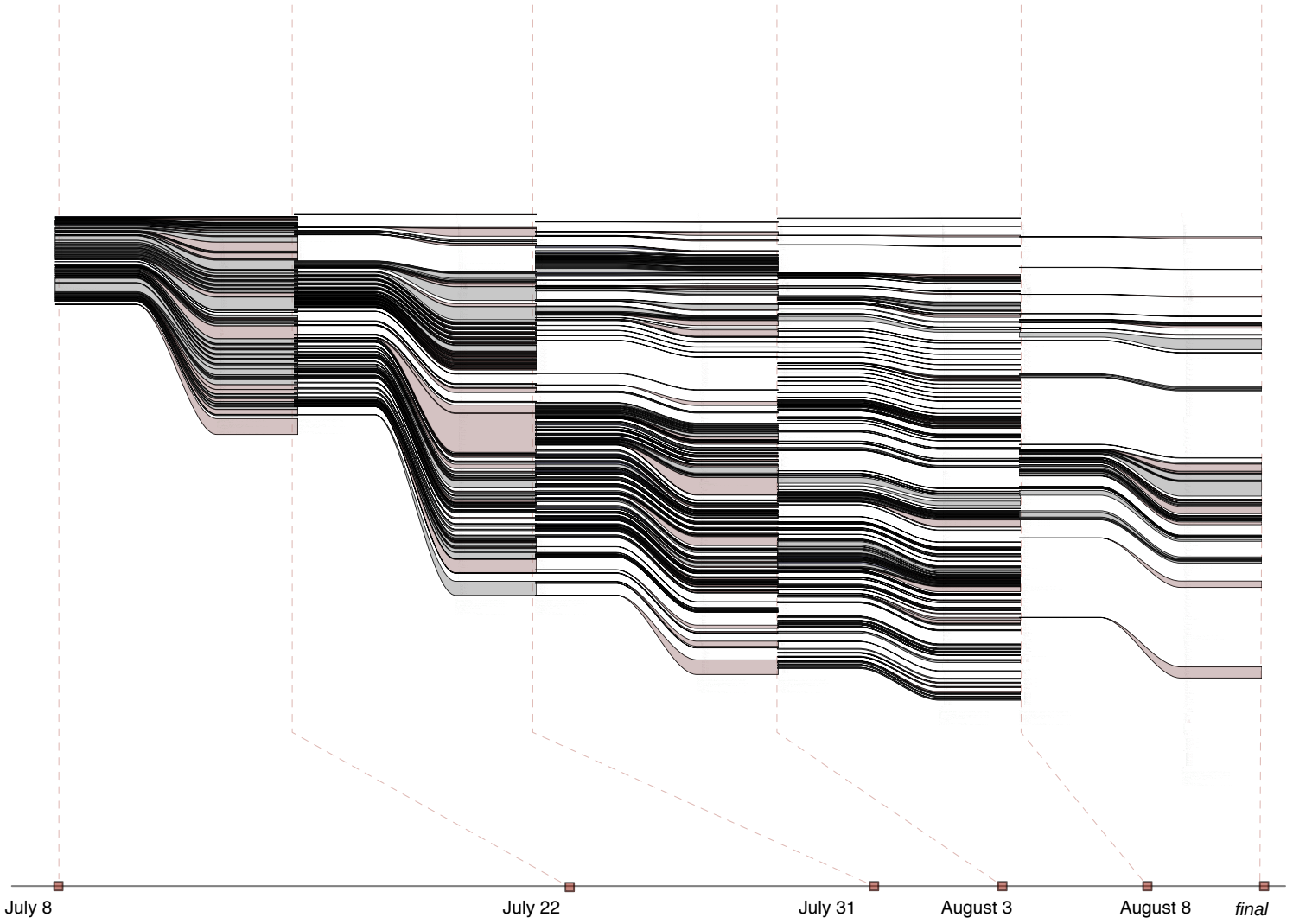


figure 24. The main action system for a wolf in alphaWolf was one of the main sites of authorship during the development of the piece, and grew seemingly without bound. Changes (gray) and additions (red) continue to be distributed throughout the file even up until the last day.

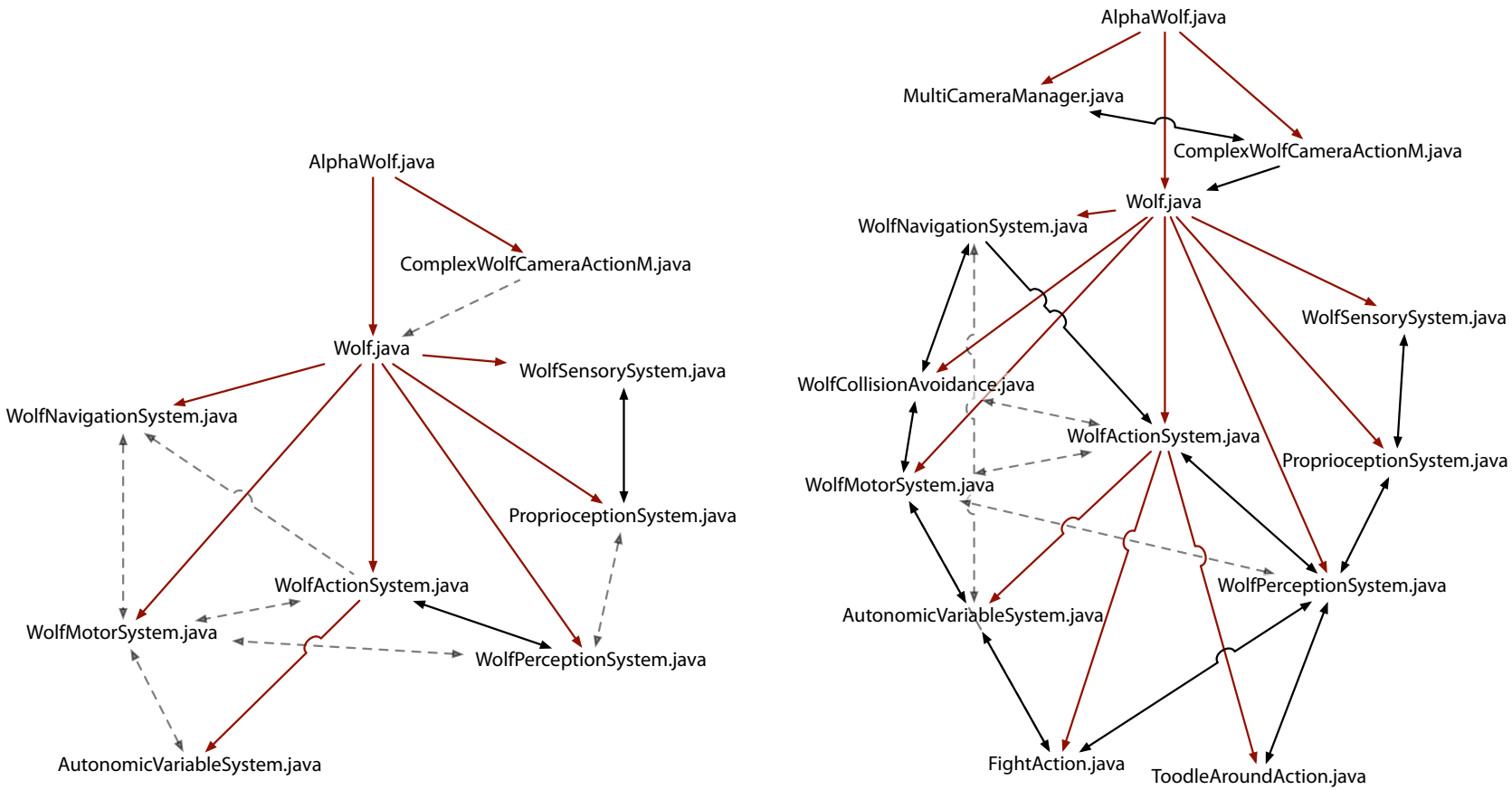


figure 25. Over time, everything becomes connected, and coupled, to everything else. The left figure, taken from early July shows how the systems are instantiated (red), directly connected (black) or weakly coupled through the wolf working memory (dashed, grey). Over time, however, these "abstraction barriers" crumble and a densely connected set of codependent files result.

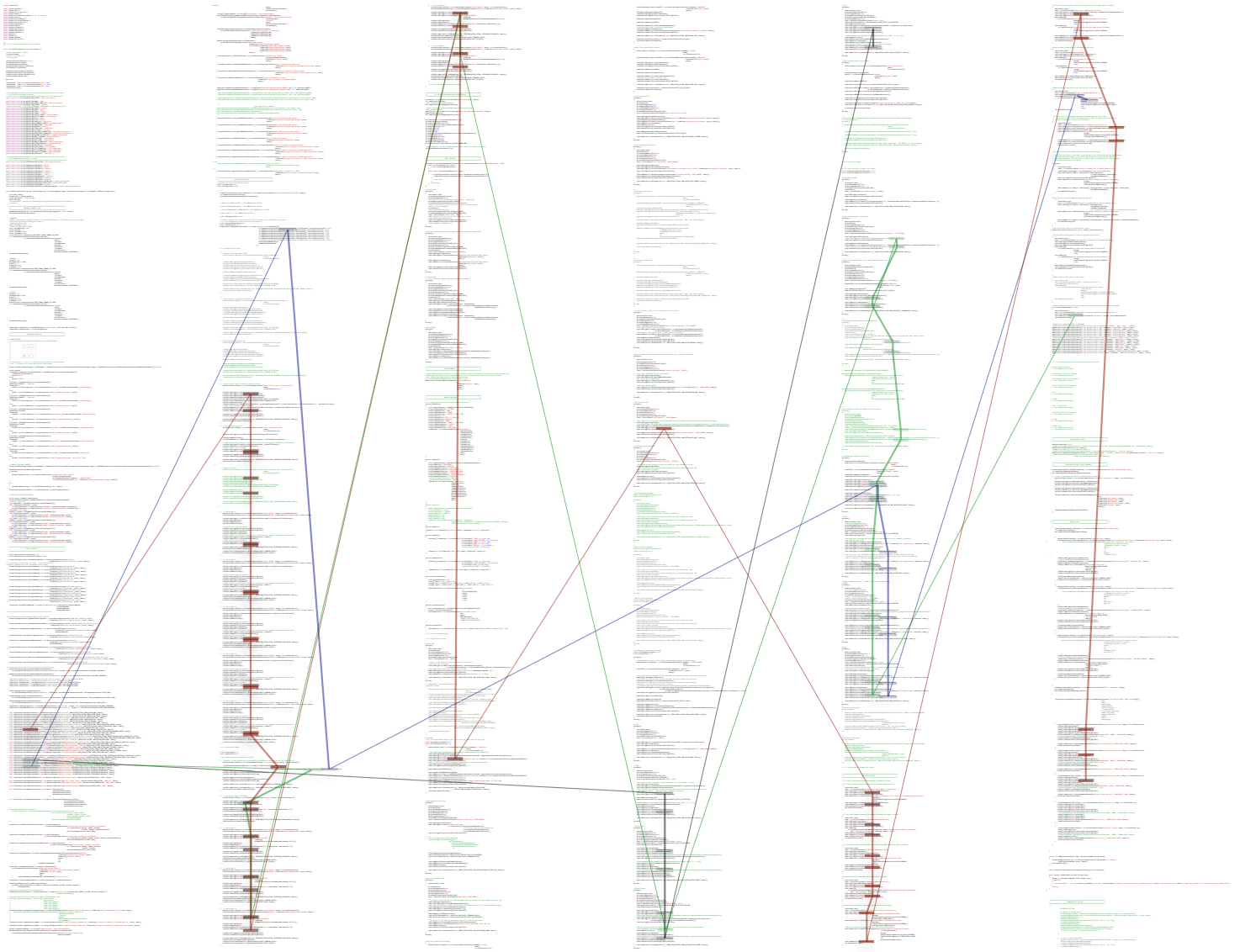


figure 26. Inside the main action system file independence between action-tuples' activations is impossible to maintain. This figure traces the thread of four "contexts" for triggering action-tuples that are themselves based on which action-tuples are active. Note how the colors weave in many different combinations — no single hierarchical reorganization of this file will capture the complex temporal patternings that these threads are producing.

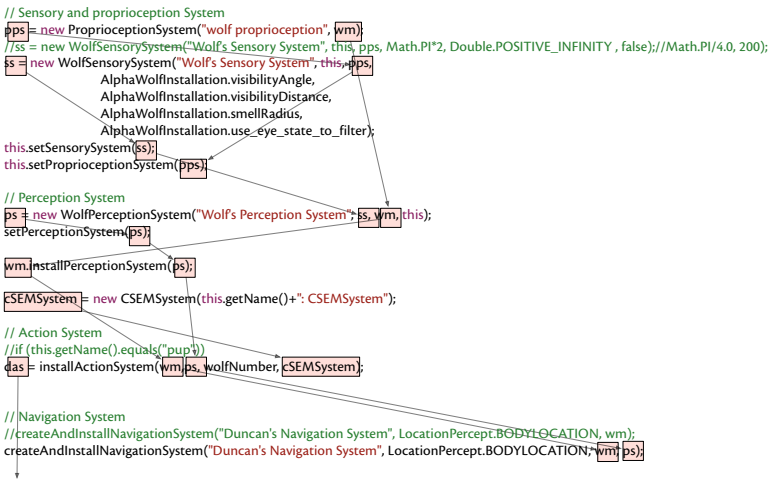


figure 27. The above, annotated segment, comes from the main Wolf character's constructor. Note the complex, coupled, and order dependent, initialization of subsystems.

By a "close reading" of the archives of this project I am tempted to find the following areas as problematic for the creation of complex agents in general.

Initialization, connection, registration and notification. All of the lines of Alphawolf.java (the main installation file) and Wolf.java (the main initialization file for a Wolf creature) that do not create a system are devoted to registering systems with each other and passing references to systems through constructors so that they can perform their own registration and connections. Early versions of *The Music Creatures* — as they cut new paths away from the toolkit, were similarly plagued by critically order-dependent but otherwise boilerplate instantiation code. This is the creation-time coupling issue, and in more complex and heterogeneous creatures there is a tenancy for this problem to increase in severity. Indeed, in *alphaWolf*, the burden of reconfiguring and disconnecting a creature is such that no creature is ever deleted from the work, at the end of a 5-minute interaction cycle, with the pups having "grown" into adults, the creature is reused rather than recreated.

→ I develop the Context Tree and a set of associated techniques to combat this problem.

No re-use or extensibility. The initialization of the wolf agents in the main behavior file WolfActionSystem.java is almost completely monolithic — it is inconceivable that as a description of behavior this class could be extended (or, in the language of object-oriented programming, sub-classed) by another to create a variety of wolf or a wolf with a particular behavioral style. Evidence for this hypothesis is present in the very form of the file, which includes in one overlapping place the descriptions for the wolf pups, the wolf "aunt" and the alpha and beta adults. Yet, at the same time, this complex initialization routine seems have little space left for yet more parameters and options.

→ I develop an extensible programming model based in the Context Tree that approaches the problem of creating extensible complex assemblages of systems with "over-ridable" or "sub-classable" default behavior.

Chains of actions are hard (B happens after A finishes), **deferred execution is hard** (B doesn't happen while A is still ongoing) — These two temporal primitives are so hard to express inside the body of the main agent behavior file that even a fairly close reading of the code will not yield this secret issue. Only by executing and debugging the code will one realize the careful choreography of signaling between one action-tuple and another, between one mode and another. The chaining of actions hides a subtler problem: Action-tuples whose values are based on the current activation of another action short-circuit some aspects of the action-selection mechanism, and reveal the complexities that the action selection technique was constructed to hide. Part of the issue stems from the occasional pathological cases in the action selection framework, some of which are described above *page 87*, but much of the problem is that there simply isn't any framework support for such chains of actions.

→ The Diagram system is constructed around making explicit the temporal sequence of events, incorporating some of the techniques that motor systems use to handle their temporal sequencing, and the Fluid tool renders the authorship of these sequences very visual while retaining the expressive power of programming languages.

That **execution ordering** is significant and difficult is also hidden inside the action-system file. Here WolfActionSystem.java goes to some lengths to ensure accurate hand-off between actions is reflected in the creatures "working-memory". Should a working memory slot go unfilled, or perhaps un-overwritten for one execution cycle, there are places where the action system stalls and the behavior of the creature breaks down. This suggests

figure 28. This figure shows the "optional" sections of the main action system file — areas activated in the cases that the wolf in question is the "alpha", "beta" or "aunt" wolves rather than an interactive wolf pup. Note also the fine scattering of numerical constants that control the behavior, one might even say the "character" of this particular character

that there is a disconnect between the time-scale that the actions are operating on and the amount of persistence that their effects should have.

→ I develop the generic radial-basis channel as a technique *page 135*, borrowed from the problem domain of motor systems, as a way of allowing actions to have short term effects that extend past their activation stories in a flexible way, diffusing the coupling between action and life-cycle.

Insufficient modularity of systems — things which on paper seem like they should be modules couple so much information into the main behavior class that they are added inline rather than as a module. We can see this in the perception-system elements, user-interface elements (the on-screen buttons for participants to interact with) or the action-system manifestations of device controllers. Other systems which are specified elsewhere (for example the perception system) have much of their structure duplicated as they too are coupled in (for example in the list of source action-tuple triggers). Such a static duplication of structure is unmaintainable in a creature that undergoes structural learning, for example *Dobie* overcomes this problem in a specific case.

→ We shall see the use of a Context-Tree-like structure to provide a easy integration and de-integration of systems, *pages 231*; generic dynamically extensible views of hierarchical structures which will form the basis for the Fluid environment, *page 397*.

Insufficient modularity of behavior — Even from the overview of successive differences between versions of the main action system file of *alpha-Wolf*, we can sense large blocks of code being introduced, scattered into the middle of the main initialization method. This impression turns out to be accurate upon closer analysis — very little of this action system was tested or even is even testable in isolation. There is overwhelming support

figure 29. This figure illustrates a particular kind of "comment" inside the main action system file of alphaWolf. While comments are in general used to pass explanations between collaborations inside the code itself, the highlighted comments do not *explain* what surrounding code does (as typical comments do), but rather store what the surrounding code *used to be*.

for isolated unit tests in the software engineering of complex systems. Why was it not attempted in this work? Some of reasons are plain to see in the relationship between the added blocks of code and their surroundings — they require too much from their surroundings to be efficiently tested in isolation, they couple so strongly to their environment that to isolate them would require an accurate duplication their environment.

→ The lessons learned will form part of the motivation behind the Diagram system, where we explicitly decouple systems (for example "abstract balances", *page 269*) as well as some of the techniques used in *The Music Creatures* where the coupling between systems is the subject of long-term (that is longer than the execution cycle of a creature) learning, *page 143*. Finally, we design the *Fluid* environment to promote a bottom-up testing methodology by incorporating techniques that allow small components to be assembled into longer sequences — effectively allowing the reuse of the small testing scenarios.

Storage of history — Finally, we note the growing number of comments, notes and markers in the code, the annotation of and the incorporation of the history of the file into the file itself — as notes to collaborators or to self. Is this the correct place for it? And what history is missing — certainly the expected execution orderings that are implicit in the file, but also the to-ing and fro-ing of the numerical parameters in the file as the behavior is pushed one way, perhaps by one collaborator, and another, perhaps by the addition of a new behavior.

→ I develop an environment, Fluid, for creating, monitoring and debugging complex assemblages of code that explicitly offers more historical information about its own use, *page 410*. I also create specific, long term database structures for other aspects of a work and the work's agents, *page 143* and *page 225*.

There will never be another *alphaWolf,* and certainly not simply for the purpose of comparing a programming technique with another. The exciting work of 6 programming collaborators who remained dedicated to one vision of an installation for more than 3 months cannot be duplicated for the purposes of an attempted scientific comparison. So these ideas will be developed throughout this thesis and be tested in other installations, both bigger and smaller than *alphaWolf*, but we shall not be able to return to the exact same project for a conclusive demonstration of their applicability to this particular domain. However, it is worth noting that there was at the time and beyond consensus amongst the collaborators on *alphaWolf* that this work took us to a plateau of complexity that could not be safely crossed in a reasonable amount of time— neither with an extra pair of hands or an extra month.

It was with this thought that I turned to the next 4 years of work — developing the tools and techniques required to traverse this threshold as I took the agent, and the agent toolkit, into new territory.

Loops was commissioned as a "digital portrait" of choreographer Merce Cunningham, and takes as its point of departure a motion-capture recording of Cunningham performing his 1970s solo dance for hands of the same name. It is a piece for screen but has been presented simultaneously with the parallel work Loops Score *which provides a related soundtrack.*

Chapter 3 — *Loops*

Cunningham originally created *Loops* as a solo to be performed in front of a Jasper Johns painting at the Museum of Modern Art.

"Described by Cunningham as an 'Event for soloist,' *Loops* was performed by him at the Museum of Modern Art, New York on 3 December 1971... The piece was performed in front of Jasper Johns's large painting *Map, after Buckminster Fuller's Dymaxian Airocean World*, in the Founders' Room on the museum's sixth floor... *Loops* was performed again at New York's Whitney Museum of American Art on 18 May 1973 (as *Loops and Additions*), and it also gave Cunningham material for his appearances in Event performances, such as the solo in which his hands move through the air around his head and torso, fingers flickering and twitching..."

D.Vaughan, *Merce Cunningham: 50 Years,* Aperture, New York, 1999.

Loops is a portrait of Cunningham — it attends not to his appearance, but to his motion. It is derived from a motion-captured recording of his 1971 solo dance for hands and fingers entitled *Loops*. In this work, his motion-captured joints become nodes in a network that sets them into fluctuating relationships with one another, at times suggesting the hands underlying them, but more often depicting complex cat's-cradle variations. These nodes render themselves in a series of related styles, reminiscent of hand-drawing, but with a different sort of life. Many viewers liken their experience of seeing *Loops* to that of gazing into nature: its flickering motions put them in mind of fire or of primitive biology, perhaps seen under a microscope.

Loops is computed in real time and is, in effect, a live performance (the program is the only "performer" of this choreography other than Cunningham, who has never set the work on any other dancer.) Thus *Loops*, the digital program, confers an odd kind of immortality on *Loops*, the physical dance, for in essence it keeps improvising itself. Manifesting itself through the probabilistic interaction of its distinct parts, it reveals something new with every playback.

1. An overview of the artwork

Art is the imitation of nature in her manner of operation.

This quote from **Thomas Aquinas** seems to be a favorite quote of Cage's (sometimes attributed by Cage to Coomaraswamy).

c.f. **J. Cage,** *Composition as Process, Changes* and *On Robert Rauschenberg* collected in : *Silence,* Wesleyan University Press, 1961.

c.f. **T. Aquinas,** *Summa Theologica.*

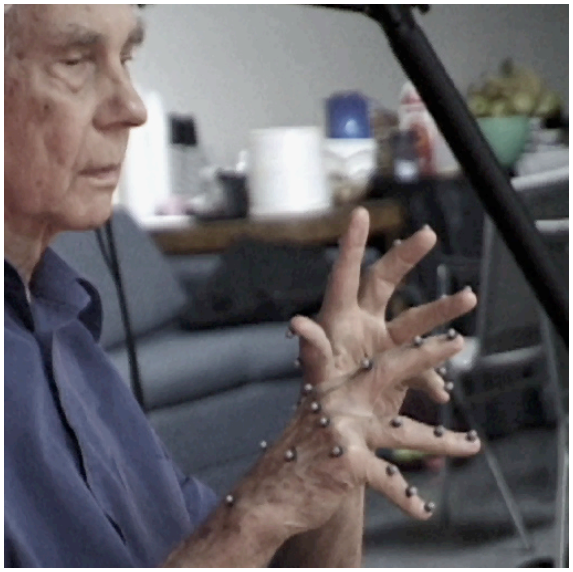


figure 30. Cunningham performing *Loops* for motion-capture session in 1999.

This idea, cited often by collaborators John Cage and Merce Cunningham, led them to a deeper kind of realism, one that mirrored not the world's outward appearance, but rather its underlying processes. One such process, which fascinated both artists, was the workings of chance. They decided that by leaving many of their creative decisions to the roll of the dice, they could give their artworks true autonomy.

Loops explores, questions, and then extends these radical notions of realism and autonomy. *Loops* is a work about distribution and change, about distributing networks of coherence over underlying human motion that are never stable, but the remains of which become the material used to create new networks. There is a minimalism to the monochromatic imagery, but it is accompanied by a sense of finiteness of material — a sense that there is a limit to the number of lines and the number of points available and that the piece, which has no beginning or end, is inefficiently enumerative.

In its particular fashion, the work indicates the first direct point of contact between the history of chance operations in digital art, in which the long time collaborators Cunningham and Cage play a significant role, and the probabilistic action-selection strategies of artificial intelligence. The agent metaphor enters this work through the motion-capture points — some 42 points are represented in the original motion-capture session, and we distribute 42 simple creatures as a "colony" across the hand data. This reflects the desire for an complexity that was entirely interior to the moving hands (this piece is *live* but not *interactive*

with the viewers). The desire to explore, and given the unlimited duration of this piece potentially exhaust, the making and unmaking of relationships between the finite number of points on the hands implies that the creatures' perceptual world should be quite rich — for what the creatures' sense of their peers is the hidden underpinnings of these fluctuating relationships.

Therefore, their perceptual worlds included the movement of the motion capture point it is associated with an individual creature, a number of senses of its local neighborhood of points and signals sent directly from other creatures. These latter two senses were additionally available in forms weighted by the existence of visible connection between the creature being perceived and the creature doing the perceiving. Signaling in the micro-world of the piece takes place at a finite speed, inside a simulated virtual fluid. Thus signals propagate and join in waves throughout the space of the colony and, as they push the behavioral tendencies of the creature around, these signals are also rendered visible on the creatures' bodies.

Indeed the visual appearance of the work stems as much from rendering the perception of these creatures visible as it does from allowing the creatures to chose their appearance, but from the perspective of our agent framework in these creatures' bodies we find an important example of a heterogeneous, non-movement-oriented motor system. Each creature's "body" consists of: a set of ordered lists of points that each start with the creature's own point (for example, these may be drawn as connected line segments), a set of filter coefficients applied to a motion sampler that samples the underlying motion capture data, and a position in a blend space of rendering styles.

Motion capture is the name given to a technology that uses multiple, calibrated cameras to reconstruct the three-dimensional motion of points in space. These points of motion are typically markers attached to human movers. When "cleaned" offline, by hand, motion capture offers a sometimes astonishing accuracy of reconstruction of moving skeletons — the kind of fidelity appreciated by motion-capture's roots in both the biomedical community and military simulation. Today it typically provides the source material for the animations found in computer games, and Hollywood 'digital extras'.

Most recently commercially available real-time motion capture has become a practical reality — and systems made by three hardware manufacturers are now available.

Motion capture hardware, however, remains within the budget scale of Hollywood, the military and computer games. *Loops* used offline-motion capture of a performance by Merce Cunningham that was generously donated by Modern Uprising Studios. The two other dance works use real-time motion capture, with hardware and engineering support donated by MotionAnalysis Corporation.

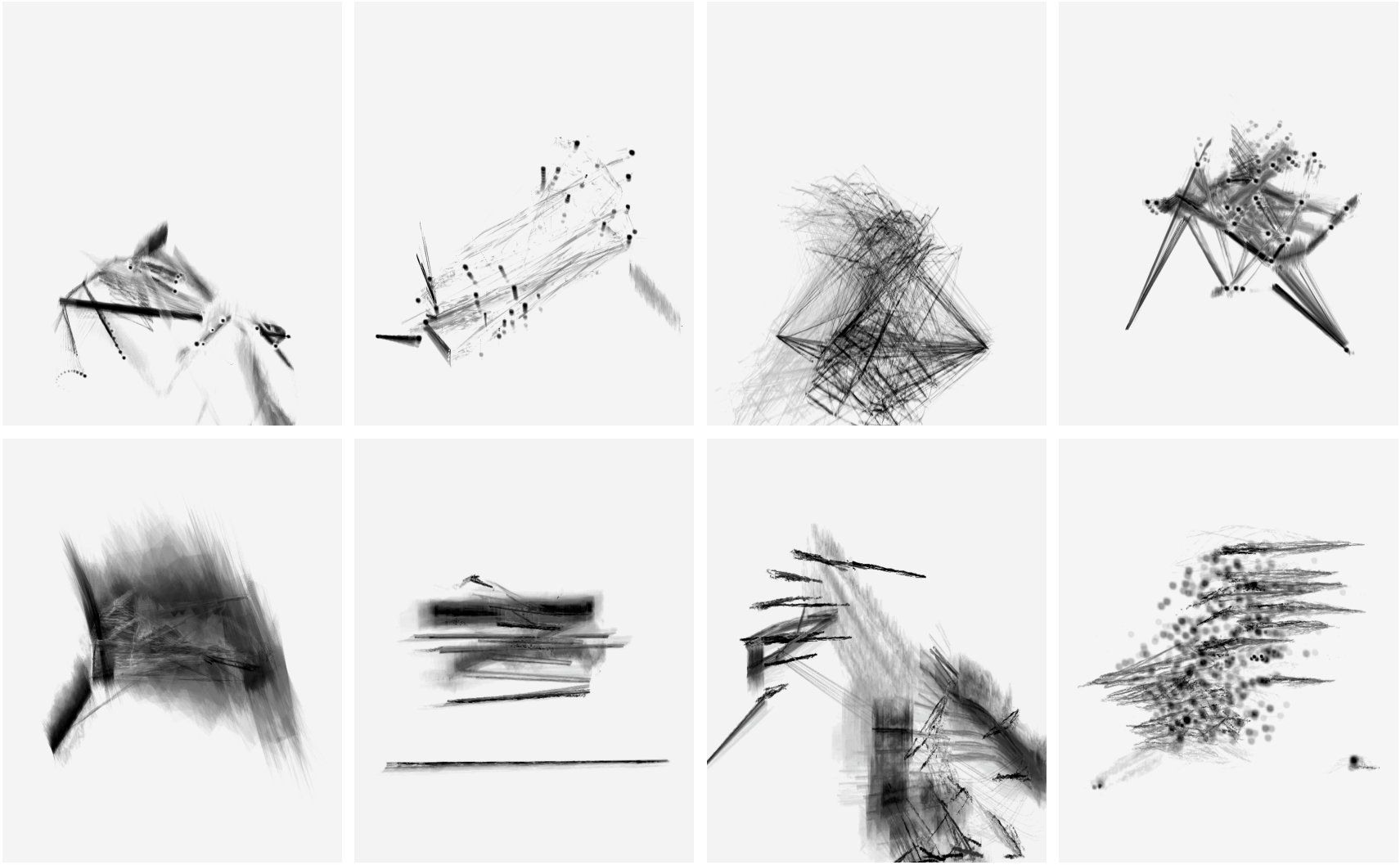


figure 31.
Loops (inverted).

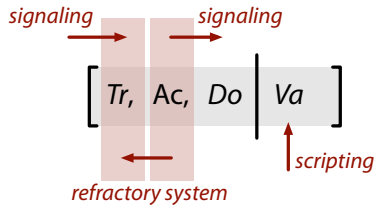


figure 32.
In *Loops* the action-tuple is augmented with two structures, the trigger system and action collaborate in a refractory mechanism and a signaling process. These signals couple the behavior of the agents in Loops to each other.

Since we are concerned with the distribution of similarity and difference, much of the technical and computational resources deployed in *Loops* are concerned with the action systems of our creatures. These systems are constructed using a probabilistic action selection framework very similar to c43, *page 71*. Recall that the two basic construction units of this hierarchical behavior system is the *action-group* and the *action-tuple*.

The contents of each of these parts will be the subject of much description: the triggers for each action-tuple come from a perception of what other creatures in the colony are doing — this is a signaling mechanism global throughout the colony; the do-while is a duration distribution that starts out set by hand; similarly the values of the action tuples are also hand-set initially.

Two extra elements are added to this basic configuration. First is a *refractory multiplier* for the action — actions that fire are less likely to fire again. This dodges many of the coupling / temporal pathology problems discussed in the critique of the c43/c5 action selection strategies, *page 87*, effectively damping away any chance of a one-iteration dither, which otherwise may occur since this action system is coupled through the signaling mechanism to 41 others.

The second element specific to *Loops* is an expectation mechanism that cuts across the action-tuples' duration-controlling do-while — in the event of something "surprising" (to be defined below) occurring, a re-selection is almost certain to occur.

The strength of coupling between each of these two systems and the action-tuples are adaptive: the refractory systems adapt their time constants to be twice the ultimate durations of each individual action-tuple; the expectation mechanisms adapt the size of their effect (equivalently what the expectation mecha-

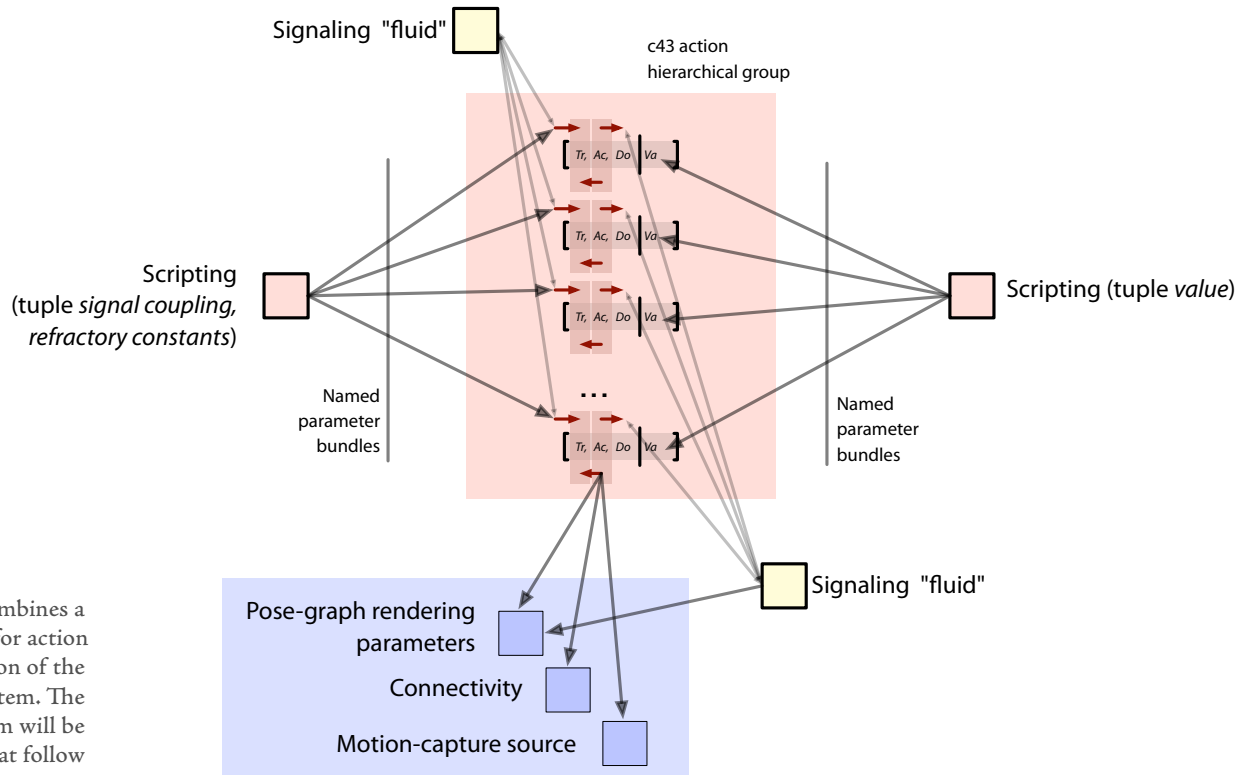


figure 33. Each *Loops* agent combines a new set of scripting techniques for action systems with a new exploitation of the generic pose-graph motor system. The contents of this diagram will be discussed in the pages that follow

nisms consider truly surprising) to try and keep the typical duration of an action-tuple to be near an author-specified duration (10 seconds). These internal adaptive parameters had, of course, the ability to be reset externally — thus these resets had the ability to be *scored*.

The motor systems

Inside each action-tuple are, of course, the actions themselves. In *Loops* these take the form of (possibly a composite of) one of three classes — asking the

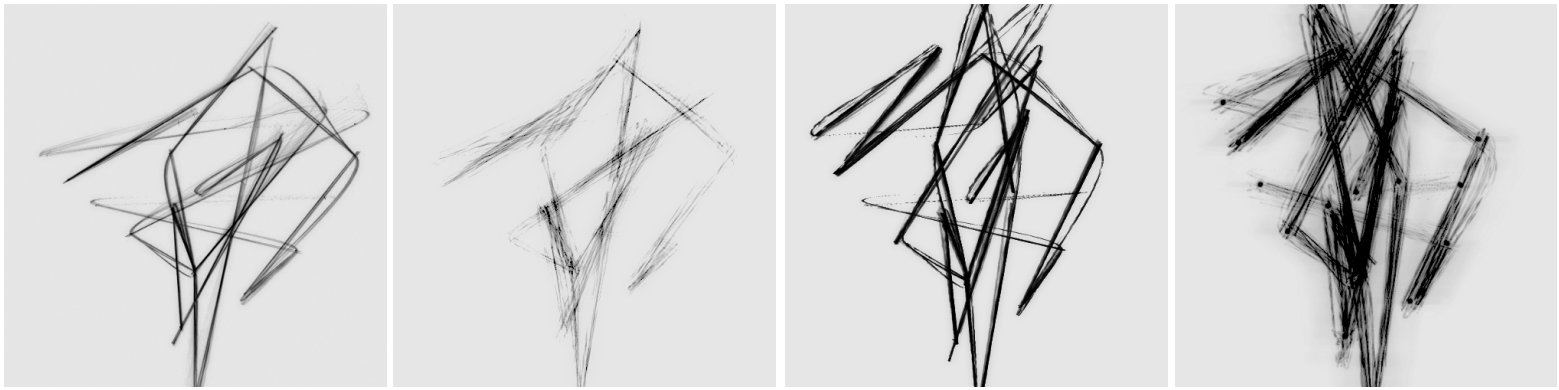


figure 34. A sampling of Loops basis rendering styles or "adjectives".

creature's motor system to change **rendering style**, asking the creature's body to change its **connectivity** or asking the creature to **send signals** out into the colony. We'll look at each of these possible actions in turn.

The creature's primary motor system is constructed purely within the pose-graph framework, but unlike earlier work using these structures, exploits the representation neutrality of the pose-graph not to render the realistic animation of a representational creature using combinations of pre-made material, but rather to control the rendering parameters of a non-representational creature using combinations of pre-made sets of parameters.

Thus we fill in the lower levels of the otherwise abstract pose-graph with the following structure:

The **pose representation** — an *ad hoc* collection of rendering parameters, line styles, noise amplitudes, couplings to signals. These bundles are precursors to other persistent structures in the agent toolkit, the persisted partial trees, *page 225*, and the long-term learning database, *page 143*.

As we have seen, this pose-graph motor system is representation agnostic; we need to plug-in some representations and metrics to handle this specific pose representation. Specifically:

distance metric — the χ^2 distance between the overlapping parameter sets in the nodes (the pose-graph is highly connected, thus this distance metric is less important than in other motor systems). Specifically for all of the parameters N that are present in two poses a and b we have a distance

$$d_{a\to b} = \sum_{i=0}^{i=N} \frac{|a_i - b_i|}{|a_i + b_i|}$$

time metric — each *ad hoc* parameter has a well-known range, and a well-known time-scale for traversing this range, the time metric is given by the average of each of the times given by these ranges.

interpolator — the interpolator for poses is a bundle of interpolators for each individual parameter, each parameter parameter has a "bias" $\beta \in [0, \infty]$ that modifies these linear blends between two values a and b to be $v = b\alpha^{\beta'} + a(1 - \alpha^{\beta'})$ where $\beta' = \beta$ if $|a| > |b|$ otherwise $\beta' = 1/\beta$. This means that the interpolator will err on the side of smaller values of v for $\beta < 1$. This, and the ability to interpose nodes into the pose-graph to deal with the special cases, seemed to give enough control to avoid situations where a linear, independent blend of a great many rendering parameters produced unpleasant intermediate renderings. Finally, *Loops's* motor programs are always interruptible, all blenders are capable of taking the rendering parameters set from any intermediate state.

In addition to causing the rendering style of the creature to move around the pose-graph, actions also choose to modify what the body of the creature is con-

nected to. The body is a set of ordered lists of line segments, with this creature at the head of each list. To keep things simple (and stochastic), the actions which modify the connectivity of the body fall into three different operations:

delete(*segment distribution*) — deletes a line segment drawn from a distribution;

change(*point distribution*, *segment distribution*) — changes a segment to connect to a point

add(*point distribution*, *segment distribution*) — adds a point after a particular segment

An action might perform these operations regularly until some condition is met — for example it might delete(*anything*) until there are no more connections left; or it might add(*any_opposite_hand*, *shortest_list*) until it is connected to 5 other points.

We then construct a vocabulary of distributions. **Point distributions**: *anything* — any point; *any_opposite_hand* — any point on the opposite hand; *corresponding_opposite* — the point corresponding to this on the opposite hand; *correct_hierarchy* — the points above and below in the "correct" hierarchical skeleton of the hand; *across_hand* — the points at the same level on nearby fingers; *connected_amount* — proportional to how many connections a point has; *is_connected_to_me* — only points that are connected to this point; *nearby* — points close by. **Segment distributions**: *longest*, *shortest*, *furthest*, *closest* — each over the length of the segment chains in spatial distance or numerical segment length. Fuzzy binary and unary boolean operations "or" (a+b), "and" (a*b), "not" (1/(1+a)) were also created such that these can be put together.

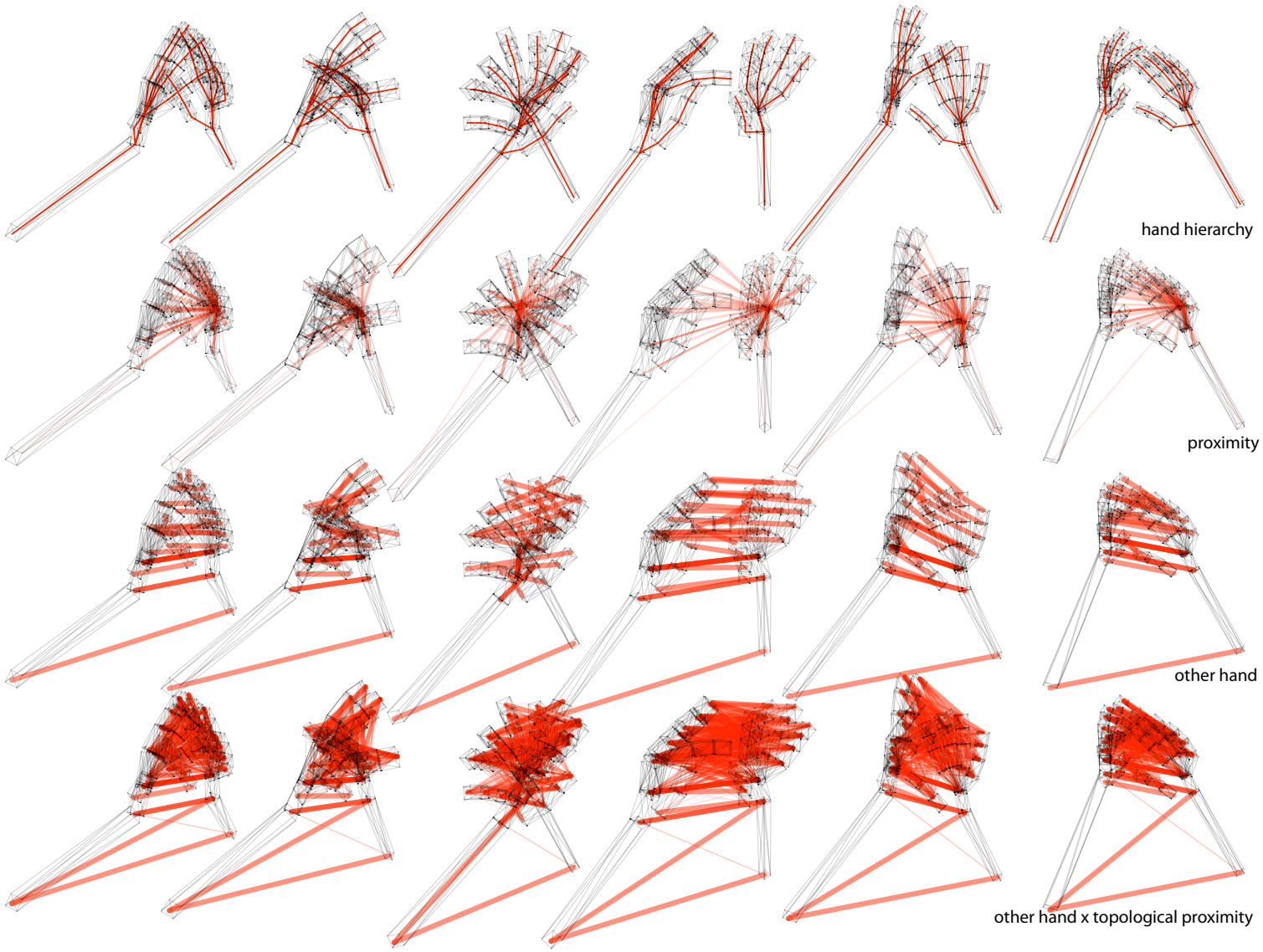


figure 35.
This diagram shows a small sampling of the connectivity metrics distributed across successive frames of motion-capture animation.

Signals and expectations

Signals are named, vector values that propagate away from points inside hidden fluids — one independent fluid for each vector signal. *Loops* chooses to model the fluid by storing a highly down-sampled history of point positions and a sampled history of signals sent by each point.

The signal s present at a particular location p for M signaling markers at positions $p_m(t)$ at time t_0 is approximately:

$$s_p = \sum_m \int s_m(t_0 - |p - p_m(t)|\, v)dt$$

where v is the speed of propagation and $s_m(t)$ is the signal sent by marker m at time t. In practice, of course, the integral is replaced by a sum over the (highly) down-sampled history of each point.

Signal transmission has a refractory nature to it — a signal sent constantly gets used up and fades to a low value, so the fluid model is more efficiently modeled as a set of sources, as above, and as a general background. This sparsifies the sum over markers above.

This weighted average is fed into the action systems of each creature and in the majority of cases, receiving this signal causes the creature to perform a similar action. However, this is not necessarily a constant cause of homogeneity inside the colony — it is easy to author situations using the refractory mechanism that are homogenous but highly unstable, ripe for change. And when this change occurs isolated pools of different behaviors spread throughout the colony and compete for space. It can be some time before homogeneity arises again. These behavior-activation patterns are reminiscent of the percolative behavior of porous solids in physics, or the simulation of "forest-fires" in experimental mathematics, *page 362*.

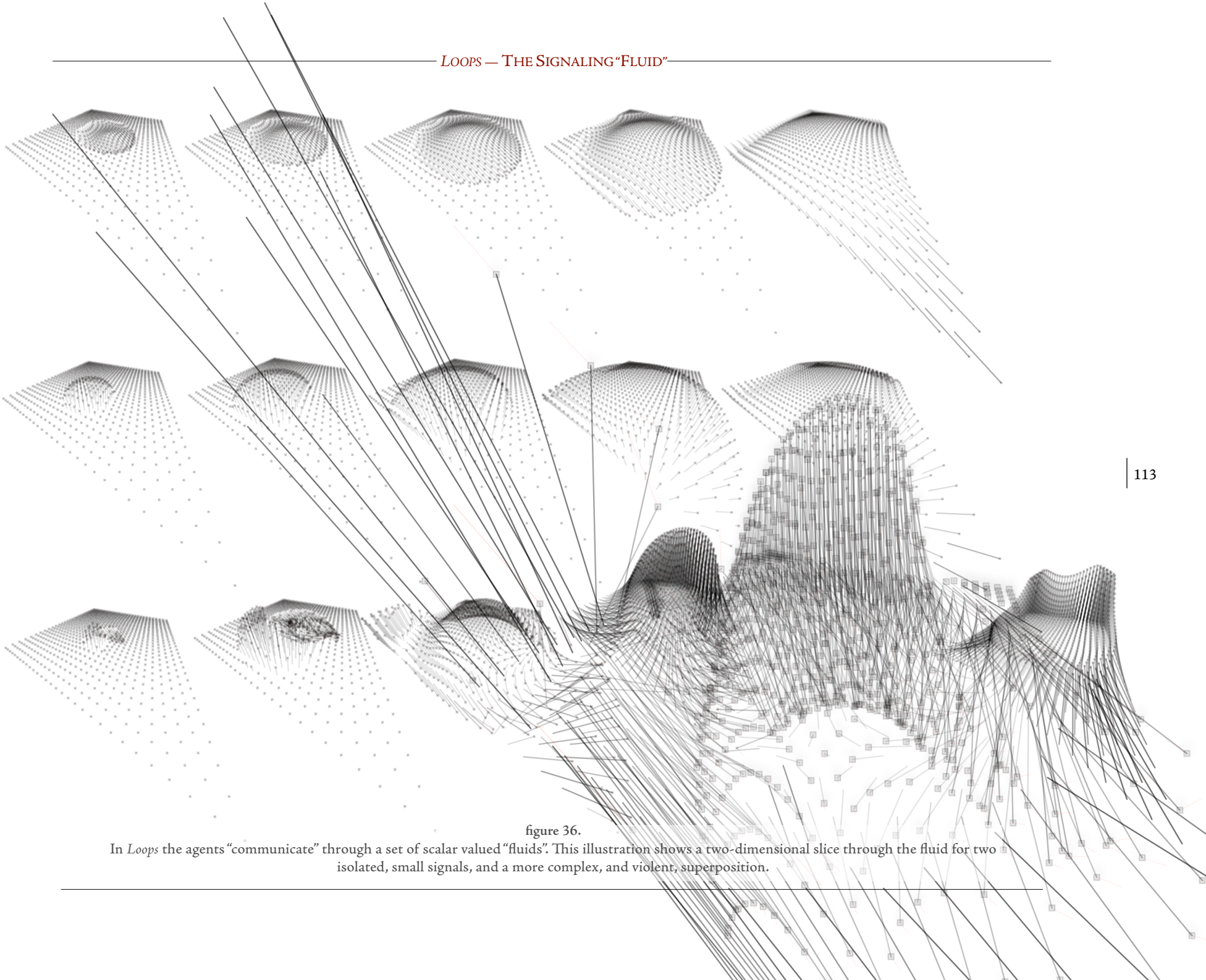


figure 36.
In *Loops* the agents "communicate" through a set of scalar valued "fluids". This illustration shows a two-dimensional slice through the fluid for two isolated, small signals, and a more complex, and violent, superposition.

In addition to forming this weighted average, we can also form a weighted radius:

$$r_p = \max \left| s_p - \sum_m \int s_m(t_0 - |p - p_m(t)|\, v) dt \right|$$

This radius is used as the basis for the *expectation* models of the creatures. Integrating across all of the signals, normalized over a long time period, creatures can observe ongoing, conflicted change nearby in the colony. One of the most important, non-action, signals propagated through the hidden fluid is the acceleration of the points themselves. Therefore, the motion of the hands, and in particular their unexpected motion, seeps into the action selection of the points that perform the motion.

Naming

The final set of behaviors and behavioral parameters that made it into the creatures (some 30 action-tuples) are pushed around by a running script which consists, in essence, of a series of these named states. The act of recalling a *name* either resets an adapted parameter to a particular value, resets a equilibrium point in a drifting parameter, or modulates the value of an action tuple to make it more or less likely to fire.

But what does a name refer to? and should the author of an agent have to know? much ambiguity remains in the single act of naming — does our new label "forest fire (white)" refer to these refractory periods, or the values of those signaling behaviors? In *Loops* the ambiguity is reduced by collecting multiple examples (and remember, for many named states in the colony, we have up to 42 examples for any particular snapshot). By using the consistency between examples to modulate the effect of the saved parameters, when they are recalled, these multiple examples help articulate what it is that we are specifically inter-

ested in, what it is that the artists are in fact naming. Those parameters which show little variety throughout all the examples, upon recall, act forcefully upon the creatures' action systems; those that show no consistency have no force upon reapplication.

K-means is a standard unsupervised clustering algorithm — I enjoy the presentation in **C. M. Bishop**, *Neural networks for pattern recognition*, Claredon Press, Oxford. 1995.

This formulation of the use of the Bayesian information criterion is after **D. Pelleg** and **A. Moore**. *X-means: Extending K-means with efficient estimation of the number of clusters.* In Proceedings of the 17th International Conference on Machine Learning, pages 727–734. Morgan Kaufmann, San Francisco, CA, 2000.

One must be a little careful as to how this "consistency" is calculated if we are to fully exploit the information contained within a potentially heterogeneous set of examples. Rather than using the spread of a value, or its standard deviation we repeatedly cluster using a simple k-means clusterer with k=(1..4). We choose our "spread" to be the maximum of the Bayesian information criterion (BIC):

Given a particular clustering C of the (q-dimensional) points $\{p_i\}$ with $i = 1 \ldots n$

$$\mathrm{BIC}(C|\{p_i\}) = \mathcal{L}(\{p_i\}|C) - \frac{k(q+1)}{2}\log n$$

we take the maximum likelihood estimated, log-likelihood $\mathcal{L}(\{p_i\}|C)$ assuming k spherical clusters with centers μ_i each with n_k points:

$$\sum_{c=1\ldots k}\left[-\frac{n}{2}\log(2\pi) - \frac{n\cdot n_k}{2}\log(\hat{\sigma^2}) - \frac{n-k}{2}\right]$$

with (point p_i belongs to a cluster with center $\mu_{(i)}$):

$$\hat{\sigma^2} = \frac{1}{n-k}\sum_i \left\|p_i - \mu_{(i)}\right\|$$

We can then compute a raw consistency measure just from the "error" of the highest scoring clusterer, here we set this to be $1/\hat{\sigma}^2$.

We then allow each cluster to act separately on the action system weighted by the distance from the current parameter to the cluster. These weights are nor-

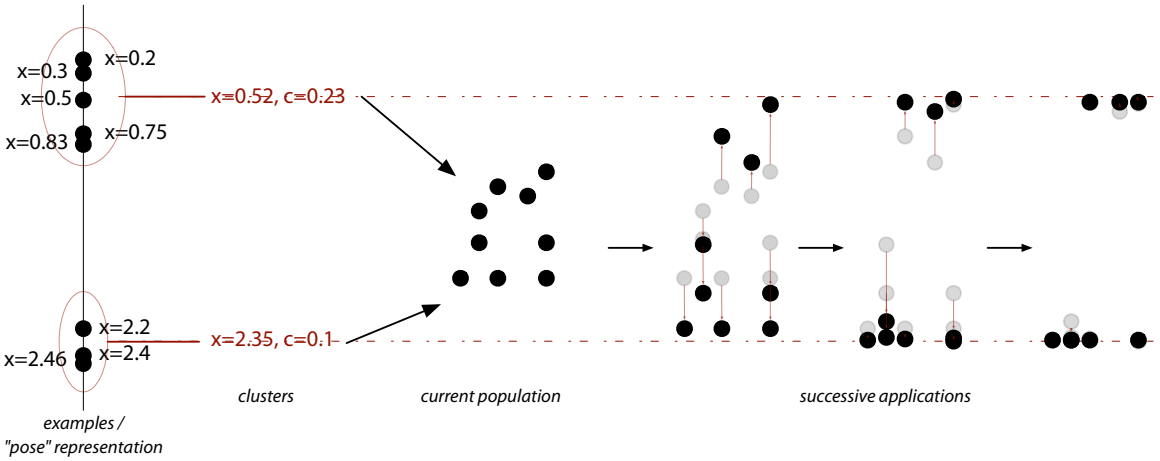


figure 37. Multiple examples inside the pose representation are clustered. During "recall" or pose-interpolation these clusters act on the creature or the colony separately, to reintroduce heterogeneity.

malized over all clusters in this example, but multiplied by our consistency measure (normalized over all clusters for this value in the database):

Given a cluster C_i with mean μ_i acting on a attribute vector v the iteration is:

$$v \leftarrow v + \alpha \frac{\sum_{i=1}^{k} \left\{ (\mu_i - v) \cdot e^{(\mu_i - v)} \right\} \cdot Z / \hat{\sigma}_i^2}{\sum_{i=1}^{k} e^{(\mu_i - v)}}$$

where Z is max $1/\hat{\sigma}^2$ for this value in the database.

There are two desirable results of this scheme: parameter bundles that have, for example, two clearly defined examples for the sample parameter are not penalized as "inconsistent" and thus are weakly recalled if these exemplars are widely separated; secondly, such consistent, but heterogeneous example sets work to increase (and ideally, restore) the heterogeneity of the colony when applied.

These names appear in *Loops* in two places. Firstly they are the bundles of rendering parameters stored in the creatures' pose-graph motor systems. They are recalled by the actions, and smoothly transitioned to by the motor system as described above. Early in the production of *Loops*, these bundles of parameters are created by the hand exploration of the rendering space; later in the production they are sampled and stored from the colony using these techniques. I see this shift from manual exploration to exploration enabled by the processes, if not provoked by the processes, and supported by our agent metaphor, to be a significant hybridization of method in *Loops*. Secondly, names that refer to non-motor parameters — the running actions, the coupling from actions to signals, and the filter dynamics of the refractory system — are assembled by hand, into a looping script which declares at what time what named states act on the colony and with what magnitude (α above). This is created by the artist's loosely distributing change and contrast throughout the 18-minute cycle of time.

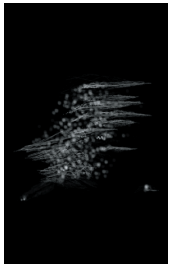
In any case, a rather odd thing has happened here. We can recast this scripting view of all of the creatures' action systems as a basis representation for a "body" for a super-agent motor system. Some of the symptoms that are hallmarks of a motor-system-like solution appear at this level, *85*: we are interested in manipulating the flow of time through an otherwise constrained set of examples (the *sequencing* of animation); there are constraints that are specified in terms of how this flow can occur that cannot be made ahead of time (the *constraints* of *contents*). Although the "script" for *Loops* ultimately consisted of a single chain (in an endless loop) of "actions" driving this action-system-motor-system, this layering of representational style hints at future conceptual possibilities and technical implementations. This is our first inversion and embedding of the perception-action-motor decomposition within itself — there will be others, and when the time comes to revise the agent toolkit, these inversions will be expected, *page 211*.

```
public class LoopsScript
        implements Updateable
{

        Script s;

        public LoopsScript()
        {

                s = new Script();
                s.new LoopingRealTimeBase
                        (mins(10.6), true);

                // cats cradle
                s.new Event( 0.1 ).add(new String[]{
                        "s_pureCatCradle=100",
                        "s_cameraTime=0.6",
                        "s_timeFlow=2",
                        "pointTrans=0.3"
                });


                // a little of "xray"
                s.new Event( 30 ).add(new String[]{
                        "s_xrayContext=5"
                });

                // back away (make the transition tentative)
                s.new Event( 35 ).add(new String[]{
                        "s_xrayContext=0"
                });

                // mixed state — some "xray", some "cat's cradle"
                s.new Event( 40 ).add(new String[]{
                        "s_xrayContext=5",
                        "s_pureCatCradle=5"
                });

                //force a transition into "xray"
                s.new Event( 45 ).add(new String[]{
                        "s_xrayContext=100"
                });

                s.new Event( 46 ).add(new String[]{
                        "s_pureCatCradle=0"
                });

                // thin out density and show points
                s.new Event( 45 + 30 ).add(new String[]{
                        "s_nothing=10000",
                        "s_forwardSampling=0",
                        "s_accPointSize=1",
                        "pointTrans=0.2"
                });

                // scribble (by sampling motion forward in time)
                s.new Event( 45 + 30 + 25 ).add(new String[]{
                        "s_onlyPoints=0",
                        "s_forwardSampling=10",
                        "s_xrayContext=0"   //???
                });

                // complicate matters by introducing some "whiteGia"
                s.new Event( 45 + 30 + 45 ).add(new String[]{
                        "s_whiteGia=20",
                        "s_nothing=1",
                        "s_accPointSize=0",
                        "pointTrans=0.0001"
                });

                // move camera towards hands
                s.new Event( 45 + 30 + 45 + 20 + 20 ).add(new String[]{
                        "s_forwardSampling=10",
                        "s_nothing=0",
                        "s_cameraTime=0.47"
                });

                // transition to "whiteGia" complete
                s.new Event( 45 + 30 + 45 + 20 + 25 ).add(new String[]{
                        "s_whiteGia=100",
                        "s_nothing=0",
                        "s_forwardSampling=2",
                        "s_timeFlow=1"
                });

                // propagate force messages between creatures
                s.new Event( 45 + 30 + 45 + 20 + 25 + 20 ).add(new String[]{
                        "s_doForceFlare=100"
                });
                s.new Event( 45 + 30 + 45 + 20 + 25 + 25 ).add(new String[]{
                        "s_doForceFlare=10"
                });
                s.new Event( 45 + 30 + 45 + 20 + 25 + 35 ).add(new String[]{
                        "s_doForceFlare=0"
                });

                // thin out "whiteGia"
                s.new Event( 45 + 30 + 45 + 20 + 25 + 45 ).add(new String[]{
                        "s_nothing=1",
                        "pointTrans=0.2"
                });
                s.new Event( 45 + 30 + 45 + 20 + 25 + 47 ).add(new String[]{
                        "s_nothing=0"
                });
```

throughout the script there are references to terms such as "**xRay**" or "**amoeba**". These are names that the artists used to talk about the basic stylistic vocabulary built for the piece. They refer to behavioral tendencies, connection topologies and/or rendering styles. These common labels became increasingly important as the piece's stylistic vocabulary developed.

the creatures are responsible for showing how they are connected to other points. Sometimes they choose to connect themselves to points that are make sense in a traditional joint hierarchy. However, they can choose to produce complex 'cat's cradles' or sparse points.

"**whiteGia**" refers to a rendering style that we found reminiscent of the line quality of Giacometti's portraits.



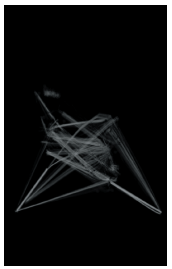
"**forward sampling**" randomly elongates the point creatures bodies along the animation material. The result is often similar to inverted comet trails.

"**force propagation**" refers to the virtual dispersive medium that the point creatures are 'embedded in'. Creatures can inject force into what is in essence a simple cloth simulation, to perturb and fatten the geometry of nearby creatures.

```
// scribble (by sampling motion forward in time)
s.new Event( 45 + 30 + 25 ).add(new String[]{
        "s_onlyPoints=0",
        "s_forwardSampling=10",
        "s_xrayContext=0"   //???
});

// complicate matters by introducing some "whiteGia"
s.new Event( 45 + 30 + 45 ).add(new String[]{
        "s_whiteGia=20",
        "s_nothing=1",
        "s_accPointSize=0",
        "pointTrans=0.0001"
});

// move camera towards hands
s.new Event( 45 + 30 + 45 + 20 + 20 ).add(new String[]{
        "s_forwardSampling=10",
        "s_nothing=0",
        "s_cameraTime=0.47"
});

// transition to "whiteGia" complete
s.new Event( 45 + 30 + 45 + 20 + 25 ).add(new String[]{
        "s_whiteGia=100",
        "s_nothing=0",
        "s_forwardSampling=2",
        "s_timeFlow=1"
});

// propagate force messages between creatures
s.new Event( 45 + 30 + 45 + 20 + 25 + 20 ).add(new String[]{
        "s_doForceFlare=100"
});
s.new Event( 45 + 30 + 45 + 20 + 25 + 25 ).add(new String[]{
        "s_doForceFlare=10"
});
s.new Event( 45 + 30 + 45 + 20 + 25 + 35 ).add(new String[]{
        "s_doForceFlare=0"
});

// thin out "whiteGia"
s.new Event( 45 + 30 + 45 + 20 + 25 + 45 ).add(new String[]{
        "s_nothing=1",
        "pointTrans=0.2"
});
s.new Event( 45 + 30 + 45 + 20 + 25 + 47 ).add(new String[]{
        "s_nothing=0"
});
```

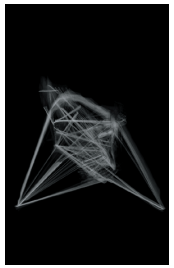
the behavior system paradigm is particularly good at generating hybrid states with some of the colony agreeing on one behavior and others performing some other actions.

```
// transition to a "forest fire" based dynamics
s.new Event( 45 + 30 + 45 + 20 + 25 + 50 ).add(new String[]{
        "s_nothing=100",
        "s_forestFireOne=1",
        "s_cameraTime=0.6",
        "s_randomPointSize=0.0",
        "pointTrans=0"
});
s.new Event( 45 + 30 + 45 + 20 + 25 + 53 ).add(new String[]{
        "s_nothing=0"
});

// a delicate, transient state
s.new Event( 45 + 30 + 45 + 20 + 25 + 60 ).add(new String[]{
        "s_whiteGia=0",
        "s_catCradle=10",
        "s_xrayContext=1",
        "s_randomPointSize=0.0"
});

s.new Event( 45 + 30 + 45 + 20 + 25 + 80 ).add(new String[]{
        "s_cameraTime=0.47"
});

// "message passing" between randomly flashing points
s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 ).add(new String[]{
        "s_message=40",
        "pointTrans=0.4",
        "s_catCradle=0",
        "s_xrayContext=0",
        "s_randomPointSize=0.4",
        "s_cameraTime=0.47",
        "s_timeFlow=0.2"
});
s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 30 ).add(new String[]{
        "s_randomPointSize=0.0",
        "pointTrans=0.4",
        "s_timeFlow=1"
});

// a complex heterogeneous state — Large Glass?
s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 ).add(new String[]{
        "s_tendril=55",
        "pointTrans=0.00001",
        "s_xrayContext=1",
        "s_message=0",
        "s_timeFlow=0.5",
        "s_cameraTime=0.6",
        "s_randomPointSize=0"
});
```



the way in which the point creatures adapt their geometry to indicate how they are 'connected' to other points changes throughout the piece. One of the earliest styles we built was the "tentative tendril" growth style, where points seem to be gently seeking nearby points in the hand.

by changing how gradually or suddenly new behavioral tendencies are introduced into the creatures by the script we can modify the abruptness of the transition. If we quickly force a behavioral tendency to have a very high value we *startle* creatures into revaluating their behaviors. But by gradually introducing new behavior we can create hybrid and "indecisive" states into the colony.



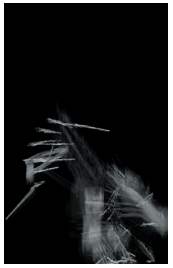
"**nameBug**" refers to a dense, nested connection topology between points that was introduced by a simple mistake in code. Once the mistake was traced and corrected we rebuilt the style.

"**forest fire**" message propagation refers to a complex extension of the "force propagation" used earlier. Instead of passing force into a simple physics simulation, points pass messages of behavioral tendency. This creates a deliberately brittle positive feedback system. Behaviors change between points in a way similar to how fire spreads in a forest. These complex behavioral dynamics were extensively simulated in isolation and could be visualized while the piece was running.

```
// which we slow down for
s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 + 14 + 81 + 20 )
        .add(new String[]{
        "s_timeFlow=0.2"
});
s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 + 14 + 81 + 30 )
        .add(new String[]{
        "s_timeFlow=0.5"
});
s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 + 14 + 81 + 40 )
        .add(new String[]{
        "s_timeFlow=1.5"
});

// startling transition into "name bug" connection topology
s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 + 14 + 91 + 45 )
        .add(new String[]{
        "s_bugName=1000",
        "s_amoeba=0",
        "pointTrans=0.2",
        "pointWhite=1",
});
s.new Event( 45+30+45+ 20 + 25 + 80 + 45 + 45 + 14 + 91 + 45 +10 )
        .add(new String[]{
        "s_timeFlow=0.2"
});
s.new Event(45+30+45+ 20 + 25 + 80 + 45 + 45 + 14 + 91 + 45 + 15 )
        .add(new String[]{
        "s_timeFlow=1"
});

// thin out density
s.new Event(45+30+45+20 + 25 + 80 + 45 + 45 + 14 + 101 + 45 + 20 )
        .add(new String[]{
        "s_nothing=10",
        "s_timeFlow=0.2" // ?
});
// thin out density
s.new Event(45+30+45+20 + 25 + 80 + 45 + 45 + 14 + 101 + 45 + 25 )
        .add(new String[]{
        "s_nothing=0"
});
// complex forest fire with densely connected graph
s.new Event( 45+30+45+20 + 25 + 80 + 45 + 45 + 14 + 101 + 45 + 30 )
        .add(new String[]{
        "s_forestFireOne=10",
        "s_bugName=2",
        "s_tendril=0",
        "s_timeFlow=1",
        "s_cameraTime=0.47"
});
```

```
        s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 + 14 + 101 + 45
    + 35 ).add(new String[]{
                "s_forestFireOne=100",
                "s_forwardSampling=100",
                "s_bugName=0",
                "pointTrans=0.05",
                "s_timeFlow=1",
        });

        // move camera in
        s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 + 14 + 101 + 45 + 35
    + 40 ).add(new String[]{
                "s_cameraTime=0.6"
        });

        // starting transition into tendril growth, with whatever
        // stylistic parameters happen to be there
        s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 + 14 + 101 + 45 + 35
    + 60 ).add(new String[]{
                "s_forestFireOne=0",
                "s_tendril=100"
        });


        s.new Event( 45 + 30 + 45 + 20 + 25 + 80 + 45 + 45 + 14 + 101 + 45 + 35 +
        60 + 30 ).add(new String[]{
                "s_pureCatCradle=100",
                "s_tendril=0"
        });
    }
}
```

we also can change the speed with which we play out the underlying motion capture data. For example here we craft a moment of calm surrounded by frenetic activity.



we do not return at the end of the script to exactly the same place (behaviorally) as we started from. The next and subsequent times through this script (which is looped) the behavior and choices of the creatures will be slightly more complex, slightly less clear cut and creatures respond to residual behavior in the system. Thus there are behavioral timescales much longer than the length of this script.

Rendering Loops

I conclude this overview of the *Loops* installation with a description of the graphical rendering constructed for the bodies of the agents — which at the time offered a unique exploration of what the computer graphics community would refer to as the "non-photoreal".

The material to be rendered is given directly by the point-line segment level description. These line segments are in fact descriptions for splines that originate at the point agent's position but use other agents as control points. It is on these lines that the *Loops* agents' "rendering parameters" act. These lines are interpreted using one form of parametrically controllable spline — the so-called TCB (tension, continuity, bias) spline family. Roughly speaking these parameters control, per control node, the sharpness (T), the "loopiness" (C) and the assymetricallity (B) of the line. The distribution of TCB values along the lines then marks an important class of rendering parameter.

These splines, in turn, are used to deform predefined geometric meshes that span the space of smooth, rough, spiky — the position inside this blend space forms another parameter that controls the appearance of this agent's line.

Prior to reaching the drawing surface, screen-space noise is added to the position of each vertex. The parameters (amplitude and direction) of this noise are not specified directly as a rendering parameter. Rather, their couplings to the signal-propagation layer are specified. This offers a back door for the action selection of points to be visualized as sweeping across the graphical representation of the colony.

These meshes are alpha-composited into the screen as transparent geometry. And the multiple overlappings of the randomly perturbed geometry add considerable texture to the drawn line. However, this texture is entirely controlled by the geometry and, even instantaneously, offers a genuine patina of process to the drawn material.

Finally, the frame-buffer on which *Loops* is drawn exhibits its own, very simple, memory of process. Rather than, as is typical in computer graphics, clearing the screen prior to each new frame, the previous frame in *Loops* is dimmed slightly and the new frame drawn on top of its fading trace. The result is an accumulation of geometry driven textural complexity.

Calculating the spline-based distortions of the blended mesh was sufficiently complicated to occupy much of the processor power present on the high-end commodity hardware available in 2001 when this piece was constructed. Needless to say, Moore's "Law" has turned this aspect of *Loops's* renderer into an altogether more trivial computational load. The mechanism behind the distinctive appearance of *Loops* was reconsidered for each subsequent work and its "hand drawn" aesthetic can be felt in even my most recent work.

Gordon E. Moore's "Law" states that the complexity of an integrated circuit available for a fixed cost doubles every 18 months. For example: http://en.wikipedia.org/wiki/Moores_Law

The longevity of this hand-drawn "look" in my work is not motivated by the desire to display technical virtuosity, nor the delight in a perverse reappropriation of the hardware designed for the photo-real. Rather, it stems from the importance of the sense of effort, the sense of mistake and subsequent correction, and the sense of being trapped within and exploring a finite world of possibilities. Even in the most recent dance piece *how long...* as we move from a viewpoint of agent-as-computing the work to agent-as-seeing the work, this hand-drawn referent remains motivated by that work's *notational* concerns. The effort, the uncertainty and ultimate transience of the hand-drawn, the sketched, remains.

Of course, this geometrically controlled emergence of texture was reinvestigated using more computationally intensive strategies beyond that of simply not clearing the frame-buffer "properly" between successive frames. And my fascination with the gestural connotations of the drawn line motivated a balancing countermove back towards the "photo-real" in the work *22*.

3. Concluding remarks — authorship and emergence

The simplest form of adaptation that can occur in this system is one that modifies the "internal value" of the action tuple. Lower the value, and the probability of it firing, all other things being equal, decreases. This kind of adaptation, provoked by simple, hand-reinforcement, occupies a kind of middle ground — both technically and temporally — in a chain of possible adaptive processes that shaped *Loops* while it was being made. For *Loops* was made as a collaborative work, and in this particular case a collaboration with non-programmer artists. After assembling a certain confidence in our materials, our methodology began with running a small-scale version of the work and tuning it, and growing it. This small-scale version started by using a limited amount of motion-capture material (to help us maintain our bearings) and limited sections of the action system — over time, pieces of action system that had been worked on extensively were pieced together and the system opened up to more motion material.

Initially, most of the tuning took place on the smallest, least "process", and most "direct" level: the appearance of the creatures, and the shape of the blend spaces that their rendering parameter-based bodies traversed; by the end, most of the tuning was devoted to large scale signaling interactions of the colony. At each level (and there were almost always more than one being worked on at a time), there was a cycle of exploration and adaptation succeeded by naming and verification that there was a consensus of reproducibility between the artists and the

colony — that everyone, including the creatures, agreed on the name and what it was that was named.

The list of "adaptive levels" was quite long and reflects, I believe, the depth of collaboration achievable using this stack of adaptation / persistence: **rendering parameters** for the creatures were deliberately altered and new example parameter sets were named and injected back into the blend space available for all creatures; **connectivity tendencies** were assembled and named; **actions** were added into action systems of creatures (sometimes, for experimental purposes, to the creatures associated with one hand) and named; **behavioral decisions** were reinforced (and negatively reinforced), reward signals delivered to the entire colony, to a particular hand, to a set of creatures exhibiting a particular behavior, or, more likely, to a set of creatures exhibiting a particular rendering style; **behavioral configurations** were named, including the preferences created through reinforcement and the internal parameters of the refractory and expectation mechanisms.

Each of these adaptive levels forms an intricate emergent structure; but each pairs a downward specifying *force* with this upwards, emergent, untamed *potential*. Upwards — rendering parameters, although manipulated by hand are constantly being blended together and juxtaposed by the creatures' multiple motor systems; downwards — the sampling, storage and editing of new parameter-sets back into the vocabulary of the colony. Upwards — a basis set of connectivity patterns are created, but here, too, the creatures spend much of their time in intermediate states; downwards — the direct modification of the connectivity metrics. Upwards — the interaction of newly added actions with the existing action system; downwards — the sampling of active actions or the hand creation of partially active sets of actions, or the reinforcement of actions and signaling mechanisms and the annotation of this reinforcement into the script itself.

Thus *Loops* represents in miniature the whole argument of the agent-based practice — it offers a framework for organizing navigational and specificational strategies that mine the potential latent in algorithmic systems. Rather than choosing between a rejoicing in the sheer size of the abstract potential developed by fusion of multi-media and digital process (as offered by artificial life), or the hand-tuned system that acts as a refusal of potential (as offered by practices of mapping), the agent offers an alternative path, where algorithmic, formal ideas are permitted their *potential* while artists are permitted multiple strategies for exerting their *taste*.

Thus the incredible flexibility of the action system, the renderer and the analysis of motion are paired down, even sculpted, interactively by the artists making this work. At each level, points, directions and planes are stored, named and folded back into the work. The first indication that our method was truly "working" occurred when the colony was first exposed to the whole motion-captured performance. *Loops* became a richer work, a surprising work, and yet, simultaneously remained the same work. This stable expansion of a formal idea is often a distant dream of interactive works — far easier is the over-fitting of a piece's parameters to a particular "correct" interaction; far more common is brittle failure in the light of the unexpected. Even today the resulting system is both complex and opaque enough to keep some of its secrets until years have passed — this installation has been touring since 2001, and is booked through until the end of 2005 — and yet was, during its making, controllable enough that this surprises could be captured, assured and incorporated back into the work.

While the specifics of each the levels themselves are concretely tied to *Loops*, this idea of a stack of such levels is not. Indeed, this layering of freely emergent systems with systems that impose not order or control, but explicitly a navigation or the ability to draw a map, offers us a general alternative model to that of artificial life. The agent-metaphor, together with it telescoping structure of time-

scales and self-interactions helps organize how this stack intersects with the artwork's interaction — even if, in this case, the work only interacts with the artists as they are developing it.

We'll note in passing that *Loops*, although it reuses much of the c43 toolkit, occupies a different area of our earlier axial decomposition of action-selection techniques. Viewed from outside, from the perspective of the creators of this work, *Loops* "action selects" on two levels: *Loops's* multiple, interacting creatures allows access to the "multiple concurrent actions" domain previously far away from c43. In the work that follows we'll see further efforts to allow a complex "choreography" of simultaneous actions that moves further away from c43's starting point.

I believe that these layered structures are at the core of why the agent-based offers an organizing alternative to the positions of mapping and emergence, in general, for creating interactive artworks. And while *Loops* was created over a very short period of activity, we shall see this argument only growing stronger as the agent enters either dance theater or long-term collaborative art-making.

Loops then, within its limits, is a work that I claim as successful. Successful in the sense that it leads to something — that is, it does not exhaust the potential developed by its seed technical ideas, but rather leaves one with a better sense of the territory of that potential (for just one example, the point-and-line-based bodies appear, regeneralized in both *Lifelike* and *how long...*); successful in its creation of an authorable yet emergent methodological process; and successful in remaining open to the opportunities of the material that it interacts with. This given, the real question lies in how to expand these "successes" into larger and more complex works. *Loops* had much many attributes in its favor towards these goals. Although collaborative, it was an intensely personal work — it was very much made on our own terms, in our own time — what happens when the

working practice is expanded out to other collaborators (e.g. choreographers) with other time scales (e.g. rehearsals and workshops) and other non-personal, non-constant spaces (e.g. theaters and galleries)? Although complex, *Loops* establishes this complexity by the duplication of simple parts, and therefore risks falling into the anonymity I accused artificial life of cultivating. While the transient presence of the human form prevents the singular from disappearing from *Loops* altogether, it was clear at the time of completion of this work that an engagement with a smaller number of more complex agents was on the horizon — that *Loops* has deferred, but not solved, the software engineering problems apparent in *alphaWolf*. Seen in this light, the "success" of *Loops* promises much but speaks little to these problems. In order to create my next artworks, these issues would have to be addressed.

This chapter introduces the interactive installation artwork The Music Creatures *and some of the technologies it provoked — of most note the* coroutine scripting technique *and the* generic radial-basis channel, *which will be extensively explored and used later. It brings the agent-based to a very particular kind of sound-image relationship, and extends both the authorship and rendering techniques of* Loops, *and the use of the pose-graph based motor system.*

Chapter 4 — *The Music Creatures*

The Music Creatures are a series of multi-screen, multi-speaker installations that directly investigate the use of the creature metaphor in creating interactive music. Over the course of three years, a number of musical creatures have been constructed and revised. The discussion here will focus on the most recent installation, commissioned by the Ars Electronica festival in 2003. that used an ongoing population of four creatures drawn from a population of eight species.

1. An overview of the artwork

Each creature follows the same design principles: a music creature is one that creates sound solely through the movement of its body, and restructures its body to reflect its current understanding of its sonic world; although networked together these creatures will communicate with each other solely through the air — each creature gets a screen, a speaker and a microphone; each creature lives for a short duration (around 7-10 minutes) the life-cycle and in particular the learning-cycle of the creatures will be carefully arranged, by analogy with the sensitive periods found in animals that possess acoustical pattern-learning such

as birds; rather than attempting to capture some complete competence in a particular musical style or cultural context, each creature will have a particular competence in a field that broadly underpins music itself, inspired by the apparently *proto*-musical competences of animals.

This inadequacy of a particular music creature's intelligence to capture a complete musical field or to appropriately capture the complete control surface of its body is a deliberate strategy to create creatures that appear to strive, without long term success, for order and stability. This apparent intentionality, the articulation of effort and the display of the dynamics of expectation and surprise mark the aesthetic goals of this work.

"Bio-musicology" and agent based AI

In this work we are driven by the desire not only to investigate the relationship between musical problems and motor problems but to engage the (necessarily) biological roots of human music and press the field of artificial intelligence into the service of interactive music and digital animation. In this wider context, we have a bold hypothesis: only by beginning with a study of the animal roots of musical behavior — roots which may include the organization of both sound and gesture in time — can we begin to create systems that we can interact with musically. We further hypothesize that traditional, western "high art" music theory might have no place at all in the construction of primitive, artificial, interactive musical agency, and that a study of proto-musical capabilities and the commonalities of animals may ultimately prove more useful for the creation of new interactive musics. Therefore, we make perceptive, learning and motor representations of these music creatures biologically plausible, in the hope that they will be useful for communicative and expressive ends.

For example, the compendium: **N. L. Wallin, M. Björn,** and **S. Brown,** *The Origins of Music.* The MIT Press, Cambridge, MA. 1999.

This often comparative work connects nicely with the ongoing work on the cognitive and neurobiological origins of music. For example, **I. Peretz,** *Brain specialization for music : New evidence from congenital amusia.* In: Biological foundations of music. Annals of the New York Academy of Sciences, 930, pp. 153-165, 2001.

And the tentative relationship between proto-music and proto-language: **R. Jackendoff,** *A comparison of rhythmic structures in music and language*. In **P.Kiparsky** and **G. Youmans** (eds.) *Phonetics and Phonology, Vol. 1. Rhythm and Meter* (pp. 15-44), Academic Press, New York . 1990.

S. E. Trehub, E. G. Schellenberg, D. S. Hill, *Music perception and cognition: A developmental perspective*. In: I. Deliège, and J. A. Sloboda, *Music perception and cognition.* Psychology Press, Sussex, UK. 1997

Despite a recent resurgence of interest in such biomusicology, science cannot yet provide a computationally constructive or artistically useful theory of musical production, consumption or collaboration. And it is unlikely that pure biological approaches will bear fruit without complimentary constructive artistic experimentation.

These musical creatures are early moves toward these goals — they are posed as a response to the problem and draw their inspiration from the problem, rather than as offering any solution to the problem. While their scientific contribution, in terms of either their explanatory or predictive scope is limited, they do however represent what I believe to be the first artistic contribution to the field of biomusicology — a field that, unlike, for example, artificial life, has so far failed to capture much attention either within the interactive art community or within the computer music community. If biomusicology is to live up to its goals — to revolutionize our understanding of music's relationship with the mind and the human's relationship with music — then digital artists will need to figure out how to play a part.

With regard to animals and music, *The Music Creatures* are a set of agents that:

> possess a variety of prototype implementations of **acoustic templates**, in the sense that they segment and understand their acoustic environment (shared with humans) in order to create sound in it. The goal is to create a set of acoustic processes that are capable of simultaneously generating novel material, and of being crafted in ways that the artist considers perceptually intelligible. I hypothesize that to find algorithms and representations which, when placed in the real world, offer the artist a creative balance between surprising novelty and meaningful control, we should look to the algorithms and representations found in nature.

possess abstracted, **simple bodies**, the control of which they learn, and the movement of which create sound. Success will be achieved when there is a certain unity of form: between visual depiction of the creature, the sound that it creates, and its expressive and communicative needs for doing so. This unity is seldom achieved in the plentiful multimedia artworks created to date. I hypothesize that animals provide an ideal example of understandable and engaging expressive agency.

develop on long, environmental, time-scales. They will be works with rather classical form — their narratives will have "beginnings, middles and ends"— rather than a temporal heterogeneity that I believe has become the norm in generative art. Such life-spans of creatures are not without precedent within the agent-based artifact, but the inclusion of biologically motivated and genuine development in such works perhaps is. I hypothesize that by looking at such natural "narratives of development" we can understand how to make interesting autonomous art that unfolds over long periods of time.

We cannot survey all the art that has been, or could be, made with a concern for these characteristics, nor can I cover the variety of possible animal analogues. *The Music Creatures* is simply one extended elaboration on these principles.

Bird song — ontogeny

But before I begin to describe the unfolding of this work, there is much to be gained from discussing a particular natural analogue that was influential in its developments — song birds. *The Music Creatures* are not birds, and unlike, for example *Dobie*, we do not look directly to an animal analogue to find the specific behavior, body and interaction for our creatures; we draw, instead inspiration from the challenges that songbirds face, and the ways in which they face them, to provide a level of "structuring complexity" for the artwork.

This view of the ontogeny of bird song is derived from the long and fascinating literature on the interaction between the innate and the learnt in songbirds. Sources that have been inspirational include:

towards constructible models of song learning — **P. Marler**, *Three models of song learning: evidence from behavior*, J. Neurobiology, 33:501-516. 1997.

an early overview of the problem — **W. H. Thorpe**, *Bird-song; the biology of vocal communication and expression in birds*. Cambridge University Press, Cambridge, UK. 1961.

a later overview **P. Marler**, Song-learning behavior. The interface with neuroethology. Animal Behavior Vol. 30 pp. 479-82, 1991.

and on the connection with human music — **P. Marler**, *Origins of music and speech: insights from animals.* in: **N. L. Wallin, M. Björn,** and **S. Brown**, *The Origins of Music.* The MIT Press, Cambridge, MA. 1999.

Many researchers in classic experiments on oscines have produced a number of well established and fascinating key results, including:

memory based learning. There is a sensitive period for song learning, where infant birds must hear fully developed songs of their species in order to develop normal song. Acoustic isolation, or exposure solely to the song of other species typically results in abnormal song in adult life. However, birds do not sing until much later than this period (the waiting period in white-crowned sparrows, for example, is around 100 days). This sensory -acquisition phase is generally completely distinct from the motor-production phase. This fact is an unavoidable obstacle to any simple motor theory of song production. *The Music Creatures* make no sound, make little use of their bodies and do not progress normally along their developmental narratives in the absence of sound in the gallery.

subsong. As the motor-production phase begins, birds "babble" — a phenomenon not unlike the one we observe in human infants. Amorphous, this babbling doesn't have much connection with what was heard previously. However, birds to seem to require self-feedback at this stage — deafening birds during this period prevents the ultimate production of normal song. It seems possible to conclude that during this phase the bird is constructing sensorimotor mappings for its complex sound production systems. Temporarily deprived of a connection to their motor systems early in their development, the agents presented here would not develop stable musical patterns in later life.

plastic song / overproduction. During the next stage, the song possesses many syllabic elements; some are those heard during the tutoring time and some are "improvisations on a theme". Most will ultimately be discarded as the song crystallizes. Analyses of the songs produced in this phase, however, imply that more acoustic material has been memorized by the birds

that than is evidenced by the structure of ultimate adult song. *The Music Creatures* here, through their actions, organize their musical material while playing it, crystallizing a particular pattern or a particular set of sounds.

innate acoustic templates. But in this memorization process birds do not act as acoustic sponges, nor do they soak up all their early sonic environment while they are young. Instead, birds tend to learn only from the songs of conspecifics. Careful experimentation reveals that perceptual and motor limitations are not enough to explain the lack of openness in the learnt song, rather, a certain quantity of innate knowledge about the ultimate song form is required. *The Music Creatures* here sample only parts of the musical domain that they are equipped to understand — one agent looks for rhythmic material, another cares only for timbre. Their predispositions to parts of human music shape what it is that they tend to learn, and what it is that they *can* learn.

While *The Music Creatures* are the basis for no scientific claims, they are clearly not unrelated to natural processes, and they do, I believe, represent one of the first synthetic and artistic deployments of agents that possess some form of biologically inspired development.

2. Narrative descriptions of *exchange, network, line* and *tile*

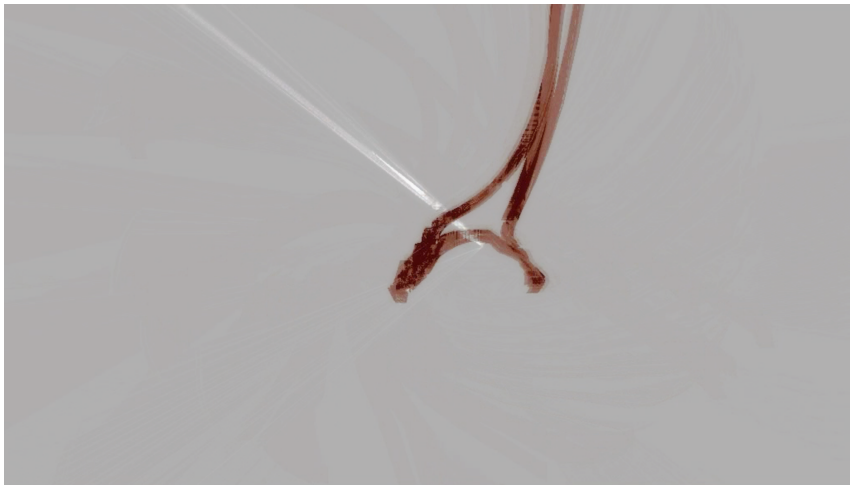
Four creatures from the 2003 version of this work are sufficient to give a sense of the conceptual span of the creatures and the deployment of the technical systems described in this thesis (the remaining four are variations of this set). These creatures are: **exchange** — that constructs and then learns to play a spatial percussion instrument of sorts, while constructing and learning how to move its body; **tile** — a body-visualization of a rhythm analysis that understands and generates rhythmic patterns; **network** — a body-visualization of a

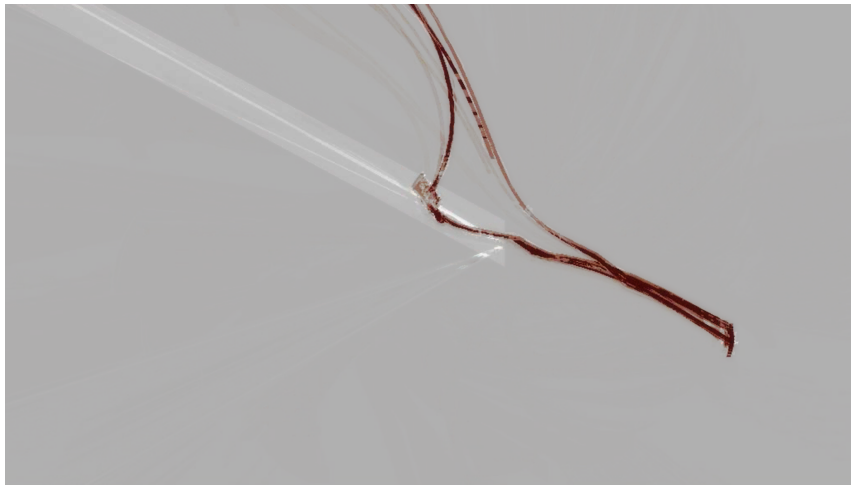
simple transition network model of acoustic material that tries using its body to accompany the sounds that it hears; **line** — a metaphorical tape loop that records into an expanding pose-graph motor representation the sounds that it hears and sings.

Each offers a response to the life-cycle of a song bird: they marry both the memory-based aspects of "song learning" with an innate specification for the end of the learning period — and this connection is further underlined in installation where the recording of sonic material is displayed quite visually and sound leaves traces on the screen. Their approach to formulating a sound-image relationship is grounded solely in their bodies — there is no sound without movement and any sonic learning that occurs takes place close to, and is exploited by, their motor-system structures.

Of course, unlike birds their virtual bodies are not necessarily constrained in construction nor in appearance by the physics of the real world. However, *exchange*, *line* and *network* all exist and fight with small "physics" simulations. This underscores in installation the physicality of the sound production that we are undertaking. Neither does a single creature stabilize on a single, robust song for a long period of time — these creatures' lives are evenly distributed between periods of open learning and periods of more closed use. However, this is an approach to the "composition" of music that is analogous to the the indirection of birds' learnt song. Rather than specifying the music as notes on a page, the mode of the music's becoming is predefined in the authorship of the creatures. As the work proceeds, the life-cycles of the creatures on the multiple screens diverge, thwarting any direct compositional tendency that remains.

The language used in the following outlines sacrifices detail for the sake of a brief overview. Later sections will fill in these descriptions considerably, clarifying exactly what these creatures do and, perhaps more importantly for this document, how they were made.





exchange

The control parameters for the surface that makes up this creature's body are four rotational degrees of freedom. The surface itself is "skinned" using the conventional skinning algorithm — see page 352.

The exchange creature's body is a simple parametric planar surface embedded in a physics simulation that preserves linear and angular momentum. By changing the control parameters this surface, kinetic energy is injected into the physical system, propelling the creature around its rather small world. Three pre-made "animations" of these parameters, constructed by hand, are played out by the creature, randomly at first as it learns the mapping from surface parameters to the resulting physical movement. This damped, simple physical world offers the opportunity to construct motor-learning in miniature.

In accordance with an emerging general principle in this work, the accumulation of acoustic events are both marked in space and by the indicated growth of the creature's body. This surface, although distorted, is isomorphic to a plane, we place material spatially and graphically (represented as vertical lines) at the most appendage like position, here the fastest moving edge vertex, and growth of new vertices occur around the fastest moving edge. Thus even eventual form of the

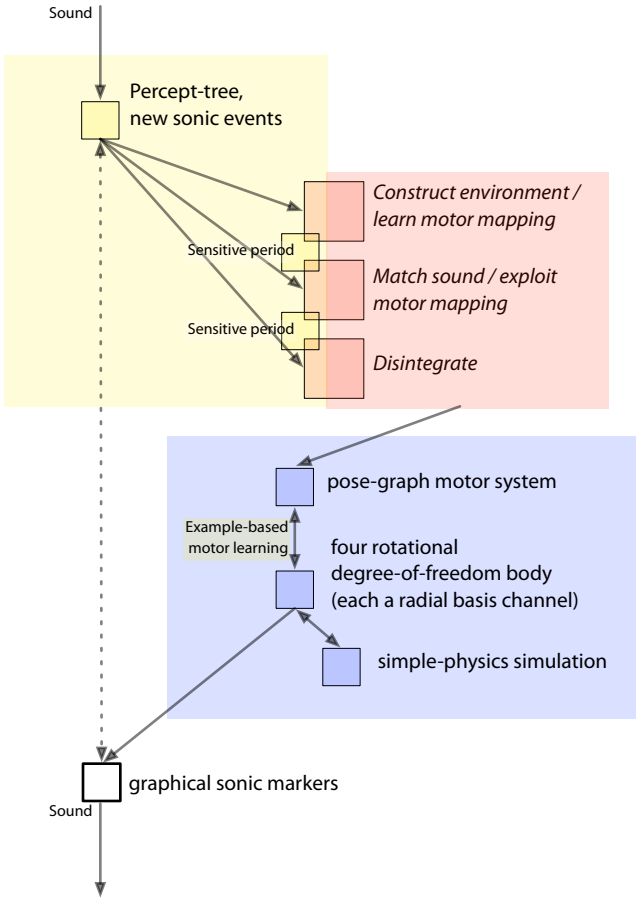
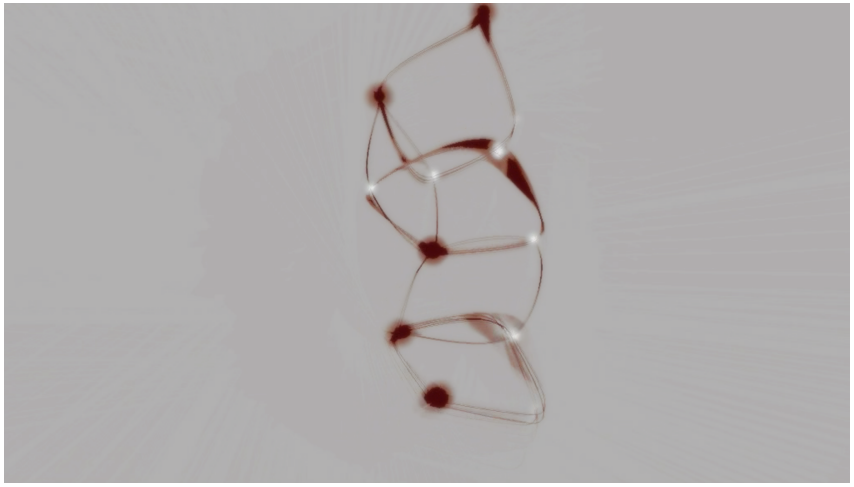


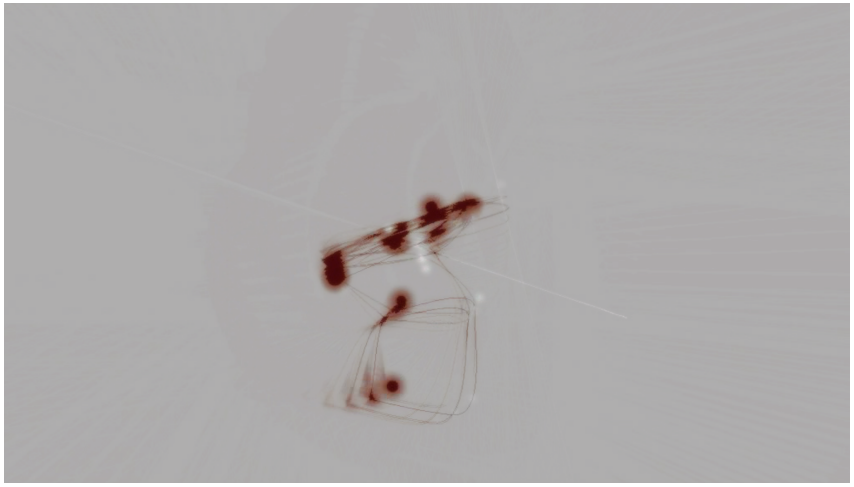
figure 38. An overview of the *exchange* agent.

body is deferred to the agency of the creature and its interactions with the environment..

New acoustic lines distort the locations of previous lines, evening them out. Once the creature has accumulated enough body material, enough evenly spaced acoustic material, and a complete enough map of its body control surface, this learning period ends. Having satisfied this 'motor-acoustic template' the creature now begins to propel itself around the world a little more willfully and a little more flexibly using a more connected version of its pose-graph motor system. Its action system here is still quite simple (and will remain so throughout the piece) moving between seeking notes that are related acoustically with events that it hears, to seeking notes that are as different as possible. The amount of energy that it devotes to moving is coupled to a reservoir filled up with the amount of activity in the room and damped by a longer term increasing fatigue.

Eventually the notes are used up, and as they disappear the body of the creature disintegrates.





tile

There is much precedent for rhythm tracking using phase and frequency locking. In a musical context , the entraining oscillator model is close to the model used in this agent, see: J. D. McAuley, *On the Perception of Time as Phase: Toward an Adaptive-Oscillator Model of Rhythm*. Ph.D. thesis, Indiana University, 1995.

E. W. Weisstein. *Net*. From *MathWorld— A Wolfram Web Resource*. http://mathworld.wolfram.com/Net.html

The tile creature's body is made up of a variable number of flat tiles, and each tile is coupled to a single ongoing model in a rhythm tracking perception system. Each model represents a phase and frequency hypothesis for the acoustic pulses heard in the world. The tiles are connected in a hierarchical structure; initially this structure reflects the parent-child relationship between hypotheses in this percept tree, and free edges for growth are chosen at random. Initially the creature is content to grow its body on a relatively flat surface, appearing like the flattened *planar net* of a complex polyhedron with small oscillations of the tiles governed by the frequency of of the corresponding rhythmic hypothesis.

With enough confidence in the shape of the structure, the amplitude of these

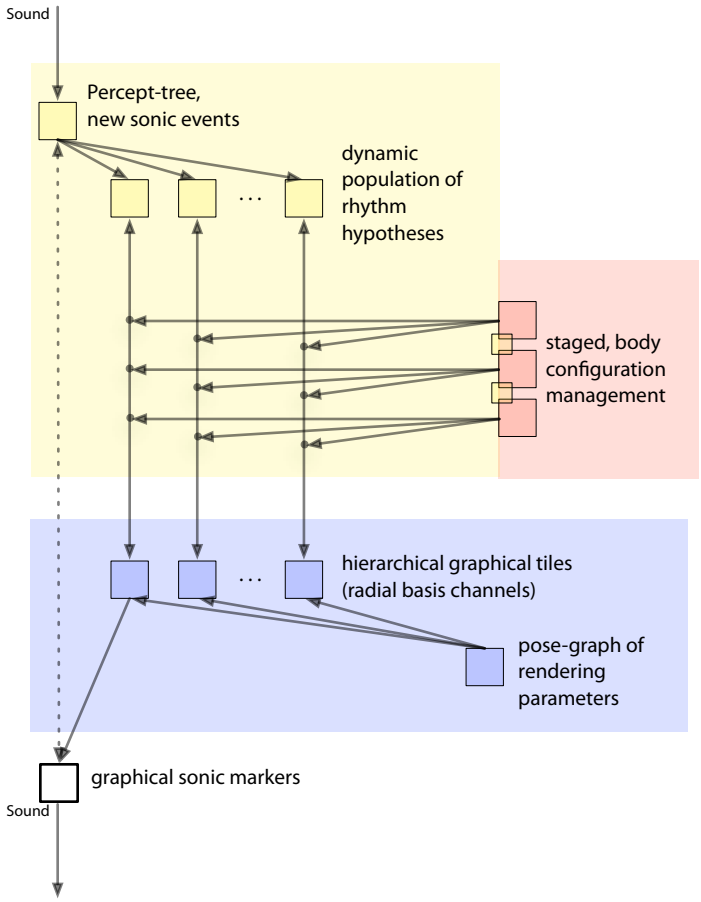
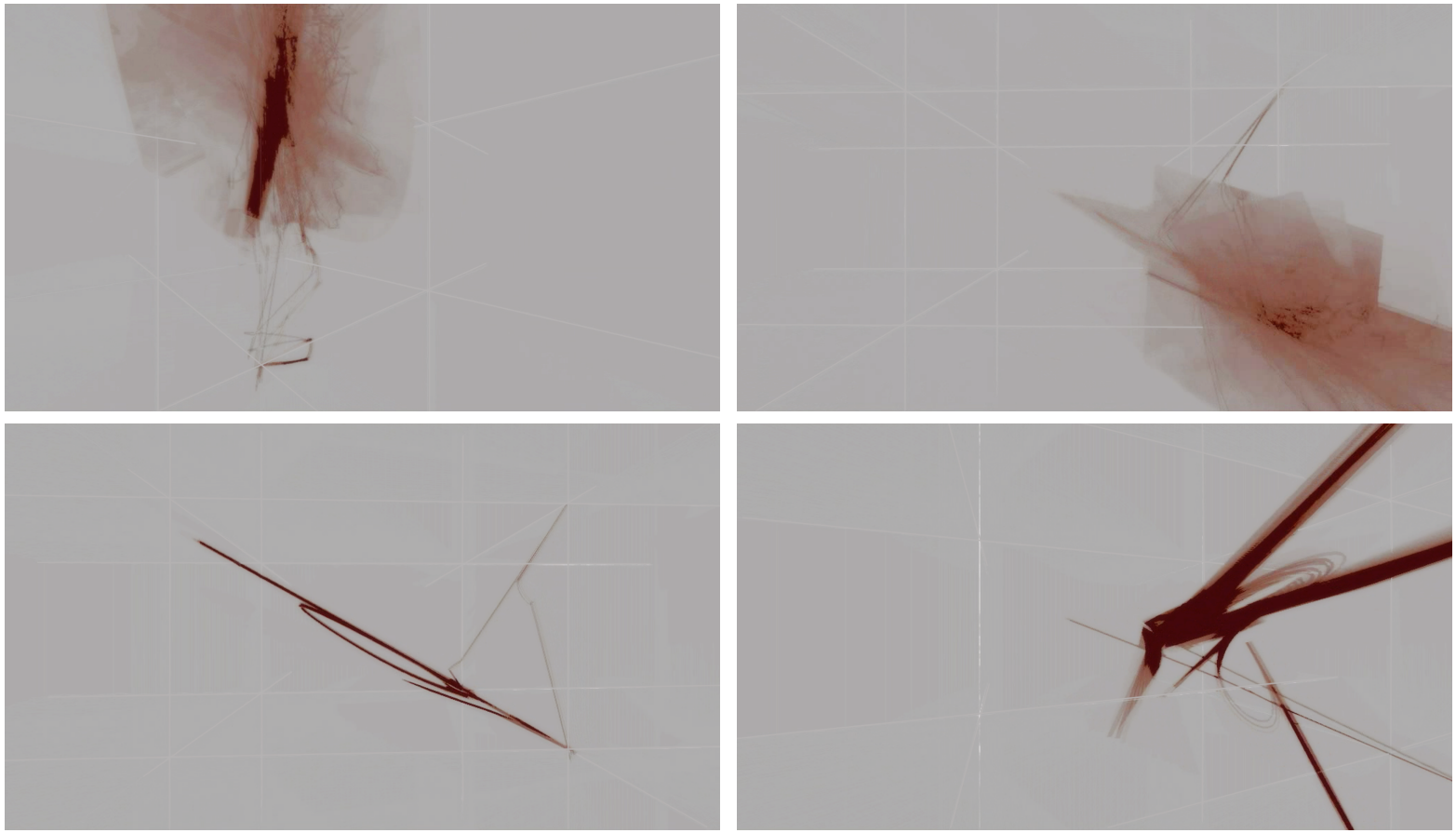


figure 39. An overview of the *tile* agent.

oscillations increases until they become complete rotations. Material that is heard is placed at the location of the tile corresponding to the model that best explains this particular event, and is struck in subsequent rotations. The resulting patterns are strongly poly-rhythmic. Due to the complex folding of this nesting, rotating structure, other tiles can strike these lines — these secondary events are heard as a faint, arrhythmic echo.

The tiles need to be continually reorganized either in terms of confidence (higher confidence tiles migrate to the root of the structure) or, later in the life-cycle of the creatures, frequency (lower frequency models migrate to the root of the structure). During this reorganization it also moves from the complex, randomly branching 3-dimensional structures to what is in essence a single two-dimensional curve made out of squares. After this structure is achieved, subsequent reorganizations drop rather than transfer tiles. The body evaporates.



network

The body of *network* is a point-line connected graph constructed by building a vertex for each identifiable sound event in its environment and an edge between subsequent sound events to indicate temporal succession. It is thus a literal visualization of a transition graph model for musical structure. Duplicate edges are slowly removed and lengths slowly adapted to relax tension in this potentially over-constrained graph structure.

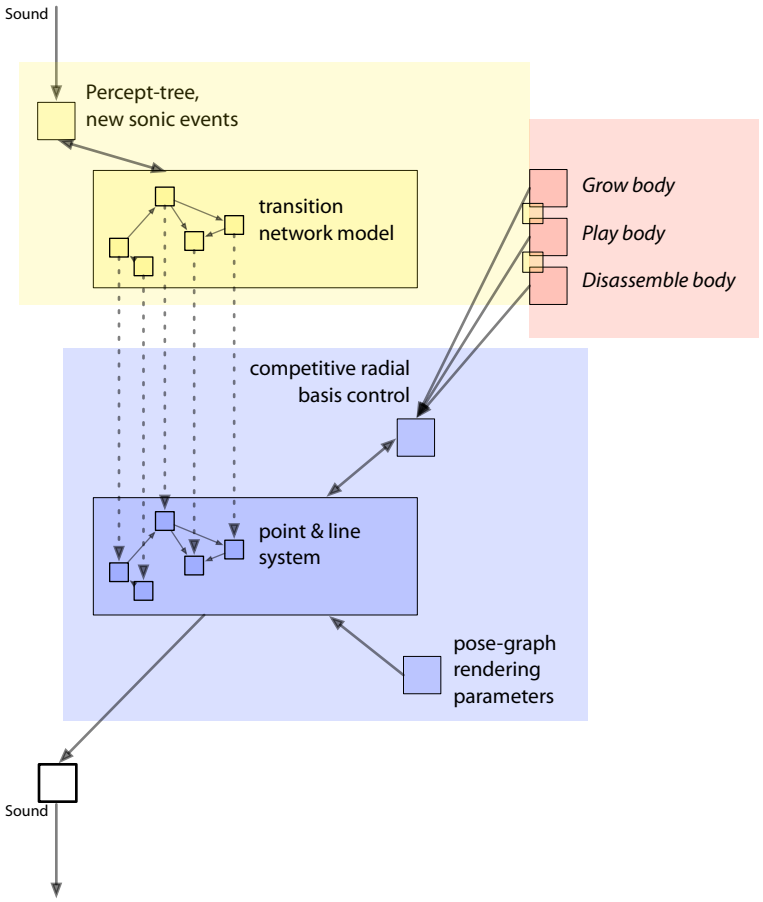
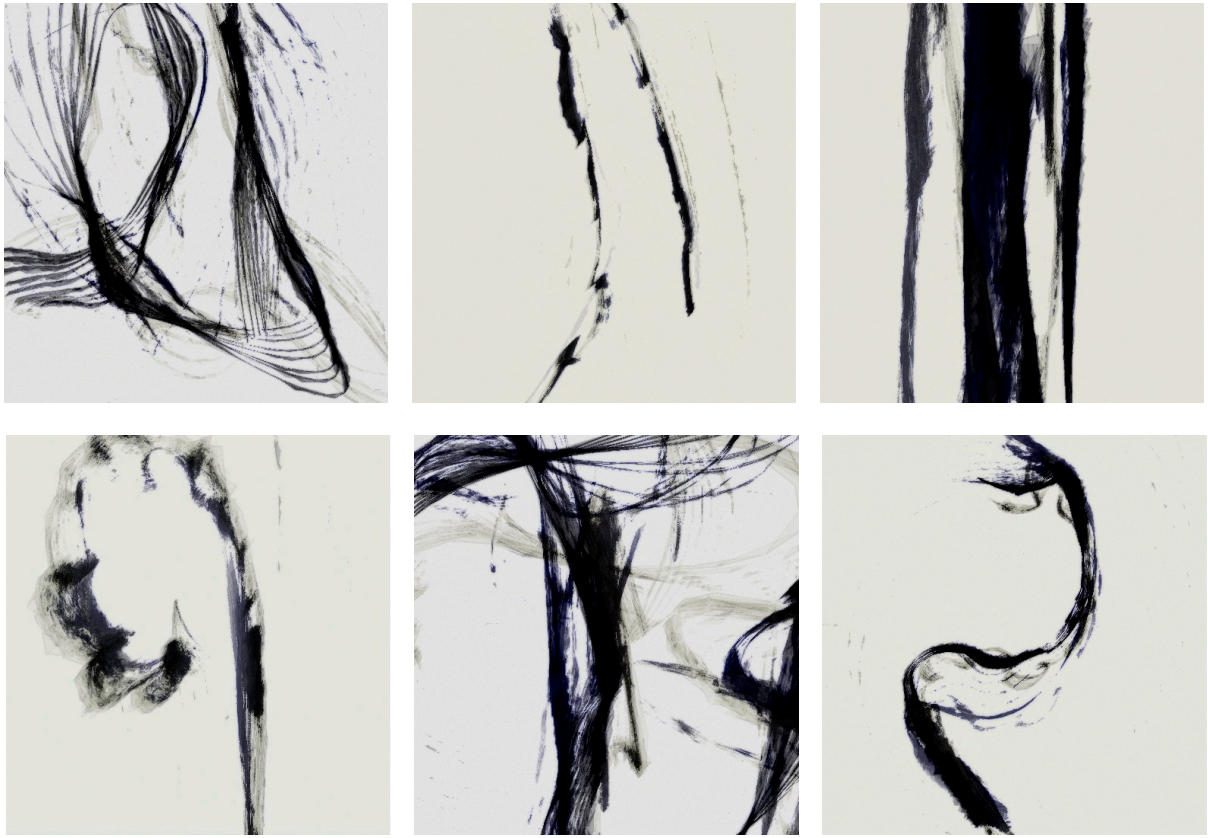


figure 40. An overview of the *network* agent.

After we have a well-ordered graph of sufficient complexity, and after there is a level of confidence in the consistency and relevance of each of the acoustic models, the creature can attempt to respond to acoustic events, by playing the node(s) that are likely to come next given a particular stimulus, after the expected delays. Essentially, *network* "joins in" with musical fragments that it recognizes.

However at any moment a number of hypotheses about the next sound fight for the control of the diagram of possible music that is the creature's body. However, there is a catch, a physical constraint: in order to "pluck" a node, the creature must have physical support from an underlying lattice structure — the nodes either side of a node must be bound lattice before a node can be plucked.

Unheard nodes are eventually forgotten, culled, and replaced by new material. Eventually the culling continues, but this sense of "newness" is not refreshed and the network disintegrates.



line

A single, simple, looped chain attempts to cycle through three hand-drawn shapes — a triangle, a square and a spiral. However, its body representation is an under-specified node-local representation rather than global so the process of transition proceeds through search rather than as a direct morph or blend. Each node acts through impulsive forces propelled by the sonic events in the room rather than through a direct control structure and thus liable to overshoot, escaping local minima for the search. Faint traces of globally consistent solutions to the problem are indicated through nodes that are getting close to a solution.

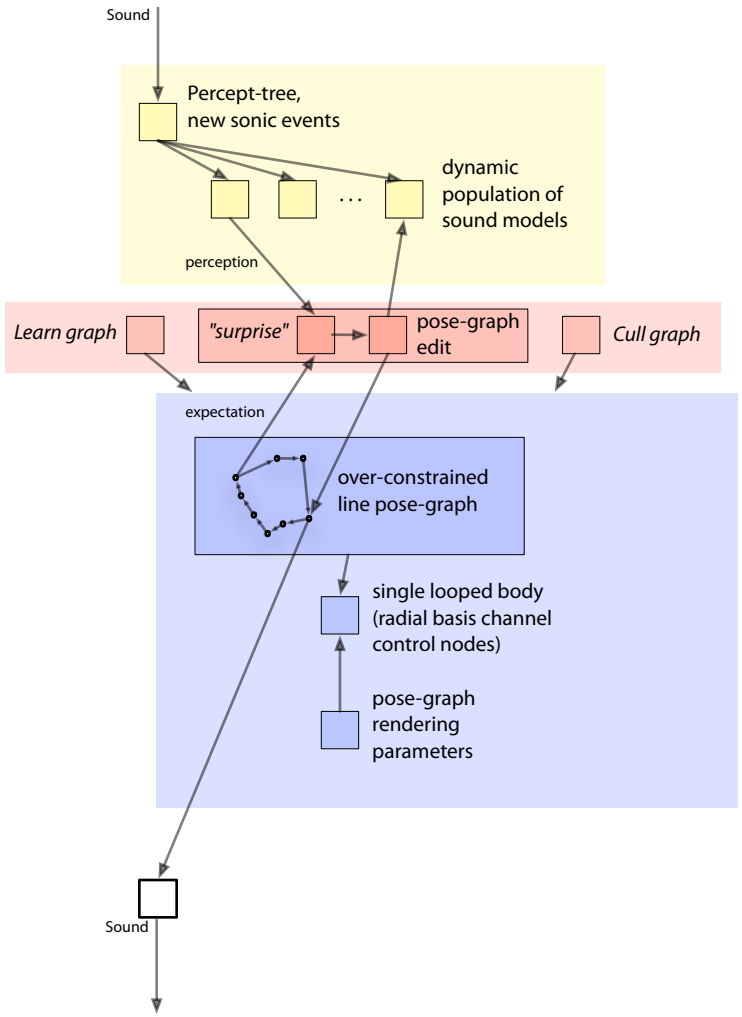


figure 41. An overview of the *line* agent.

The three-state motor sequence is represented in the language of the pose-graph motor system, as a looped graph structure. As it attempts to go through the sequence of states it records the sounds that it hears in those nodes using the pose-graph as a secondary memory structure.

Having successfully toured its graph and learnt the sound of each pose, it begins to sing the contents of the pose as it traverses them, slowly drifting from node to node, but still being propelled by the presence of sound. However, important sonic events that are unlike the contents of the nearest pose-graph node are "unexpected" and cause the insertion of new nodes into the graph, and thus a new pairing: the current state of the body (which has, likely, been thrown off course by the force of the sonic event) and the sonic material that caused it.

Nodes, especially nodes that have not been successfully visited in a long while are slowly dropped from the graph until only two remain.

Eight creatures, the four above and variant of each were assembled and tuned in two months, including the time taken to construct the new non-photorealistic renderer that drew their bodies. Despite this short time-frame, *The Music Creatures* departs from *alphaWolf* and *Dobie* in refusing to structure the interaction between participant and agent with an overt narrative or a specially constructed interface. Nor does *The Music Creatures* develop a way of simplifying the world of the agent: rather the creatures are simply given a microphone each.

In this formulation of the agent and world, much is unforeseen. Thus The Music Creatures were a challenge — how open to the potential of the sounds in an unknown gallery, their interactions with the agents, and the interactions between the agents could an artwork be?

This time-frame was made possible by the re-use of many of the techniques described in this thesis, in particular the c5 agent toolkit. But specifically, this toolkit had to be surrounded in technologies crafted to deal with both the unforeseen of the interactive and the unforeseen of the authorship process.

We will unpack some of the ones more specifically behind those four descriptions here.

Long-term learning and persistence

The above stories of course, omit many implementation details; however, the first and most glaring omission are implementation *numbers* — the decisions, thresholds, limits, the choices concerning when and how things should occur. From the above descriptions of the music creatures we can read: "after there is a level of confidence in the consistency in the contents of the node-level acoustical models" — this level is a number, a **cutoff**; "important sonic events that are un-

like the contents of the nearest pose-graph node are unexpected" — this measure of unlikeness is a **scale**, specifically it is a mapping from some unspecified range to the domain of 0...1; "Having accumulated enough body material ... this learning period ends" — this is a sensitive **period**, a mapping from some unspecified value scale and a time-period to a decision that some learning epoch should end.

Setting these numbers, or equivalently calibrating the range (and the units) of the quantities that they intersect or scale, is a considerable burden to the author of a complex system. In the particular case of this installation, the burden is impressive: *The Music Creatures* installation was a complex five-computer, four-screen installation that proved impossible to assemble prior to its arrival in the gallery space — rather the "installation" was made piecemeal, agent-by-agent in a large open-plan office area. Knowing the dynamics of all four creatures when they begin to hear each other, the precise characteristics of the sound available in the galleries or even just the sound level there in advance was simply impossible.

The impossibility of this kind of foreknowledge of an interactive process is the burden of interactive art. However, it is also the condition and indeed the opportunity of interactive art. To shirk this burden by direct specification is to close off potential before it develops.

However, in each case there is a way of specifying what these numbers should be that is simultaneously more indirect and more explicit. Instead of specifying the number *directly*, we specify what this number should *achieve* and let some simple online learning technique this indirection. This, coupled with a rather Lamarkian inheritance of these learnt "traits" across multiple instantiations of the creatures, ameliorates much of the difficulty of setting these numbers and scales. The agents indirectly *learn*, and in learning *specify* the cut-offs, scalings

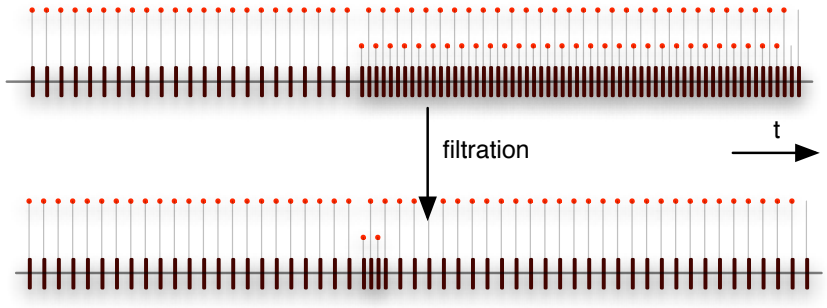


figure 42. An easy cut-off problem. Here we are trying to find a cut-off that maintains the same output rate from a filtered set of valued-events with a non-stationary distribution of values. After a brief moment of uncertainty, the windowed cut-off correctly adapts to a sudden change in the underlying statistics of the channel needing filtering. For much harder examples of this working, see figure 67 on page 269

and period-closure criteria in different ways, and we will take each of these in turn.

A **cut-off** tries to form a boundary in a range that splits the number of examples on each side into a specific percentage. To find a cut-off we store the history of examples. A simple estimator would place the cut-off point for an acceptance ratio of α this fraction of the way through a sorted list of example values. This example history is maintained over a very long window of time, since the cutoff itself is rarely changed. Keeping the examples around has the additional advantage that we can change our mind about the cut-off ratio. We shall see this technique, in a more dynamic situation, re-used in the Diagram framework, *page 269*.

For modeling **scalings** we have a richer set of options. Scaling by the maximum and minimum ever seen is one option, but again, the possibility of the data being non-stationary complicates things. The following algorithm uses a simple "effective" maximum and minimum:

initialize min a, and max b, at time $t_a \leftarrow t_b \leftarrow t_0$

then at any time t_1:

$$a_e = a + \gamma \frac{(b-a)}{2}(1 - e^{-\beta(t_1 - t_a)})$$

$$b_e = b + \gamma \frac{(a-b)}{2}(1 - e^{-\beta(t_1 - t_b)})$$

then the scaled version of a datum d is:

$$d' = (d - a_e)/(b_e - a_e)$$

if $d < a$: $a \leftarrow d$, $t_a \leftarrow t$; if $d > b$: $b \leftarrow d$, $t_b \leftarrow t$

this implements a simple forgetting strategy with a rate governed by β and a completeness governed by γ.

For small β and $\gamma \sim 1$ this strategy does not alter the distribution of example values, but simply squeezes or stretches it to fit the unit interval; thus many of the properties of the original examples survive. To make the output distribution more uniform, to "whiten" the input distribution, the music creatures use a self-organizing map trained on the history of the data to model the distribution of these data. For scalar input d, for each example presented to the map (in a random order, multiple times) we perform the following iteration:

The classic introduction to self-organizing maps (including the edge-effect phenomenon) is: **T. Kohonen**, *Self-Organizing Maps*. Springer Series in Information Sciences, Vol. 30, Springer, Berlin, Heidelberg, New York, 1995.

over the N "self-organizing" nodes at locations a_n :

$$i = \arg\min_n |d - a_n|$$

for $n = 1 \ldots N$:

$$a_n \leftarrow f(|n-i|)d + [1 - f(|n-i|)]\, a_n$$

where $f(|n-i|)$ is the self-organizing map's kernel function, for this work a Gaussian of radius $N/4$

Additionally, to avoid the "edge effects" of self-organizing maps, we modify the original self-organizing learning strategy: stretching the range of the map on each iteration to be equal to the the effective maximum and minimum computed by the previous scaling formula.

Implementation extensions for this algorithm include: ensuring that $a_{i_0} \neq a_{i_1}$ and using a more robust interpolation strategy (one that takes into account more than two of the self organizing nodes might be to use radial-basis functions after a straight line fit).

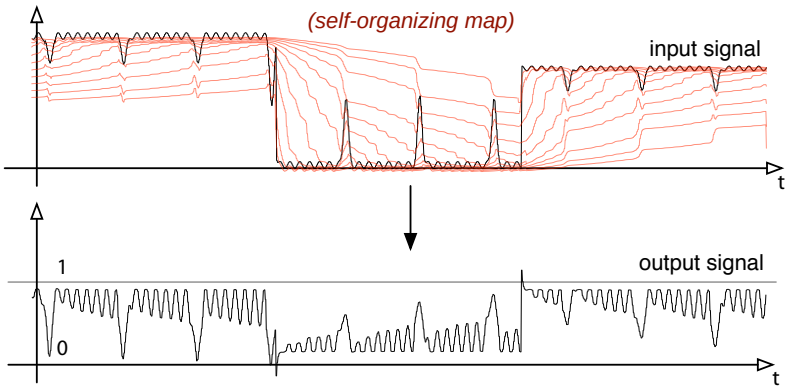


figure 43. The one-dimensional self-organizing map rescales an input signal, "whitening" its distribution, despite large shifts in the input distribution. This makes it an ideal intermediate representation for coupling systems together.

Then, given a new example, we can perform the following lookup and interpolation from this map.

for datum d on map $\{a_n\}$:

$$i_0 = \arg\min_n |d - a_n|$$

$$i_1 = \arg\min_{n \neq i_0} |d - a_n|$$

if $i_0 \leq d \leq i_1$ or $i_1 \leq d \leq i_0$:

$$\alpha = (d - a_{i_0})/(a_{i_1} - a_{i_0})$$

$$d' = (\alpha i_1 + (1 - \alpha) i_0)/N$$

else if $d > i_0$:

$$\alpha = (d - a_{i_0})/(a_{i_1} - a_{i_0})$$

$$d' = (i_0 + \alpha)/N$$

else if $d < i_0$:

$$\alpha = (d - a_{i_0})/(a_{i_1} - a_{i_0})$$

$$d' = (i_0 - \alpha)/N$$

Finally, to choose the end of a **sensitive period** based on a value v and a time t is the problem of choosing a cutoff β and a coefficient γ for the following equation:

for a process that yields a value v at time t we pronounce the end of the period if:

$$v + \gamma t > \beta$$

The task here is to accept moments that have high value v but decrease our threshold over time, getting less choosy about the v that will be accepted such that normally the process takes time t_0. There are, again, a number of ways of doing this, especially if one has information about the process generating v. But in the laziest position —a position of ignorance — is to re-scale v to the unit interval. In doing so we can eliminate the choice of β and accept if

$$2v - 1 + \gamma t > 0$$

without loss of generality. We then estimate γ given a set of examples $\{v_i, t_i\}$ and a target time t_0:

$$\hat{\gamma} = \langle 1/t_0 - 2v_i/t_i \rangle_i$$

None of the above "learning" formulae cause any particular theoretical difficulties, especially in the case where a good first guess is available in addition to the indirect specification; for the wide variety of *ad hoc* applications that they are used for in *The Music Creatures* they all learned and converged quickly. Such techniques are present at all levels of the creatures: at the basis of interpreting sound levels — for segmenting sound into acoustic events; judging when an agent has a sufficient knowledge of its sound-body mappings — so that periods can end; comparing sound models — when is a sound "new", when is it surprising? It is inconceivable that this piece could have been completed without the tactical widespread deployment of these simple techniques; it is similarly inconceivable that such a piece would have been attempted without them.

The methodologies of learning

However, such technical competence is not the end of the story. There is a significant gap between the academic reality of this simple learning and its practical use, its integration into a working practice. This "implementation difficulty"

stems from the way that agents are authored, and indeed from the fact that the agents are *being authored* while this learning is going on. This makes how this storage and recall of this data occurs surprisingly tricky.

Ideally, we'd like to specify our numbers, cutoffs, scaling etc. with as little fanfare as possible:

```
PersistedScaling magicNumber = persistedScaling("cutoff-for-loudness", 0.0f, 1f);
```

this declaration loads a long-term (that is, longer than the execution life of a creature) model, initializes a model if there isn't one already (to be between 0 and 1 in this case), prepares it to collect examples upon use:

```
for cutoffs:   passed = magicNumber.filter(value);
for scalings:  value = magicNumber.filter(value);
for periods:   passed = magicNumber.filter(value, time);
```

and arranges for it to be saved upon the termination of the program.

Clearly this persistent data faces the same challenges as faced by the persisted rendering and action attributes of *Loops, page 114*. Part of our defense against the system changing from invocation to invocation is our handling of non-stationary example sets.

But there is an additional problem not faced by the *Loops* database, an additional dimension to the problem of "naming" in complex systems. The issue stems from the need to exploit as much context and data as possible for new learnt parameters — as much context and data, that is, and no more. Consider that multiple creatures may execute the same lines of code as above, consider that the same creature may instantiate many objects that have that line in them, finally, consider that we might have a small test program that thoroughly tests this line of code before it is embedded in a larger agent. This line appears as a simple declaration, however it is woven into my creative practice in an intricate fashion. In each of these cases, we might wish to exploit the knowledge accreted in the test programs, share knowledge accumulated between invocations of related creatures. Thus, we wish to structure our persistent parameters in such a way that as authors we remain mobile, and that the data can move with us: that we can move from small test stage, an exploratory phase, testing inside a larger system, back to a small debug session.

A context-tree based solution *page, 211* can help with the problem of multiple instantiated learnt parameters — a parameter instantiated in a different context counts as a separate, different learnt parameter that might share past history with parent contexts. Since the current context is an unambiguous chain of names it can be stored just like any uniform resource locator. However, this context does not suffice as the sole context for these parameters.

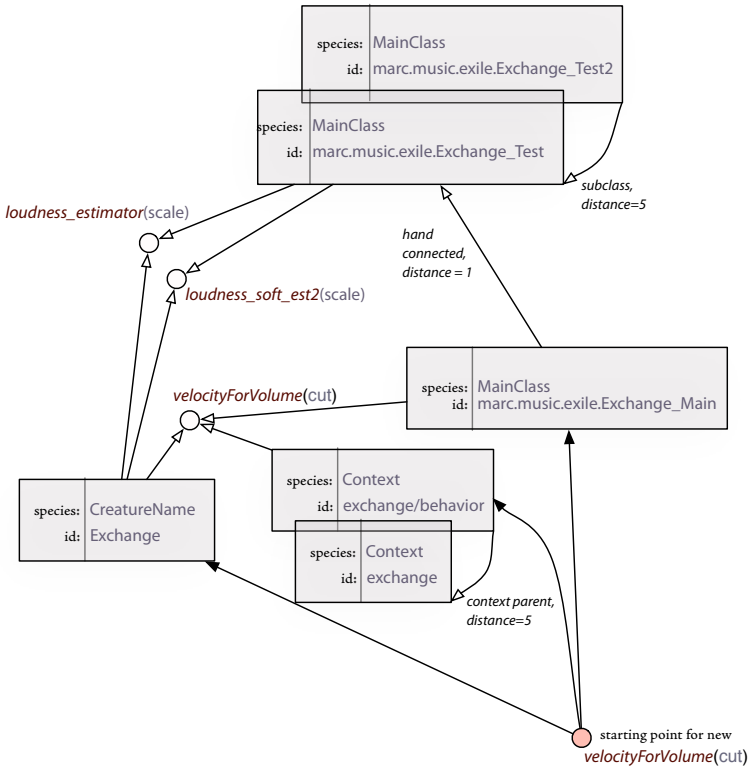


figure 44. Initialization material for newly created models is found by breadth-first search across the database. Here a new velocityForVolume(cut) would be initialized with material from a previously learnt model, learnt in the same main class (as well as the same context, and the same creature name). This node is then placed in this graph at this location.

We can list a number of potentially important "contexts": the "name" of the creature, or related, the name of the "main class" (the entry point into the entire program running the agent), and finally the name of the learnt parameter itself. Unfortunately, there seems to be no single ordering of these contexts that makes sense.

Rather than attempting to formulate a single tree structure with these learnt parameters at various leafs and nodes, we form a heterogeneous, but directed graph with contexts and learnt parameters connected together. Collecting information about the data that should be behind a learnt parameter is done by breadth-first search over this graph structure.

Many of the edges of this graph can be established automatically — for example edges are formed from a main class to its superclass (but not the other way), if the superclass existed in the graph. Other weighted edges can be created, and permanently deleted with a graphical utility (for later work, this was assembled and written in *Fluid*). This same utility rolls back the database to a previous state (when we make an experimental change); tags a database state with a useful label (something more easily remembered than a date and something that tracks the state of the project); and can list differences between database states (to allow an investigation of the project as it is changing).

This connected-graph-aware versioning system is an extension of the ideas used for *Loops*'s rendering parameter database. It is thus neither the first, nor will it be the last persistent store required, *page 225*; required, that is, to take a technology that in an academic sense is working perfectly, and turn it into a technique that is actually integrated into a working practice.

4. Advanced flow control — coroutines, radial-basis channels, and an introduction to the "language interventions"

The previous section identified a particular class of unobtainable foreknowledge in authoring interactive works — knowledge about both the coarse structuring and the fine details of the world in which the work will be installed. In finding a few algorithms I built the necessary support to ensure that they could and would be widely deployed throughout the work.

Another broad class of uncertainties faced by the agent author is less concerned with the world and the agent and more concerned with the interactions between systems internal to the agent itself. Specifically, the uncertainties faced where multiple parts of an agent fight for, or cooperate over, control over another. Such arbitration and control problems occur throughout the agent metaphor. Some are action-selection problems — objects fight for attention, action-tuples compete for expression. But others are coupled more directly to perceptual and motor systems.

Imperative programming languages earn their name by their focus on a sequence ordering of commands. With a small number of exceptional flow control structures, programs in such languages start at the top of the page, and execute line-by-line until they get to the bottom. Advanced metaphors, constructed "on top" of imperative languages increasingly complicate this flow.

The c5 toolkit makes extensive use of one set of such complications — object orientation. However, as we shall see in a number of places in this work there is an essential conflict between the flow of serial execution in an imperative programming language and the flow of time in the life-cycle of an agent. These conflicts necessitate a less generic set of extensions to the set of non-imperative flow control structures, ones specifically tailored to ameliorate the conflict be-

tween the orderly imperative of what it that an agent author is trying to achieve this moment and the disorderly, diffuse chaos of other parts of the agent framework already assembled.

As we proceed down the perception → action selection → motor control chain this conflict becomes increasingly severe. Although the perception system monitors the world as it finds it, in realtime, for the most part its flow control can be coupled quite directly to the world and only occasionally reconfigured by an action system — the world dictates what code gets executed when. As we move to the parts of an agent that take responsibility for maintaining long-term behavioral coherence the differences between the flow of this coherence and the flow of an imperative language widens to a chasm.

In this work, and many other agent systems, by the time we have reached the action-selection layer we have small parcels of imperative-code (the action payloads of our action-tuples) and a highly distributed flow-control structure (the action selection mechanism and its coupling to other systems) that changes often and changes on different time-scales. None of this code is written in what one might call an imperative spirit. As we head towards the motor system even these parcels of code need to be dissolved as the granularity of control becomes equal to the granularity of the agent's "update cycle", essentially the frame-rate, the control-rate, or the clock-quantum of the simulated world. Programming at this level is hard — hard to write, hard to maintain, it is a barren landscape for artistic intention because no thought, it seems, can be continued in any meaningful way for more than a few lines of code.

A good introduction to continuation-passing-style that is less language specific than many is: **D. P. Friedman, M. Wand, C. T. Hayes**, *Essentials of Programming Languages*. MIT Press, 2001.

The canonical definition of the Java language is **J. Gosling, B. Joy, G. L. Steele Jr., G. Bracha**, *The Java Language Specification, 3rd edition*, Addison-Wesley, 2005.

for Scheme, the following is useful: G. L. **Steel, G. J. Sussman,** *The Revised Report on SCHEME*, MIT AI Lab Memo 452. 198.

for Python, see http://www.python.org

for the Common Lisp Object System, **G. L Steele**, *Common LISP: The Language*, Bedford: Digital Press, 1990.

for C#, http://www.ecma-international.org/publications/standards/Ecma-334.htm

Part of the response to the problem will be to look to more sophisticated flow structures. I'll use this as a basis for a critique of current digital art-making environments (and the concepts they encourage). And I shall show our main, conventional object-oriented programming languages hybridized with ideas from other languages. When this hybridization occurs it will concern flow control features: I will introduce a "continuation-passing style" implementation in Java — inspired by languages such as Scheme; a "coroutine" based system implemented in both Java and Python — inspired by Python's implementation, but finding its roots from the days before ubiquitous threading; and a set of "complex, multiple-dispatch" techniques inspired by the Common Lisp Object System.

Adding these exotic language extensions and libraries to a base language requires some justification. The choice here — Java — is a popular language that has found and thrived in a fertile niche between sophisticated abstraction (the reflective capabilities of the object model and the malleability of the virtual machine code) and a practical rather than polemic level of object orientation tried and tested in other languages and implemented well. It offered support for exceptions, a safe memory model and a runtime type system at a time when implementations of these features in other mainstream languages were experimental and poorly tested. It seems a suitable language for large, complex collaborative development especially in teams with a variety of skill levels and today is surrounded by a number of sophisticated programming environments.

Where it falls down is perhaps on the smaller scale, and it is here that the so-called dynamic languages take over. Removing the type system and its textual burdens (declarations, casts, and various commitments to organizing one's abstractions) these languages are generally considered ideal for testing, for gluing together pre-existing objects and for "interstitial coding". They have a tendency to encourage language features that are thought by many to thoroughly scupper

the maintainability of large, multi-person code-bases. At this price they often offer direct or indirect support for malleable syntax and modifiable meta-class semantics that allow the appearance and benefits of domain-specific languages without any of the burden of writing and maintain one's own tool chain. Thus I augment the choice of Java here as the main programming language with a more dynamic language — the Java-based implementation of Python. Its important to note that its role as an *interstitial* language in this work is enhanced by the open-source nature of this implementation, and that it, rather than Java, forms the basis of the programming environment developed in the last section of this thesis, *Fluid*.

The choice of language for this work is not casual, indeed part of the argument of this thesis is that programming languages and the tools around them are so malleable, interesting and important in one's work that the artist bears a responsibility for their choice of language and the design of the language — that they should not be a passive consumer of a toolset. And while discussion concerning the relative merits of programming languages typically generate more heat than light, the implementation details of the broad concepts and the language-level interventions of this thesis remain significant even given the perennially unjustifiable choice of language itself.

In similar languages (and there is a lot of interest in Java-like languages for example the more recent C#, or perhaps even the contemporary style C++) it is expected that some of the implementation techniques, or at the very least styles will carry over. In other different but similarly sophisticated languages these techniques will no doubt be achievable in some other ways, and the implementation specifics here will have little to offer. Still, however, the arguments for their applicability to the problems discussed here, extension to the programmer's palette and place in agent architectures in general are still expected to stand.

This "imperative disconnect" is the second thing that the narrative descriptions of the music creature's behaviors obscure, or at the very least gloss over. How is the passing of time authored? — what happens in the space between paragraphs? Quoting again from the descriptions: "Having accumulated enough body material ... this learning period ends" — this is a *scripted* transition that waits for some condition to be true and takes one action system configuration to another; "nodes ... are slowly dropped from the graph until only two remain" — this is another *scripted* condition. "in order to 'pluck' ... it takes the nodes either side ... binds them temporarily" — this is a *scripted* motor program.

We have a number of techniques, not least of all the action systems themselves for waiting for a condition to occur before doing something. But perhaps there ought to be a lighter-weight, and more imperative, way of specifying these well-ordered sequences of tasks. Even the "payloads" of actions themselves often turn out to have a "scriptable" life-cycle — "do something when at the start, keep doing this for a while, and then make sure that this is done before finishing". This tendency is not confined to *The Music Creatures*. An example from the action-tuples of *alphaWolf* might "read" as follows "to set this variable in working memory, tell the motor system to perform a particular action, wait until it is done, or for a while; go over to that other wolf and *then* growl". These are scripts in miniature.

There is a role for such lightweight languages in the cases where the order of events are predicable and the number of possible paths and interactions through the script code are small. We see such small programs persistently at the motor level. Here is an example: action systems ask for a particular goal to be achieved and the motor system must navigate that task by calling up a walking animation until the agent is close enough to take a half-step and then, shortly after, it fi-

nally can sit down. This is a suitable problem for scripting (as opposed to a finer-grain representation like a perception / action system) because the order of events is well known and the behavior in the case of interruption is well defined — for such simple things there should be a way of just writing them down and having them execute. The remainder of this section is devoted to a technical description of the incorporation into Java of a language pattern that allows just this.

We have seen repeatedly that sequencing action over multiple execution cycles usually requires special effort for imperative programming languages. Typically, either effort is applied to rethink the sequencing in a way that is *reentrant* (in essence using a repeatedly executed switch statement) or to build the sequencing by imperatively constructing a kind of executable data-structure. We can do better using the reflexive powers of a language such as Java to implement an older programming idea called coroutines. A coroutine is a restartable procedure (or here, method call). Coroutines are easily implemented in a language with proper continuations, or even reflexive access to the stack-frame structure, and they look to be implementable in Java with some load-time byte-code manipulation. The implementation comments here are therefore of less interest to other language implementations, but how coroutines are used should still be interesting. Now we can make apparently imperative scripts →

for precedent for these "continuations" in Java, the RIFE project: http://rifers.org/wiki/display/RIFE/Home

The method used here relies on the the ability to recover the order of method declarations in a source file at execution time. The order is compiler-dependent, however all compilers used for this work since 2000 (both as part of the standard Java developer distribution and as part of the Eclipse project) have emitted byte code that is ordered in this way.

The Eclipse project: http://www.eclipse.org

```
rc beginning(){
        System.out.println(" beginning ");
        return progress;
}

int i = 0;
rc waitAWhile(){
        if (i++<10) return wait;
        else return progress;
}

rc thatsIt(){
        System.out.println(" that's it ");
        return stop;
}
```

This object is given to an executor that interprets the return codes of these methods. The critical point here is that at each return the rest of the system (and in most cases this is almost all of the rest of the agent) gets a chance to execute. Although for the purposes of the above example we would certainly be better off in a language that supported coroutines directly (for example, the above is one method and six lines in Python) the fact that we are forced to return something that describes the next move (here, wait, proceed or stop) turns out to be an interesting opportunity. For, to be powerful enough to be up to the tasks that we ask of them, our scripts (clean as they are) need to retain some of the best features of object-oriented programming. In particular, it would be better if they were a little more reusable, a little more component-oriented and more composable.

To achieve this, we begin by extending the lexicon of things that our coroutine method elements can return and by making the executor keep track of a stack of coroutines, only the top of which is executed. This is, in a sense, a recreation of

the stack behavior of the virtual machine, with added reflexivity and flexibility — a kind of homemade *continuation-passing style* of programming. In addition to the singleton objects, ***progress***, ***wait*** and ***stop*** we add: the singleton ***failure*** — which signals an exceptional "error" condition has occurred; **push**(*coroutine*) — which pushes a coroutine ahead of this one on the stack; **replace**(*coroutine*) — which replaces the current coroutine with another; **after**(*coroutine*) — adds this coroutine to the stack just before the current one (it will thus execute after this one); **trap**(*coroutine*) — upon failure, the default behavior is to tear the stack down and return failure; adding a trap to the stack intervenes in this destruction, and allows the trap to execute instead.

From these primitives we can build more useful coroutines that exploit the stack structure for their implementation but don't necessarily reference it in their definition, for example: **andWhile**(*coroutine[]*) — while these coroutines are progress-ing or wait-ing, keep running this coroutine otherwise stop; **needs**(*coroutine[]*) — needs all of these coroutines to progress before this coroutine progresses; **obtain**(*coroutine[]*) — a combination of both of these, wait until each coroutine progresses before continuing and only run while they aren't stop or failure.

The name of the last "return code" hints at where some of this is heading. Coroutines provide an excellent interface to a resource model. The simplest resource is something that only one object can own. Other objects need to wait in line to own the object or can a perhaps steal the resource, forcing the lock. If they are waiting in line, then their coroutine-based attempts to own the resource are *wait*-ing too; if they force the lock, the coroutine-based interface used by the ex-owner returns failure. Through obtain(...) and needs(...) we can attempt to claim a number of locks at the same time and give up if no progress is made on any front, before trying something else.

These coroutine / resource combinations are extremely useful for guarding access to limited resources throughout *The Music Creatures* and successive work. They are used in a strictly technical way: apportioning use of the two running video texture channels in 22 and making sure that these textures are switched out only when they are not on the screen. And they are used in a more "agent oriented" way: distributing the motor-program control over the reorganization of *tile*'s body segments — a reorganization which is a short series of uninterruptible, but potentially overlapping procedural animations; forcing the "physical constraint" that *network* suffers under before it gets to play its nodes; and of course, limiting the life-cycle transitions out of sensitive periods (essentially, executed with needs({*aCertainAmountOfLearning*})).

The coroutine / stack model also generalizes the previous stack based model for motor programs, discussed *page* 82 that had surprising longevity — it was used for *alphaWolf*, *Dobie*, *Loops*, *Max*, an early version of *The Music Creatures* and still makes an appearance in current work at the MIT Media Lab (for example, the work of The Robotic Life group). This simpler model aimed first for the reusability of components — stacks were assembled of motor programs that were themselves made out of stacks of pre-made components — but what was lost was the scriptability. Nothing much was left of the underlying imperative language (Java) which, of course, has much in its favor when it come to the task of programming. Further, it had no equivalent of trap(*coroutine*), and no exceptional error handling which made conflict handling and disassembly of motor program stacks troubling and error-prone.

for an explicit discussion of Leo and the pose-graph: **C. Breazeal, D. Buchsbaum, J. Gray, D. Gatenby, B. Blumberg,** *Learning From and About Others: Towards Using Imitation to Bootstrap the Social Understanding of Others by Robots.* Artificial Life, 11 (1), 2005.

I am indebted to Robotic Life group researcher **Jesse Gray** for bringing these ongoing issues with the single stack model to my attention, long after I had abandoned them myself.

Blumberg's motor system is described in: **B. M. Blumberg,** Old Tricks, New Dogs: Ethology and Interactive Creatures. Media Laboratory. PhD Thesis. MIT. 1996.

Minsky's centrality of (negotiable) resources is in his forthcoming *Emotion Machine*.

These extensions are not simply academic investigations into exotic programming models — the lack of any formal cooperation between motor system stacks was only tenable for characters who had only one primary motor system to begin with and characters, such as the Robotic Life Group's *Leo*, that assemble many layering motor systems together have spent much time constructing such cooperation. Its lack of a fine-grained resource-like view onto the motor system and its insistence on a single stack bought some simplicity at the expense of much power.

While the formal principles of this "new" language for motor programming are new, the underlying insight that there is a lock/resource model aspect to motor systems is hardly novel. The gated access to shared resources has formed a core part of every modern operating system for as least as long as multitasking; the importance of necessary and preferable locks forms one of the key insights of Blumberg's motor system. In a broader context still, Minsky places resources with apparently similar life-cycle constraints at the core of his recent AI architecture — but the *scriptability* of the life-cycles of these resources in an agent framework is what makes them powerful in my formulation here.

Adapting bodies

The Music Creatures is as much about the bodies of the creatures than the music that they make, and more to do with rethinking the figurative in computer graphics than about reinventing a fragmentary minimalism in computer music. The third thing missing in the narrative descriptions of their behaviors was the specifics of the control structures the agents had over their bodies. Again, quoting from the narrative overviews: "...its body representation is underspecified and node local" — this is a specific kind of body representation that can be used to generate movement; "the creature must have physical support from an underlying lattice structure, it takes the nodes either side of the node to

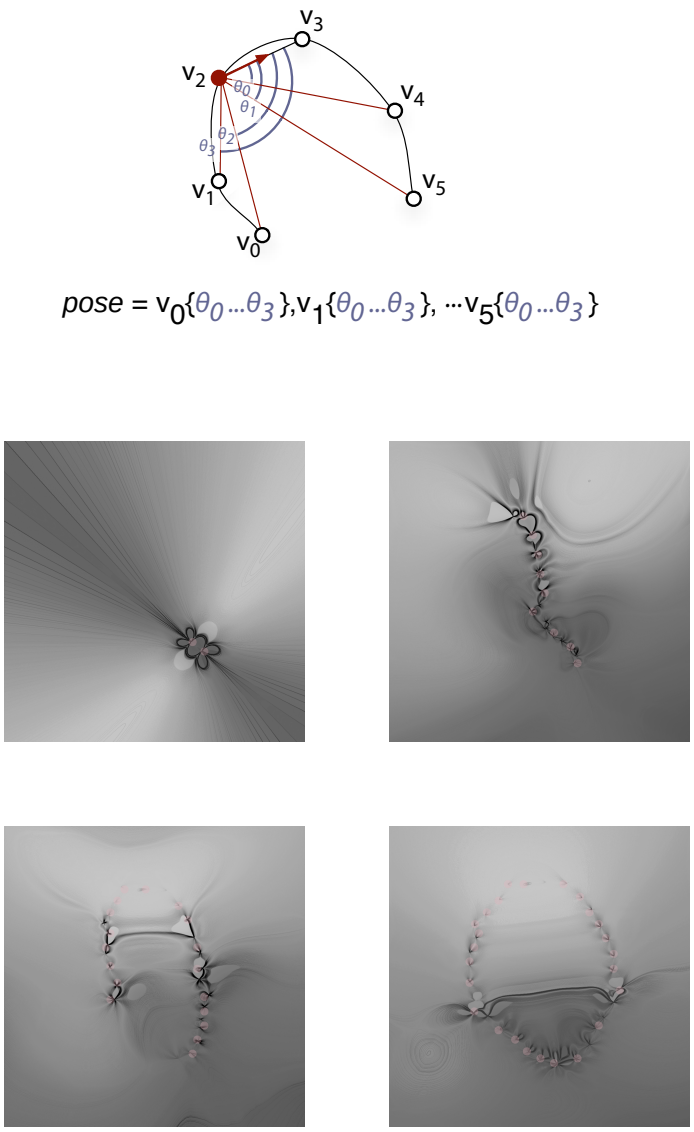


figure 45. *Line* has a deliberately perverse pose representation. Rather than remembering the location of each of its control nodes, the pose records an angular representation of where each node is with respect to each other. *Line* attempts to move from pose to pose using this representation embedded in a spring-like physics system. The result is a pose representation that is capable of transitioning between material, but does with effort and difficulty. This pose representation also gives rise to the potential fields illustrated in the lower half of this figure. These fields show, given a fragment of the growing line, where the next control node should be placed in order, here, to create a circle.

play and binds them" — this is procedurally generated animation of an altogether more flexible kind of body. "... as exchange learns the mapping from surface parameters to the resulting physical movement." — this is learning to procedurally control a body.

The pose-graph, although flexible, isn't for every situation. It handles very well the case when the control over a body can be discretized, even if those discrete units are small blend spaces. It isn't appropriate for bodies that offer a high number of independent degrees of freedom that are not broken down, or do not need to be broken down into "animations" or "poses". *Network* and *tile* are creatures with such bodies.

Before moving onto the more abstract control problems that the music creatures face, we should look at how they do exploit the pose-graph motor system to support the simple learning that they achieve.

Both the *exchange* and *line* incorporate learning into their use of the pose-graph — both are simple, data-driven, example-based techniques that gain any power they have over the body of the creature because of the framework in which they are placed.

The problem faced by *exchange* is to remember what effect its pre-made animation material has on the navigation of its body through its world. It will then use this knowledge to navigate back to various locations in order to "strike" the musical material stored there. It accomplishes this simple task by annotating its pose-graph — which begins with three, simply connected "animations" that exploit the creature's four rotational, parametric degrees of freedom — with the results of playing out that frame of animation. The multiple examples remembered from the ultimate translation vectors that result on the creature's corner vertices are stored in a clustering structure not unlike that of *Loops*'s rendering

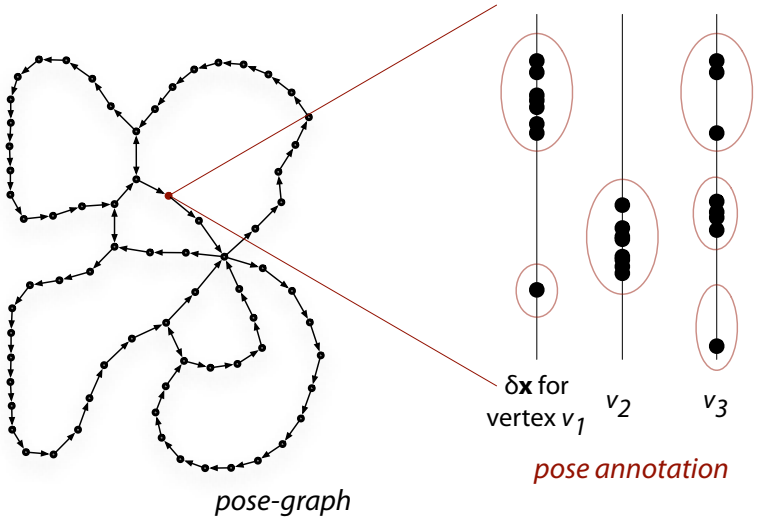


figure 46. The motor learning in *exchange* is performed by annotating the contents of its primitive pose-graph with the effect of performing those poses. Theses annotations store the effective directional change of three of the bodies vertices. Given this structure the pose-graph can now be searched in a different-way, based on these annotations. While not accounting for the momentum that the creature has, this is sufficient to allow a certain, perceptible, amount of goal-directed movement by the creature.

parameters. Having filled this data-structure with examples, *exchange* can search outwards through its motion graph in order to synthesize movement that propels the creature toward a particular point in space. The creature itself is "damped" back to the origin of its world, so it can never go too far astray.

Line operates in a similar way, by annotating the pose-graph structure. Here, however, it is not the consequences of movement that are stored in the nodes, but the a record of the sound present when the pose has been played out. Perceptual clusters of sound that fission propagate topological changes to the pose-graph itself — new pose-material is sampled directly from physics-based body.

Finally, it is important to revisit our general description, *page 85*, of the role of the motor system in an interactive work in the light of these simple learning techniques. We have already seen the motor system's primary force in organizing pre-made material, incorporating "content", made using other, non-agent techniques into the agent and becoming the location where this material is blended, spliced, resequenced. With the simple learning techniques described above, the surface between the agent and the pre-made moves outwards, in favor of the artist. The agent can now use material, not as "content", but as scaffolding for learning, the seed for the agent's own material.

The generic radial-basis channel 1 — flow blending

In general, however, the pose-graph has little to say about the problem of the body-representations for non-figurative creatures — it offers a control structure for the body for *line* but not the structure for it to control. It will have even less to say about the body (as opposed to the pose) representations for many of the creatures we see later, which are adapting their bodies or sampling them from motion-capture data. One generic framework for the construction of these

bodies will be presented soon, *page 319*. But we can treat some of these concerns here.

To review: there appear to be two essential and unavoidable characteristics of the environment in which "motor-systems" work, two unavoidable "unforeseeable" problems in authoring agents: firstly, multiple pieces of code (for example, action-tuples) attempt to access the same resources (for example, joint-angles) and these read and write accesses must be arbitrated, blended, or rejected outright. Secondly, these processes want to have their accesses at one particular time-scale, duration or rhythm (for example, when they become active) but want their effects and control to last over a different time-scale (for example, starting slowly, but continuing until a goal is achieved or until a goal is unreachable, fading out over time, etc.). We'll call the first of these the **blending / arbitration problem** and the second, because it comes from a disconnect between the flow of execution in our action systems and the flow of execution in the motor system, the **motor flow control problem**.

Both of these problems are approached by the coroutine / resource model given above, but only in the case where the arbitration and the flow problem is in some way *localizable* and only when the control itself is *symbolic*. This is often the case in high level terms in pose-graph motor systems, where there is a well defined surface between the action and motor system — and the currency of exchange across this membrane is named (that is, symbolic) poses.

However, at the lowest levels of the motor system, the communication with the body, there are often *blending* problems that cut across coroutines and are numeric rather than symbolic. These numerical blends are often the kinds of control that we wish to exert over processes — when the exit of one routine should be *smoothly* eased-out, and the entrance of another should be strongly percussive. As the motor systems that are constructed for our agents become progres-

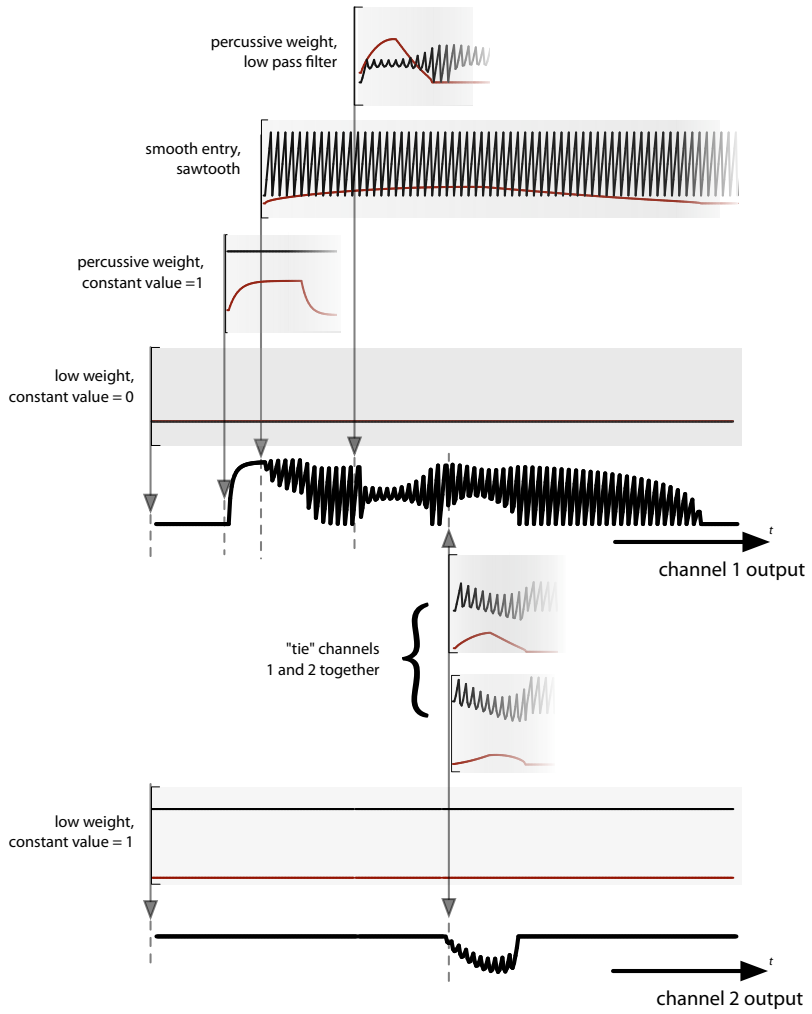


figure 47. The generic radial-basis channel is a place where multiple processes can present small, overlapping, signal processing primitives in a *ad hoc* fashion. In this example here a few postings are added to two channels, including one that tries to temporarily tie the value of both of these channels together.

sively more hybrid — incorporating both "content driven" pose-graphs and "process-driven" procedurally generated movement, these blending and layering problems will dominate as a whole host of distributed processes throw material at the control structures of the agents' bodies.

We will start by discussing a simple, but powerful general-purpose approach to constructing motor-system elements that address these two problems — the generic radial-basis channel. This has worked well for the creation of some parts of not just *The Music Creatures*, but every work since; beyond the content driven pose-graph structure.

In its simplest form a channel is a composite of the following elements: a value representation, a number of "postings" to the channel, and a (monotonically increasing) time-base. A value representation exposes a mathematical space that is at least as large as a vector space (we shall note in passing the possibilities of value representations that are non-commutative and thus are not vector spaces nor even fields) in terms of the following primitive operations:

```
interface ValueRepresentation<t_value>{

    t_value zero();
    t_value add(t_value a, float w, t_value b);
}
```

These two operations are chosen to be the minimal set required to implement an incremental weighted average. A posting is an object that has the ability to compute, given a time-base, a *value* v that is compatible with this value representation, a scalar *weight* w and a *time marker* T . Given its set of postings and a value representation, a channel can compute its value at the time indicated by its time-base by summing all the weighted values of its postings. In addition to computing its value at each update cycle, the channel also considers culling postings from the active set: postings are removed if their net relative contribu-

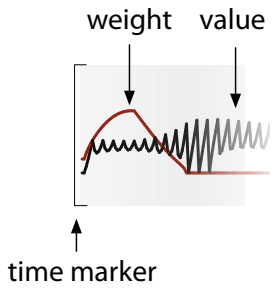


figure 48. The anatomy of a "posting" to a generic radial-basis channel — a weight function, a value (both of which vary over time) and a time marker (which may shift after posting).

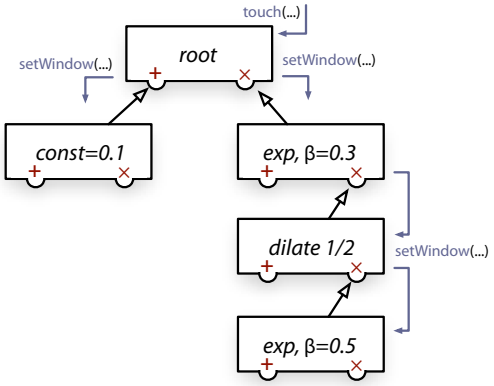


figure 49. A library of generic window functions can be created and combined using this "plus / multiply" tree structure. Each of these windows can be dynamically expanded in response to setWindow(...) messages.

tion to the channel falls below $\varepsilon \sim 10^{-6}$ and the time-base is after the posting's time marker. This arrangement shares responsibility for the removal of a posting, ensuring that elements that have an important effect on the value of the channel are kept, and that postings that temporarily have little effect can prevent themselves from being removed (by moving their time-marker into the future).

The implementation of a posting can generally be broken down into a window generator, that generates the weight for the posting and a value generator that decides on the value. Critically, the the value of a channel is changed in a very orderly fashion — first each posting is updated (in addition order), then each is asked for a value and a weight, then these values and weights are combined incrementally using the value representation, and finally, the channel checks for culling possibilities. This allows postings to immediately remove themselves from the channel — by setting their weights to zero, and their time-markers to $-\infty$ — in the knowledge that they will never be updated again; further, this gives new postings access to the value and total weight of the channel prior to their entry — useful for smoothing their entry and setting the scale of their windows.

With these simple features it is easy to build a lexicon of window functions and generic value generators. Window functions can be multiplied, blended and added, for they are just scalar functions and the value representation interface is complete enough to enable the creation of various filters without binding to the underlying representation. Window functions typically provide compact support in time — hence the name of this over-arching framework: generic *radial-basis* channel — and also share some of the control over the time marker calculation. Both these components are reusable and easy to write. But it is the control over the "posting" and its life-cycle, the relationship to the post-*ing* to the post-*er* that is more interesting, and indeed is what brought us to this structure in the first place.

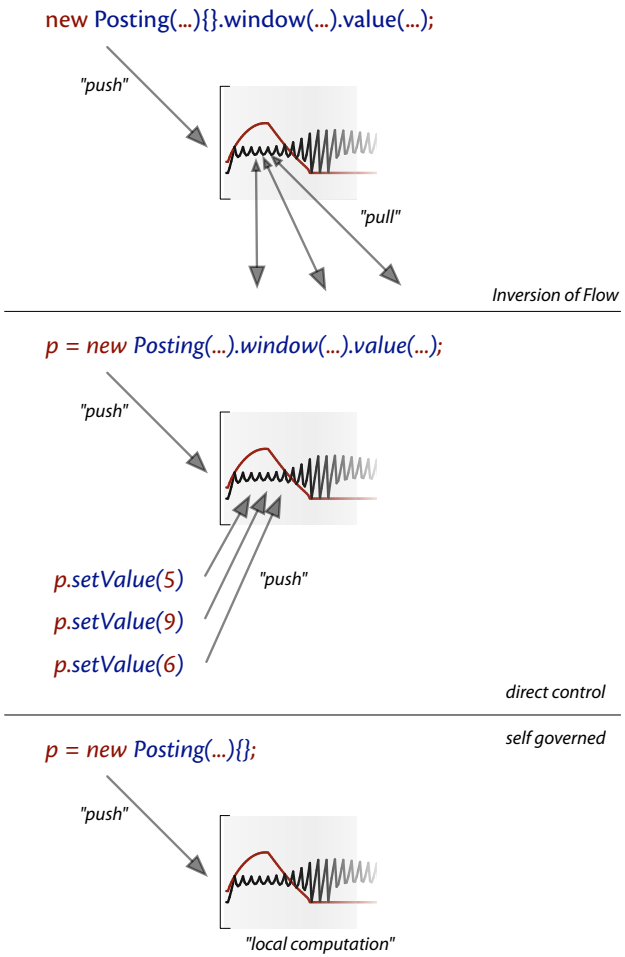


figure 50. Generic radial-basis channels offer three broad categories of flow decoupling.

We can characterize three kinds of relationship:

Ballistic — some postings are sent and then never communicated directly with again. These postings are free to pull the material and data that they need to compute their values and their ending strategy. A posting like this marks an inversion of control flow from an imperative style that pushes data to a location to one where a single imperative push of *control* to a location that subsequently *pulls* data.

Touchable — Other postings are less independent and the post-*er* frequently sets their value. However, unlike in the conventional forward situation of a purely imperative strategy, the flow of the post-*er* is shielded from the burdens of having to set a value on a fixed schedule, or a fixed order. One can construct windowing functions in this case that begin to decay away in the absence of new data; postings constructed with these windows needed to be "touched" every so often to keep them valid, otherwise they eventually fade to nothing. This allows an care-free disappearance of the post-*er* and a graceful fade out of the posting's effects.

Persistent — Finally, some postings are meant never to be culled, and manage the global characteristics of the channel (limiting velocities, smoothing over large changes etc.) or control the behavior of the channel in the absence of any other postings (drifting back to a particular set point, for example).

The generic radial-basis channel forms the foundation of a number of the bodies of the more exotic agents presented in this work (in particular it forms part of the blendable body framework constructed for *how long...* in addition to its service here in *The Music Creatures*).

This broad use is supported by a broad range of value representations. At the level of a point, it's easy to see how a three-dimensional position vector can be

exposed directly by a value-representation interface. More interesting value representations can also be written, with a little more care, for infinite lines, finite line segments, infinite planes and finite triangles.

For example the piece *22* uses a varying set infinite lines as "fiducial marks". Sometimes these lines couple to geometry in the scene, other times their movement, their momentum is connected to the movement of the performer. Clearly this is a situation where we have several procedural algorithms operating at different time-scales that need an authorable blending system. The value representation is straightforward:

2D-infinite lines: value representation is a homogeneous 5-vector:

$$(x_1\alpha, y_1\alpha, x_2\alpha, y_2\alpha, \alpha) = (x'_1, y'_1, x'_2, y'_2, \alpha)$$

The infinite line goes through (x_1, y_1) and (x_2, y_2). To form $C = Aw + B$ we find the point on A, p closest to the midpoint of B:

$$(B_{x_1} + B_{x_2}, B_{y_1} + B_{y_2})/2;$$

and form two new points, equidistant from p on line A:

$$(A^\star_{x_1}, A^\star_{y_1}, A^\star_{x_2}, A^\star_{y_2})$$

We then blend $A^\star \cdot w$ with B:

$$C = (A^\star_{x'_1} w + B_{x'_1}, A^\star_{y'_1} w + B_{y'_1}, A^\star_{x'_2} w + B_{x'_2}, A^\star_{\alpha} w + B_{\alpha})$$

Using an exponential mapping, we can form a blend space for Quaternions and have orientation valued channels. Scalar representations make channels excellent candidates for modeling the motivations and emotions of an agent — aspects of agents that are good examples of values that are influenced by a variety of systems on a variety of time-scales, while also having set points to drift back to. Low-, high- and band-pass filters can be constructed that are independent of

the value representation; persistent postings can finesse the signal properties of channels, and limit velocities. Whole procedural networks can be constructed, tested, and reused in a value-agnostic fashion.

Other than window and value-generating functions we can extend this basic framework in a number of ways: we can produce exotic value representations and extend the idea of an incremental weighted sum, *page 354*; we can replace the incremental weighted sum altogether with another structure that combines multiple values and weights, *below*; and we can, with a few more constraints on the interface that postings present to the world, allow other processes to inspect, move and organize postings, *page 248*.

The generic radial-basis channel 2 — rediscovering action-selection

Sometimes the degrees of freedom that creatures' bodies offer can be expressed succinctly using these radial-basis channels: for example *tile* doesn't need much more than channels to control its tiles (one channel per tile with a value representation that has parenting information as well as orientation); *exchange* has four rotational degrees of freedom (here, a channel each). But *network* has multiple processes fighting for control over its body. For clarity of movement just one process must be able to gain control of the movement of the network and cause it to move to the lattice. A purely channel-based weighted average is a fight without a winner.

Sometimes, then, a weighted average is inappropriate. Rather than blending between two different ideas of what a particular value, position, rotation, line or mesh should be, it would be better to pick one and stay with it for a while. Of course, this intuition needs some careful refinement — what constitutes two different ideas of what a particular value should be, and what does it mean to stay with one for a while? No matter how carefully we build the control struc-

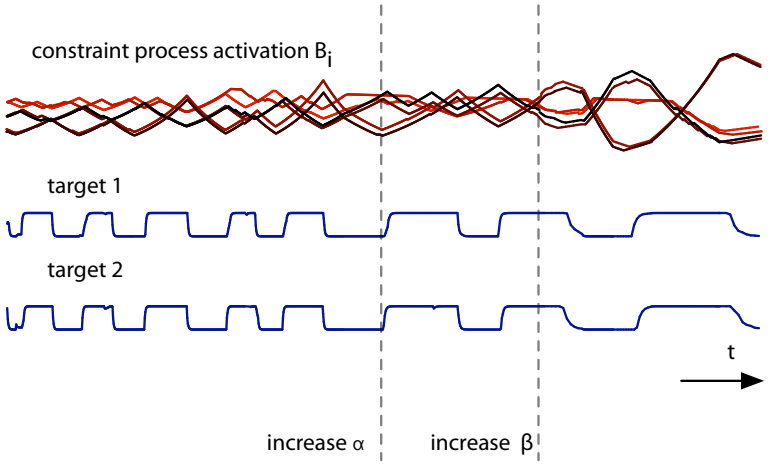


figure 51. In this figure a set of six postings are configured to influence two scalar values. The value representation in this case is a pair of proposed changes to the two targets. Two processes try and make targets 1 and 2 equal to zero; two others try to make them equal to one; and a final pair of postings try to make target 1 equal to target 2 and vice versa. The results of these (over-constrained) processes, as arbitrated by the competitive radial-basis channel structure, is to cycle through the two most stable states, targets 1 and 2 having almost the same value for almost all of the time yet oscillating between zero and one.

ture for the weights, we cannot pick the correct weights for a blended average without looking more closely at the contents of the blend itself.

So we need to extend the value representation to give a little more insight into the material being blended.

```
interface ExtendedValueRepresentation<t_value>{

    t_value zero();
    t_value add(t_value a, float w, t_value b);

➡   float normalizedDistance(t_value value1, t_value value2);
}
```

This additional method provides a normalized distance between two values inside the representation — a distance of 1 means that these values are completely different (that is, two different ideas of a value), a distance of 0 means that they are indistinguishable (that is, they are the same idea).

Now we need a structure that modifies the weights obtained from the postings to the generic radial-basis channel.

Consider formulating a matrix D of normalized distances where each element D_{ij} is the distance between posting i and posting j, with $i, j \in 1..N$ and N as the number of postings. The idea is to use this matrix D to *sharpen* the contrast between the weights of the postings W_i such that incompatible postings are not blended, but rather the strongest one wins.

Initially, we set:

$$W' = W$$

We perform the following iteration for $\forall i$,

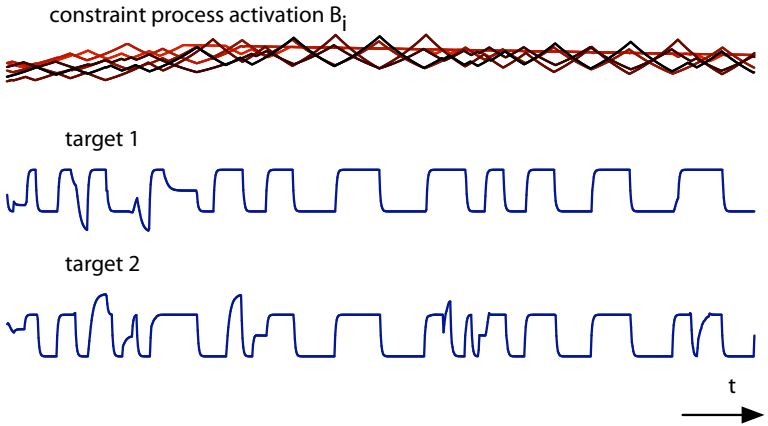


figure 52. A more complex example than the previous case. Six processes again, however two try to make target 1 equal to minus one and one, while two try to make target 2 equal to zero and two; the remaining two are as before, but compete with lower weight. The exploration of the partial solutions to this unsatisfiable constraint problem oscillates randomly; however the two channels remain highly correlated.

$$T_i \leftarrow \sum_{j=1}^{N} W'_j D_{ij}$$

$$W'_i \leftarrow W'_i \cdot (1 - T_i / \max_{j=1}^{N} T_j)$$

$$W' \leftarrow |W'|$$

With each iteration, postings that provide values that conflict with each other, mutually inhibit each other, with their power of this inhibition set by their weights. We can continue this iteration until convergence to produce a winner-take-all weight vector, which will either have only a single non-zero entry or multiple non-zero entries that correspond to zero normalized distances. This encapsulates the intuition that we should not blend between "two ideas" of the same value.

Alternatively, we can run a small number, q ,of iterations to increase the contrast between weights without removing all of the conflict. Rather than fixing q ahead of time, we might choose a $d_{\max}$ such that $d_{\max} = |W' - W|$ (we terminate should $|W' - W| > d_{\max}$ and set W' such that it is $d_{\max}$ away from W , noting that the furthest away W' could be from W , that is, the largest possible $d_{\max}$, is $\sqrt{N}$).

This may suffice for channels that have many postings that enter and exit in a constant flux. However, for channels with long-term postings the temporal dynamics of this algorithm are too clear-cut. Stronger postings that conflict always crush weaker postings — no exploration occurs. Two alterations change this:

For the purposes of implementation we must remember that this vector notation is strictly mathematical, and in fact N varies with the number of the postings.

firstly, a long-term memory, controlled by the time constant α, updated after all of the multiple iterations of the above algorithm:

$$B_i \leftarrow \alpha B_i + (1-\alpha)(1-W_i')$$

with $\alpha \in 0..1$

and, if $W_i' > \varepsilon$ then $G_i = \beta$ else $G_i = 1$, with $\beta \geq 1$

secondly, we use this memory to modulate the weights as they enter the algorithm:

$$W_{i,\text{initial}} \leftarrow W_i B_i G_i$$

Together α and β control the time-constant of changeover that we would expect between two $d = 1$ postings with identical weights. Specifically for large α and large β, changeover will occur slowly, for α close to 1 sensitivity to transient changes of weight and distance is reduced. Specifically, the time constant varies like $\ln(1/(\beta\alpha))$.

Although I have deployed generic radial-basis channel formulation inside motor systems, this extension to the framework, transparent from the point of view of the posting interface and only a small addition to the value representation, actually spans a space of action-selection algorithms, where the "action" selected is which (combination of) postings to back in the blended output. Most notably, the space of action selection algorithms spanned by choices of $(d_{\max}, \alpha, \beta)$ includes that of Blumberg ($d_{\max} = \infty$, α and β control the "level of interest" in the language of that model) but also includes models where multiple actions act

B. Blumberg, *Action-selection in hamsterdam: lessons from ethology*. Proceedings of the Third International Conference on Simulation of Adaptive Behavior, Brighton UK, 1994.

simultaneously ($d_{\max} \ll \sqrt{N}$, $\beta \approx 1$) and when the transition between actions is soft ($\alpha \approx 1$, $\beta \approx 2$).

Generic radial-basis channels with normalizedDistance(...) value representations are used in the implementation of "soft constraints" in the Fluid representation — multiple processes seek to maintain relationships between the positions and sizes of visual elements on a page. These over-constrained set may admit no global solution, yet one would like the layout to both explore the space of possible solutions and to keep close to layouts that are partial solutions. Here the "values" are modifications to individual positions and sizes, and the normalized distance between two displacement vectors V_1 and V_2 is:

$$\max\left(0, -\frac{(V_1 \cdot V_2)}{|V_1||V_2|}\right)$$

Such a metric is also used as the basis for the final output to the body of *network* to ensure that only compatible ways of satisfying its "imagined" physical constraint are attempted simultaneously. The framework allow competitive processes to be added to the generic radial-basis channels. From an authorship perspective, multiple postings can be added to the channel without fear of the system "breaking" under the strain of incompatible request — rather the channel will simply do the best that it can, and while doing so, explore many of its partial options.

This class of channel completes the discussion of the alternative "flow-control" strategies developed for *The Music Creatures* and deployed beyond. In the coroutine and resource frameworks we have, at last, a scripting technique for use in defining ordered actions and motor-programs, that is simple enough to admit a clear, imperative style, but strong enough to survive the uncertainties of interaction and complex system. In the generic radial-basis channels we have a ge-

neric framework for the creation of not only signal-manipulation networks, but "process-manipulation networks" — for the cases where a symbolic-level scripting approach does not "script" at the correct resolution. These channels are for the numeric representations, typically of agents' bodies, for the places where many systems, many lines of code, excerpt uncoordinated influences over central points. Finally, in the cases where neither a carefully computed blend, nor a strictly "action-level" action selection technique is appropriate, I have offered a hybrid — a generic radial-basis-channel-based "soft" action-selection strategy. Each of these techniques are general-purpose techniques for the creation of mid-points, loci of interactions inside complex systems.

5. Concluding remarks — *The Music Creatures'* aesthetics of agency

This chapter began with an overview of the lives of four different kinds of agents and concluded with a detailed description of the techniques constructed to realize their narratives. Despite the ordering of this chapter, the narratives did not completely precede or succeed the frameworks used to create them, but emerged as the techniques for the principles were constructed. This interplay between *narrative* and *technique* seems to be a fundamental emergent property of the agent-based installation that involves creatures possessing genuine life-spans populated by periods of learning and development. Specifically, this separates these works from those typically produced under the banner of artificial life Although oddly shaped and perhaps appearing closer to the machine than the animal, and despite their ultimate disposal and replacement on an almost regular time-scale in the gallery, these creatures are never the anonymous agents that artist's readings of artificial life seem to provoke. Rather they are, even in their limited ways, condensations of some experience particular to their installation space and to the people whose space they share.

Their seven-to-ten minute life-span takes them through a number of changes that, when multiplied by the other creatures, offers a balance of complexity and order that survives in its gallery setting. *The Music Creatures* populate a number of interaction time-scales up to this seven-minute mark: at the smallest level, their performance of sound is coupled to the small perturbations of their graphic lines; they respond promptly to sound made in the gallery; they maintain musical cells that last ten or twenty seconds; they offer visual and behavior changes that take place over a few minutes. On longer time-scales the whole installation undergoes cycles lasting hours, as material is perpetuated by creatures hearing other creatures. Change over each of these durations seems important for finding an interactivity that remains both direct and lasting.

This said, however well prepared the balancing strategies involved in the making of this work, they speak little to the problems of constructing a work that unfolds over days, weeks or months. It remains an enticing vista for future work to create agents such as *The Music Creatures* that exist on longer time-scales. We shall revisit this horizon briefly when we consider the future directions for my work

As a stratagem of indirection, *The Music Creatures* offer an extremely deferred model for the creation of music from the sounds in a gallery. Such *indirection* is a hallmark of the agent-based; the deferral to the agent marks its autonomy and indeed its very agency. In *Loops*, the flux and distribution of finite material and stylistic change is the object of indirection; when we reach *how long...*, it is the creation of animation material and meaning-bearing relationships that is indirectly specified in the agent formulations. Yet as we move away from the nearly ambient, or certainly minimalist musical aesthetic of *The Music Creatures* the complexities and the stakes of this indirection become much higher. As we shall see in *Loops Score* (the next musical work I present), by the time I am ready to approach the problems of music again, I will have had to reinvent the action-

selection architecture of the agent framework to increase the ways in which I can direct this indirection.

Additional aspects of *The Music Creatures* persist in my most recent work for dance. The most striking of these elements is the music creatures' sense of effort and intention. Each of these creatures has been, in many respects, set up to *fail*. *Exchange's* learning isn't quite strong enough to offer it a stable model of the results of executing a pose; it fails to account for the higher-order physical effects like momentum on the body, and the creature thus over- and under-shoots its target. *Line*'s internal representation of pose is left deliberately over-specified — leaving the creature to struggle from pose to pose. *Tile* is given an impossible task of grasping, indeed embodying, the rhythm of sonic material passed from other creatures that usually have little direct regard for rhythm. *Network* visualizes the frustration of an overly and overtly simple transition model of musical material. Thus, these creatures are in a constant state of imbalance and inadequacy, constructed not of perfectly functioning parts, but rather of pieces that fail and continually re-compensate in interesting ways in unpredictable environments — their own actions add to rather than negate their environment's unpredictability. As much as they are constructed with systems that offer long-term learning and the automatic calibration of their perceptual systems, this just serves to keep the creatures on their edge of complexity. That they have technologies with strongly open interfaces, waiting to blend and shift competing signals to their bodies is because they possess multiple and simultaneously conflicting intentions. Unlike animals such as birds or even characters such as *Dobie*, these creatures are not well-matched to their ecology or installation setting.

It need not have been like this, and it would have been simpler if it had not: I could have built stronger motor learning for *exchange* or given the agent access to the physics simulation so that it could "cheat"; *line* could simply have performed a linear "morph" between poses; *network* need not obey an imagined

physical constraint — it could simply twitch in time to its sound; *tile* might have acted to guarantee the rhythmic qualities of the colony.

However, this resulting aesthetics of failure and frustration, of effort and intention, stands in marked contrast to the smooth ease of much of computer graphics and interactive art, and the slick successes of the technology demonstration. That these agents manage to remain, purposefully, at a point of interactive disequilibrium is in itself the technical achievement of the work. I believe this to be one of the destinations of the agent-based, and such traces reappear in my works for dance theater. My agents there chase after all-too-inadequate and transient perceptions of movement and graphic form.

This section develops two techniques that have a wide applicability to a range of problems that agents face when trying to understand complex worlds. These techniques find a crucial place at the core of the perception systems of many of the agents discussed in this thesis — Loops Score, how long...? *and* 22.

Chapter 5 — The b-tracker framework & distance mapping

In the previously discussed decomposition of the agent — into "perception", "action" and "motor" systems — the perception system holds the privileged place as the point of entry of the external world into the agent. As we take the agent metaphor, or just an agent toolkit into new arenas — choreography, music, visual art — unsurprisingly, the perception systems of our agents require significant attention.

For the perceptual worlds inhabited by the agents are broad ranging: the agents in this thesis perceive the details and the gross aspects of human movement (motion-capture data), of human musical performance (data from an instrumented piano), of human speech (material from a narration) and of the sound of music itself (from live microphones in a gallery). One test of the agent metaphor is to organize these disparate domains without trying to unify them. Indeed in this chapter and those that follow we will see that agent metaphor offers the ability to construct a small set of principles and technologies that span this range — allowing, perhaps even provoking, new relationships between computer, movement, space and sound.

It is in this section that I articulate two such organizing perceptual frameworks — these are the "open forms" of the agent perception system, that help organize,

or indeed provoke, these relationships. It is also in this section that we can formulate most sharply the analysis and critique of the methodologies and results of the traditional digital artist's "mappings".

1. The perception system

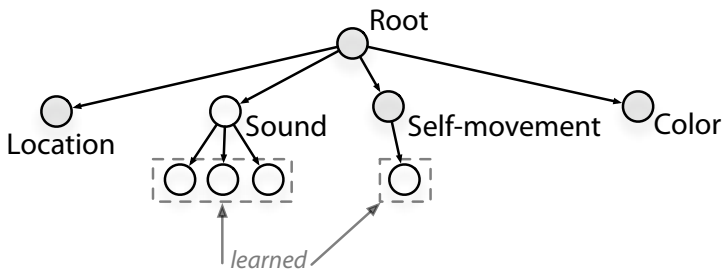


figure 53. A c5 agent "percept tree" classifies and extracts information from the perceptual world.

Often the perception system of a c43/c5 toolkit based creature follows a tree structure. The *percepts* at each node in this tree possess the ability to extract information from the rawest providers of sense information to the creature. This is a hierarchical decomposition of the *state* of the world, as sensed by the agent, into a set of categories, or at the very least, carefully treated responses to it.

Throughout this thesis there have been agents that have grown their percept trees to dynamically extend and tune the way that they decompose the world. In *The Music Creatures' exchange* agent, sub-models of the "sound" percept are dynamically added in response to hearing segmented audio in the gallery; in *network* these sub-models carry transition information that is used to create the body of the creature; in *tile* there is a population of rhythm models, not sonic models. *Loops*, for the matter of a little simplicity and computational efficiency amongst its numerous creatures has a fixed number of percepts looking for the "unexpected" in the colony's signaling environment. Finally, it is by dynamically monitoring and populating sub-levels of this hierarchical structure that *Dobie* is able to construct new states in the world from which to explore new (state,action) pairings, as described previously, *page 77*.

This hierarchical structure works well for these creatures and for a number of others, particularly in simple virtual worlds. But that this is a hierarchical *decomposition* of the state of the world needs to be made clear: a positive response from the "sound" percept of *Dobie*, the interactive, trainable dog indicates the

presence of sound of the environment; a positive response from the "black" percept of a wolf pup in *alphaWolf* indicates the presence of a another black pup in its field of vision; yet the co-activation of "sound" and "black" does not indicate the presence of a single "howling-black-wolf-object". Simply that there is howling and there is blackness somewhere.

Thus, we might say that "objects" in the world, should they exist at any level of description in the virtual environment, are broken up upon entry to the c43/c5 perception system. And that any perceptual fusion that occurs is up to the agent to perform.

This is only the first half of the perceptual fusion problem. Having completed this there exists, in some c5-based creatures, parallel perception trees that take fused packages of percept-tree response and constructs higher level recognizers that have categories for such things as "howling-black-wolf". However, even with these second order trees, there is a need to fuse the information from these perceived objects with objects previously perceived. And this *tracking* of objects is the other half of the perceptual fusion problem.

One can construct agents and interactive worlds that do not require any solution to any tracking or fusion problem: *Dobie*, for example, cared about a reward marker and the position of the interactor's avatar in the world, but cared not for any representation of their common origin; *alphaWolf* could in most cases, like many agent to agent perception problems, simply "cheat" and remember to package up all of the perceptions that came from the same particular agent together, and having done so, no ambiguity remained; the creatures of *Loops* never needed an object model, rather they sensed the average of all the creatures' effects on the surrounding signaling-fluid.

There are three possible reasons for not "cheating" or at least constructing an agent tool-kit such that cheating is not mandatory. The first is computational efficiency — complex perception for the 42 creatures of *Loops* was, at the time , out of the question. The second is what one might call perceptual honesty — that by demanding that our agent synthesize its own object-level models rather than obtaining them directly from the world, the mistakes that the creature makes concerning objects will be believable and, ultimately support an assignation, by an observer, of the agent of consistent knowledge and intentions. There is a substantial pay-off in realism and behavior for what might seem like substantial unnecessary busy-work.

Compelling evidence for this payoff is presented in D. Isla, *The Virtual Hippocampus: Spatial Common Sense for Synthetic Creatures* S.M. Thesis, MIT. 2001.

The third reason for this decision is that when one makes the connection between the virtual agent and the real world more porous, there are simply no object models to be found and the agent *must* synthesize its own. This is the problem we face in interactions less structured than in *Dobie* and *alphaWolf* and, indeed, in domains closer to sensing the real world directly such as robotics. Our agents, when they enter the context of dance theater, must synthesize and maintain models of moving dancers; when they enter a gallery they must construct and monitor models of sonic material; and when they listen to the performance of music they must create and track the location of the performance in the "world" of the score themselves. These things are simply not directly available from the "sensors" that we known how to make.

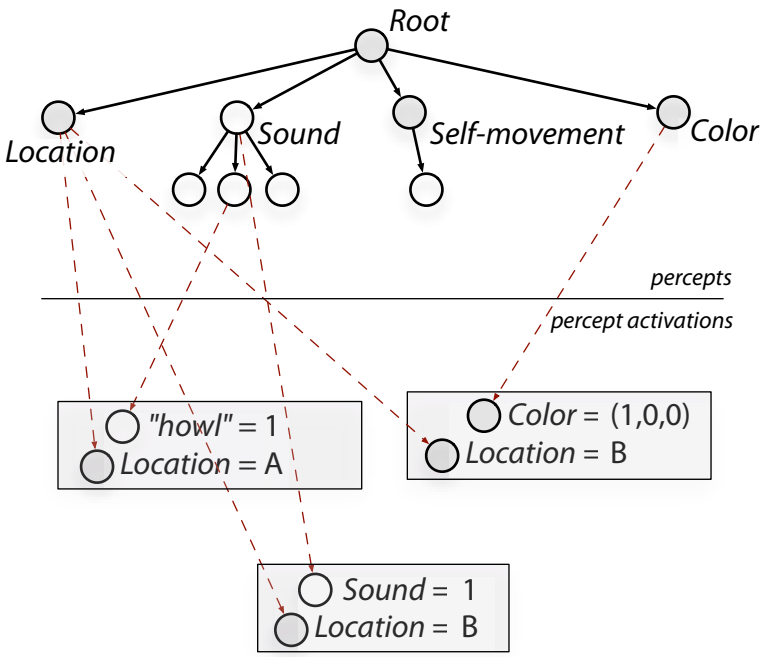
Thus in agent worlds that are more strongly coupled to our worlds, tracking and fusion problems are much less avoidable, since we seldom get an opportunity to control — as we wonder as artists from domain to domain — both the sensing and the subject being sensed.

Therefore throughout this work a broad range of algorithms are located in places that act as "perception systems" for the rest of the interactive artwork —

in particular those that build and sustain an a set of "object" hypotheses. I shall use as examples in this thesis: **score followers** — that match live musical performance to a pre-arranged score; **choreographic trackers** — that match live movement to open or closed versions of previous rehearsals; **rhythm finders** — that look for repeated gestures and form more complex ideas of the speed of a gesture; **movement trackers** — that rework synthesized movement into new sequences; **speech recognizers** — that identify snippets of sound as similar or different to previously important sounds.

2. The b-tracker "design pattern"

If the perception system is where the uncontrollable organization of the world repeatedly meets the internal author-able organization of the agent, a tracking problem occurs when the agent needs to form perceptual structures that exist longer than a single perceptual snapshot, where new information needs to be match and incorporated into older structures, when ongoing structures become repositories for learnt information, and when old structures require maintenance and extrapolation. Problems in this task range widely: short term, classic object persistence problems — is this object the same object that I saw some moments ago (from *Dobie*) ?; medium term support for ongoing actions — I have drawn a line from this object to this place, where now is this object (from *how long...?*) ?; long term memory problems — is this previously encountered dominant towards me (from *alphaWolf*) ?, is this sound like a previous sound (from *The Music Creatures*) ?



To develop a general, unifying, reusable framework for solving these tracking problems we decompose the issue into three main stages that take place over two pools of data (we'll see cases where more complicated structures are built up by layering trackers constructed in this fashion). The data pools are the **incoming elements** and the **ongoing models** and the stages are **incoming element fusion**, **incoming->ongoing prediction / match / update**, and **ongoing cleanup**. We'll first develop each stage of the framework before specifying these stages precisely.

figure 54. A hypothetical example consists of object tracking in a multi-object world. "Incoming elements" in this case are various continuous or categorized perceptions that are presumable located in the world.

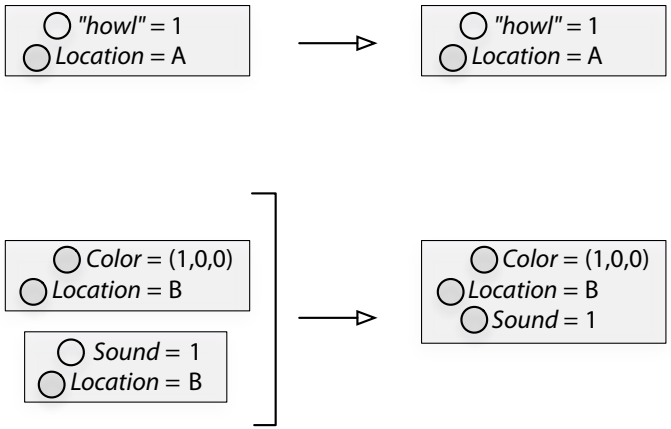


figure 55. These can be fused together on the basis that perceptions that come from the same location come from the same object.

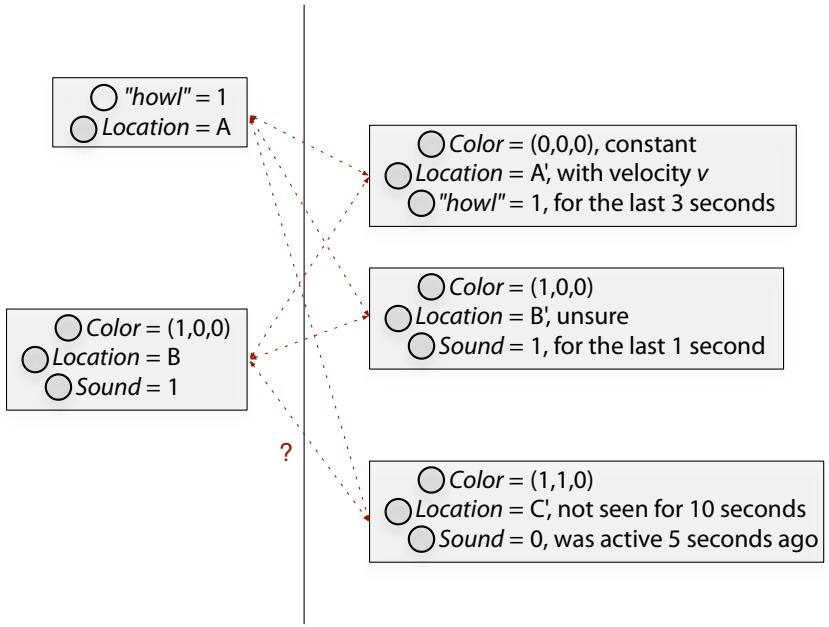


figure 56. Previously the agent has encountered a number of objects, we need to match these older objects with the new, fused sense data.

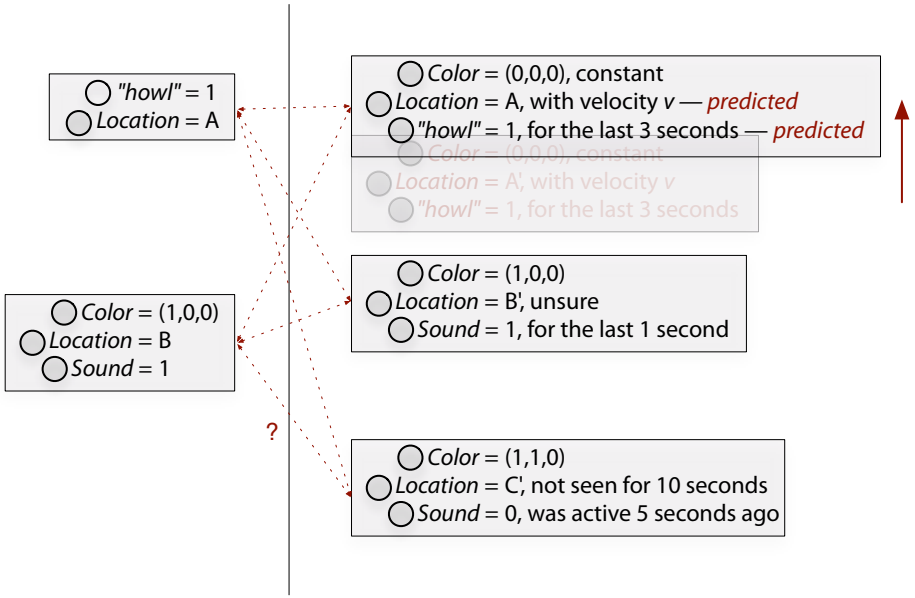


figure 57. In many cases it makes for a more robust perception system to match these new sense data with *predictions* of what these older objects should be now. This prediction process might change the apparent contents of the older-objects, or it may form and add new hypothesized descendants and nominate these as new ongoing models.

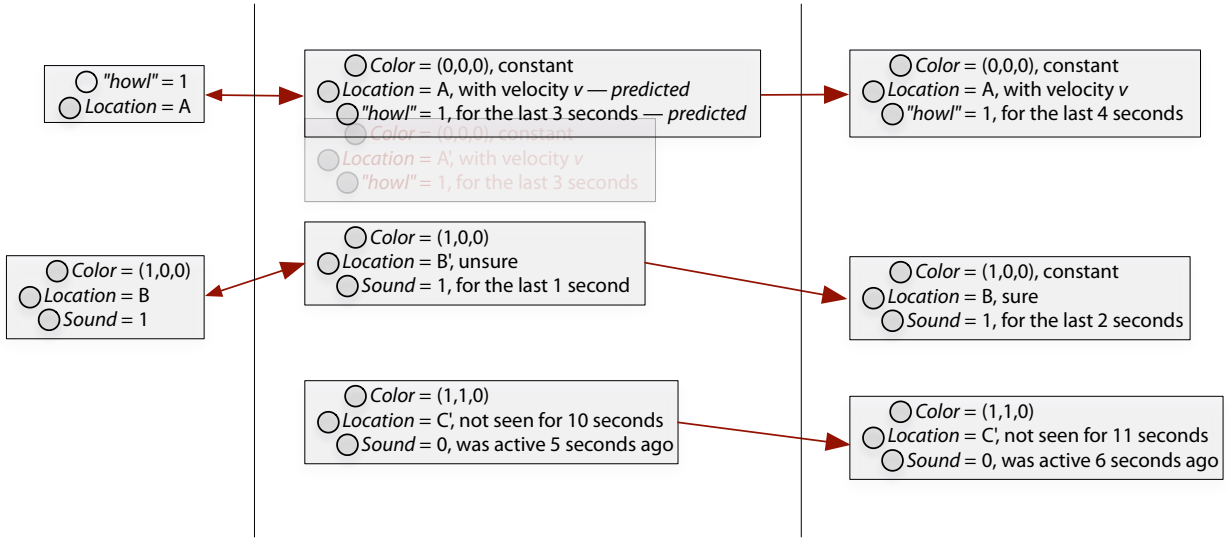


figure 58. In either case, once the new data has been **matched** with some of the older object-models, this new data is **merged** with the older object models. In particular, the agent's confidence in on object model will change, and the agent's confidence in the very existence of this ongoing object will change. Match algorithms vary in the deployment of the b-tracker framework — the two most commonly used are a greedy merge up until a certain threshold and a Hungarian assignment solver, *page 310*. We'll see a selection of simple algorithms below.

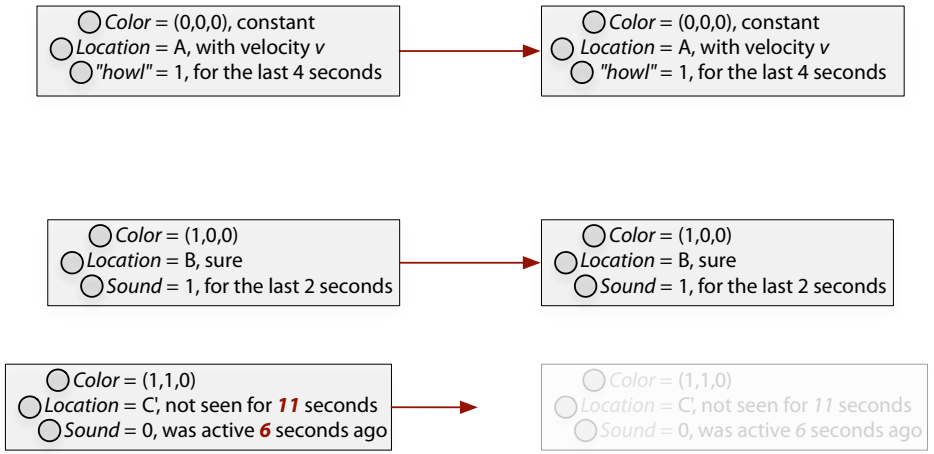


figure 59. Confidence in the existence of some objects might be so low that they are **culled**, stable confusion in the value of some data inside an object may result in ongoing model **fission** or two ongoing object models might be seen to be really the same object and **fused**.

Finally, we note that **top-down** influences may be exerted on the contents of this perceptual structure by injecting particular elements into the incoming set, or more interestingly, speculatively injecting objects into the ongoing model set (and waiting to see if any data "sticks" and increases the confidence associated with this model). In the above, diagrammed example, a creature might hypothesize the existence of an object in the world (perhaps food) with an unknown location. This ongoing model can be used as a placeholder object for action.

Of course this is all extremely general, but it's worth listing the data-structures and algorithms that need to be added to this framework, to provide a kind of template that we can fill in when we come to deploy this framework. To make concrete these stages, I'll sketch the structures and algorithms needed to construct four of the example uses of this framework that were deployed in artworks discussed in this thesis.

They are: a **Hough tracker** — given some movement (here, of dancers in *how long...?*, and the source video of *Imagery for Jeux Deux*) it tries to hypothesize and maintain straight lines that explain the movement or images, named after the Hough transform in image processing that finds straight lines in an image; **score follower** — given live performance data (notes played in *Jeux Deux*) this tracker works out where on a known musical score we currently are; **marker tracker** — given noisy, unlabeled, untracked motion-capture data tries to compute marker assignments, and smooth positions and velocities for these points while ignoring transient ghost markers; **dancer tracker** — given good marker data tries to cluster these locations into isolated areas and thus, without matching skeletons or using any other kinematic knowledge, tries to find clusters of points that are likely dancers.

The Hough Transform — **P.V.C. Hough**, *Machine Analysis of Bubble Chamber Pictures*, International Conference on High Energy Accelerators and Instrumentation, CERN, 1959.

The seminal score following work: **B. Vercoe, M. Puckette.** *Synthetic Rehearsal: Training the Synthetic Performer*. Proceedings of the 1985 International Computer Music Conference. San Francisco: Computer Music Association, 1985.

Imagery for Jeux Duex was made to accompiany composer Tod Machover's concerto for "hyperpiano" : **T. Machover**, *Jeux Duex for Hyperpiano and Orchestra*, Musical Score, Boosey & Hawkes, New York. 2005.

These problems are of roughly increasing complexity: the *Hough transform* is relatively solved problem, although I am unaware of any interest in solving it incrementally; many have written *score followers*, a fundamental if dangerous building block of interactive computer music for at least decade, and around for two, but our implementation gives us a few novel uses; in tracking *markers* it transpires that it is more important to have a solution based in this perceptual framework than it is to have a more accurate but proprietary black box solution; similarly with the *dancer tracker*, for a number of reasons we can exploit access to the specifics of this solution stratagem during the imagery for *how long...?*

Firstly, the data structures:

incoming element: has some, perhaps fragmentary, labeled piece of data associated with it. In a *Hough tracker* example this will be a short line segment, in a *score follower* example this will be a time-stamped note, in a *marker tracker* example this will be the position of a marker of unknown origin, in a *dancer tracker* this will a set of tracked marker positions.

The marker and dancer trackers are further discussed on page 305. A less direct tracker is found on page 368.

ongoing model: has space for a complete object model, together with enough element history, use history and merge history to participate in this agent. *score follower* — hypothesized score position and tempo; *marker tracker* — Kalman filter model of marker position, velocity and acceleration; *dancer tracker* — k-means-based clusterer of marker positions.

And at each stage there are algorithms that might be provided:

incoming element fusion: spots that some elements should be pieced together into intermediate packages of data in order to make the matching easier. *Hough tracker* — successive line elements that are too short to be reliable are pieced together into longer elements with a more definite direction; *score follower* — no fusion takes place; *marker tracker* — markers that are too close together are merged; *dancer tracker* — no fusion takes place.

ongoing model prediction: prepares the outgoing models for matching by speculatively, and reversibly, updating them with the current "time". *Hough tracker* — no prediction; *score follower* — predicts where we would be in the score right now if we continued at the hypothesized tempo; *marker tracker* — the kalman filter prediction cycle; *dancer tracker* — the k-means update cycle on the most recent data.

incoming → ongoing match: matches the incoming elements and element packages with the ongoing models often using a distance metric between elements and models. *Hough tracker* — nearest neighbor search using a line-segment to line-segment distance; *score follower* — all incoming data "matches" all models, every hypothesized score position and tempo has to explain the incoming notes; *marker tracker* — an implementation of the "Hungarian algorithm" linear programming method produces unique parings between markers and predicted marker positions, some markers

will be new, some ongoing models will be unmatched; *dancer tracker* — all incoming data "matches" all models, all markers have to be explained by each hypothesized dancer configuration.

incoming → ongoing merge: having made a match (or made no match) the ongoing model needs to be updated. Often a global confidence score is associated with a model. *Hough tracker* — a line segment model is rotated and translated toward the new line segment data; *score follower* — each matched model builds some good hypotheses as to what note in the score that incoming note corresponds to, and what that does to the current tempo; *marker tracker* — marker positions are added to the ongoing kalman filter as an observation stage; *dancer tracker* — potential k-means cluster fission and fusion are evaluated in the light of the new data; fitting scores are calculated.

ongoing model cleanup: culls, fuses, fissions and injects ongoing models into the pool. *Hough tracker* — poorly scoring models are dropped, good scoring models duplicated as the number of hypotheses are kept near a particular target number; *score follower* — poor hypotheses are dropped, nearly identical hypotheses are merged, good hypotheses that have multiple explanations of the most recent data split; *marker tracker* — poorly performing, lost markers are dropped; *dancer tracker* — poor models are dropped, good models that wish to offer versions of themselves with greater or fewer active clusters (dancers) do so.

top-down influences: That systems typically post-perception can offer top-down influences on the perception system is also of considerable interest, particularly in a space where an agent might commit to acting upon a model: in a *hough-tracker* — we might need to maintain lines that have been drawn or are being drawn or are the lattice for an ongoing movement; in a *score follower* — we might have alternative means for guessing

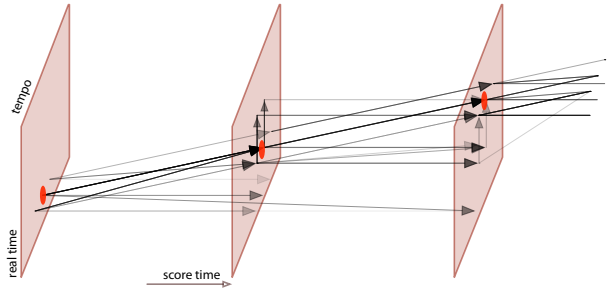


figure 60. In a score follower the ongoing model is a pair (score position, tempo). The b-tracker population of models attempt to predict the next notes and compete to explain the data as it arrives in a live setting.

the current position, during rehearsal or performance; in a *marker-* or *dancer-tracker* the agent again may already have started to act upon a marker and require that such a position is maintained and updated.

In its chameleon-like configurations the b-tracker framework relates to other work that has been used in the fields that border on that of making agents' perception systems. Maintaining a population of likely hypotheses (ongoing models) as one scans some data piece by piece (incoming elements) is very similar to a beam-search, a general purpose heuristic search technique used, for example, in planners. A beam search is a different (a more general, but often less complete) but related way of solving problems typically solved by dynamic programming.

And much has been written about dynamic programming as a framework for understanding, or at least building, musical perception, for example the work of David Temperly and Roger Dannenberg. But a careful reading of this work will show that once the initial excitement surrounding the dynamic programming trick — the spectacular apparent efficiency of dynamic programming over complete search, converting exponential time algorithms into polynomial time — it becomes increasingly hard to formulate perceptual frameworks inside the limits of the dynamic programing *per se*. Indeed, as Temperly is forced to add optimizations in his monograph on the use of dynamic programming in music and look to fusing dynamic programming processes together it looks more and more like heuristic search.

R. Dannenberg, *Dynamic Programming for Interactive Music Systems*, in Readings in Music and Artificial Intelligence, **E. R. Miranda**, (ed.), Contemporary Music Studies series, Vol. 20, Harwood Academic Publishers, 2000.

D. Temperly, *The Cognition of Basic Musical Structures*, MIT Press, 2001.

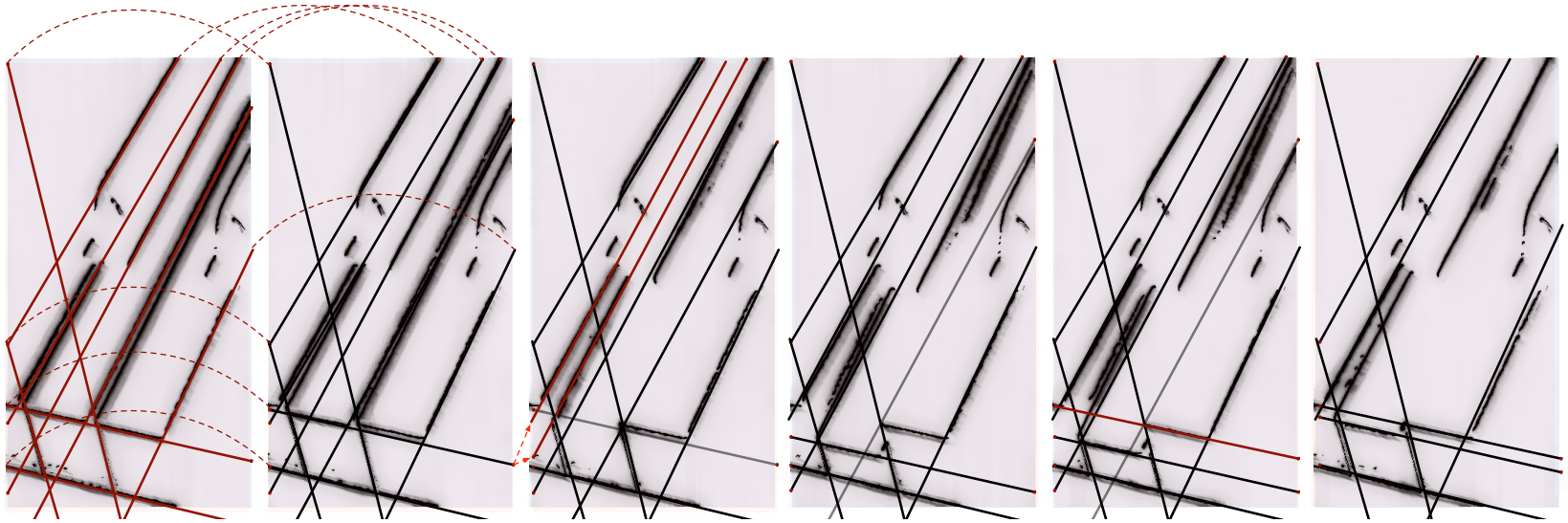


figure 61. In *Imagery for Jeux Deux* video of piano key depresses was integrated into the piece's network of points and lines by annotating the video with straight lines. These lines, identified by a Hough transform on each video frame were then tracked using the b-tracker framework. This yields lines that follow follow the underlying animation of the key press and can be connected to other material in the work.

Especially in the case of the marker tracker there is a clear a relationship between the b-tracker as described here with the Conditional Density Propagation algorithms first described in:

M. Isard and **A. Blake,** CONDENSATION — *conditional density propagation for visual tracking*. International Journal Computer Vision, 29, 1, 1998

A. J. Viterbi. *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*. IEEE Transactions on Information Theory 13(2):260–267, April 1967.

With application to hidden Markov models, **L. R. Rabiner.** *A tutorial on hidden Markov models and selected applications in speech recognition*. Proceedings of the IEEE 77(2):257–286, February 1989.

Frames: **M. Minsky,** *A Framework for Representing Knowledge*. MIT AI Lab, Memo 360, June 1974.

Of course, the most important example of dynamic programming is arguably be the Viterbi "search" of hidden Markov models. Indeed, we'll see a gesture recognition task posed in the b-tracker framework, *page 315*. A population of simple predicting trackers that explain incoming data looks a lot like the condensation framework used in computer-vision tracking problems. Further afield, the top-down injection of ongoing models reminds one of symbolic AI's frame structure — when actions want to hypothesize the existence, perhaps of an hidden object, they might instantiate an ongoing model that will act as both a repository for information, should this object become, visible and as a token for other actions to use (for example to provoke and guide search behavior) based on the uncertainty of various "slots" in that "frame".

It is not then that the b-tracker framework necessarily opens up previously in-

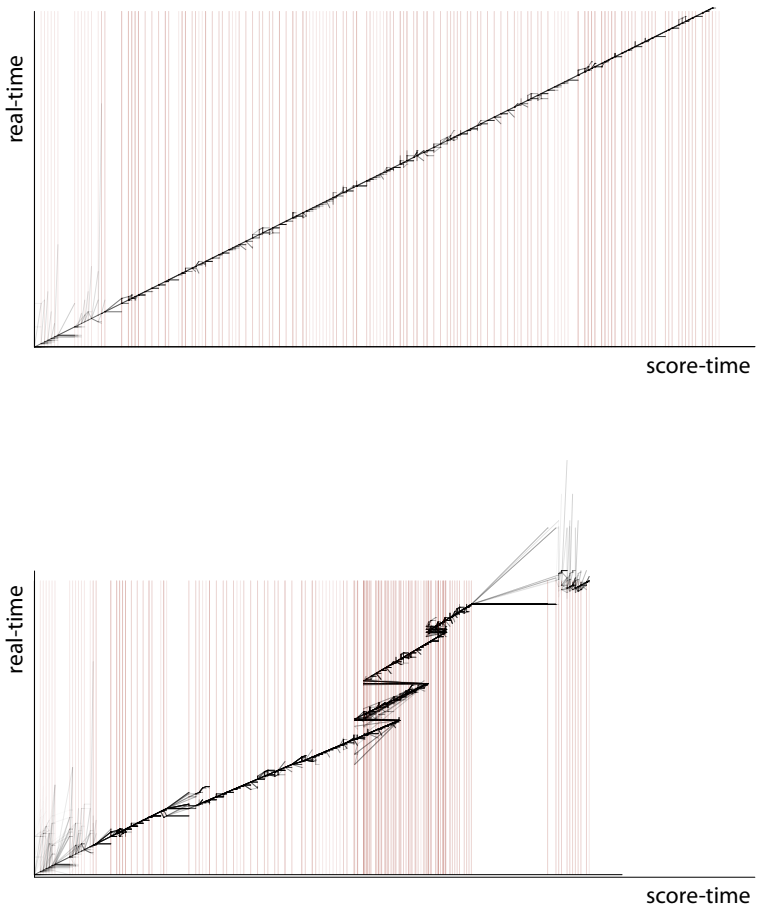


figure 62. The b-tracker framework allows a robust score follower to be quickly created from reusable parts. The above diagram shows a tracking of score position using a synthetic performance of Tod Machover's *Jeux Deux* that has been artificially corrupted with 10% of the notes played wrongly. Despite the noise the tempo is clearly constant and accurate. The lower diagram shows a snapshot from a rehearsal — where at one point a few measures of music are repeated.

tractable problem domains — in terms of analysis — but rather that is is a single core framework for thinking about, and implementing, many kinds of things that intelligent systems end up needing to do. The motivation for and the success of this framework comes from two places: firstly that its broad applicability will allow a great many agent perception systems to be quickly and robustly considered and assembled, exploiting a common set of code and visualization tools; secondly, that as a way of allowing an agent to see the world it is not only an open or "white" box but better — it offers the right kind of openness and the right sort of partial inner structure for other systems to communicate with, on the level not just of "output" or "results" but of inner dynamics as well.

It is worth pausing to reflect upon the openness of this structure compared to other approaches — since the b-tracker framework offers a concrete way to talk about some perception problems, its i worth taking stock and comparing this framework to other "perceptual frameworks" used in digital art. While some incredibly well-written analysis might offer a single "answer", a single "perception" of, say, our current position in a musical score, or the position of a dancer on a stage, the b-tracker offers a small, trackable population of scored hypotheses with histories and uncertainties. Which would we rather work with as artists?

I suggest that tracking problems occur in exactly the places where a mapping approach would fail to gain traction. In lieu of the perfect answer — the exact score position, the precise dynamics of the stage, a stock function-like transformation offers to *flatten* the information present in the perceptual world into a single quantity, which in turn allows subsequent simple transformation. It makes no difference if this quantity is of high dimension or a single number, the function-like core of mapping and transformation promotes a *constant* data dimension.

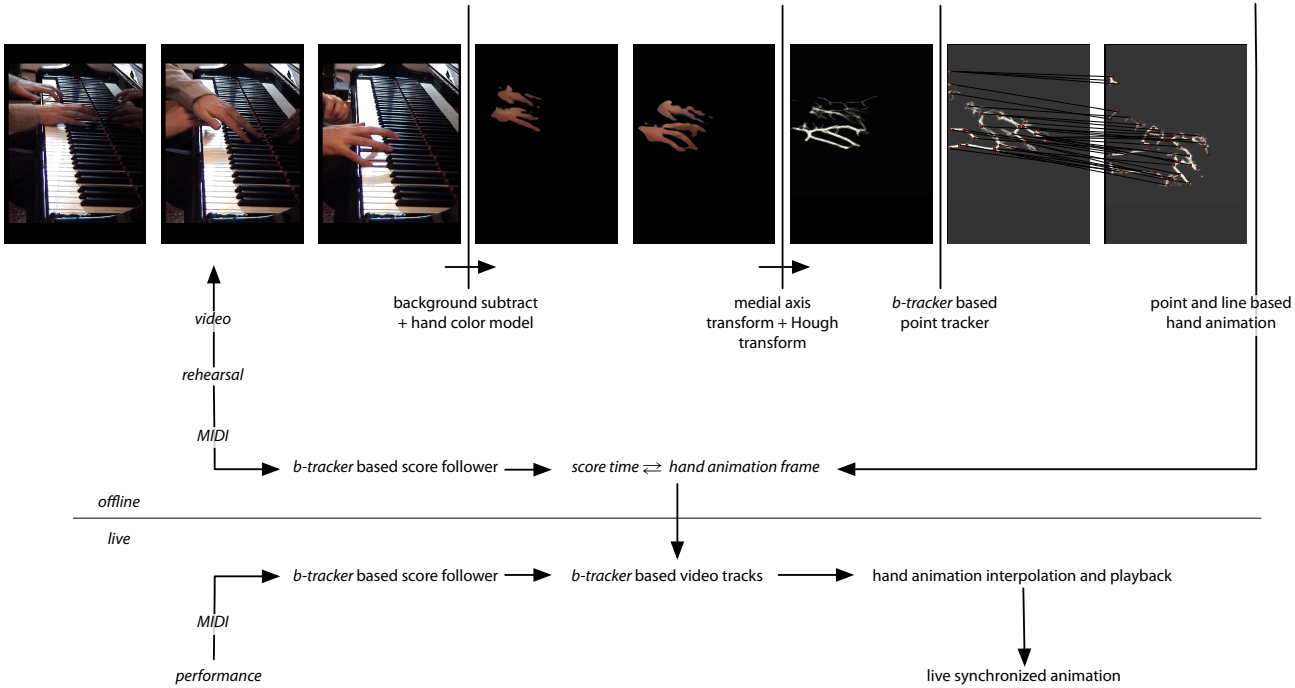


figure 63. *Imagery for Jeux Deux* couples two b-trackers together to synchronize animation (derived from video footage of a rehearsal) to a live performance. First, the note data (MIDI) of the rehearsal is converted to a score-time (in quarter notes) using a score follower. This relationship can be inverted to provide a video time-code and playback rate per quarter note of the music. Then the score is tracker live. As the live b-tracker follows the score, animation material from the rehearsal tracks each score hypothesis, fading in and out with the ongoing model confidences.

Tracking problems occur in places where simple averages may capture nothing, where what is being perceived is intrinsically multi-modal (in a statistical, rather than media sense). The *average* score position when there is uncertainty over whether a performer is repeating a section of music is worse than most other guesses one could make; the *average* position of a dancer when there is uncertainty over whether there is one or two dancers can be arbitrarily poor. A motion capture marker-mean-position might be so noisy as to be useless (consider the case where a dancer lingers on the edge of the motion capture volume), and might be arbitrarily far away from the dancers (consider the case of two, opposing, lingering dancers). When confronted with such "noise" mapping tool-

kits offer to bury these measurements in increasing levels of filtration. But if there is insufficient information present in this signal to start with, if the perceptual world of the autonomous artwork is already aliased so severely, to look for the solution by eliminating even more information from this signal seems perverse, *page 307*.

In some cases an agent can work with these potential hypotheses without flattening them in any way: in *Imagery for Jeux Deux*, multiple video streams synchronized with individual score-tracking hypotheses fade in and out with the confidences of these models — the resulting perception is of a continuously synchronized visual performance, that waxes and wanes with the certainty of the tracking, which in turn is affected, on a different level, by the very clarity of the music at the point. Even simpler, lines drawn to moving points on the stage of *how long...?* commit the b-tracker to maintaining an active hypothesis for the end marker while the line exists, allowing the linear form to unfold gesturally, rather than appearing in a single frame.

In other cases, of course, an agent has to pick one hypothesis and stay with it. What constitutes a good decision-making technique ? — an action selection strategy. We have already seen that such algorithms are judged by their relevance (they pick good models), their coherence (they stay with these models long enough and no longer).

By using an action-selection framework our agents can then become extremely conservative concerning the deletion of information. And this aspect of this work — and it really is an aspect of the agent approach that we are building here — is reflected directly in the artwork. It is the difference between being able to stably form a dancer-like cluster of points, maintaining a top-down influence over that cluster and performing a visual operation on their position — one that is coherent over seconds or minutes — versus being able to "visual-

Of course, this flavor of explicit *representation* — our population of hypotheses about the world — appears to go against the spirit at least of the early agent-based research in the field. However, we can avoid some of the dissonance with our historical narrative by realizing that the actions that our agents make are not dictated by or dedicated towards maintaining this model of the world, nor is this model a singular and totalizing end in itself.

ize" the mean of all the markers on the stage. The former is tentative but specific, possibly fleeting, but a particular segmentation of the perceptual world; the latter is an affirmative, constant, broad averaging over the whole environment. The former meets the world on its own terms; the latter slices the world in a predefined manner regardless of the specifics of what it happens to slice.

The b-tracker framework offers a fundamentally different authorial position on the perceptual world occupied by an interactive artwork. Constant dimensional, pre-arranged slices of a perceptual world that do not segment the perceptual world are of limited use in complex worlds; in fact this might be the very hallmark of complexity itself. By using, instead, these techniques, our agents are more open to the interactive possibilities of those complexities.

3. The distance mapping algorithm

This is not to say that the action *selection* begins and ends the *use* of perceptual information —the properties of the thing perceived of course leak (or perhaps are even mapped) into the action that demonstrates that the perception has occurred (move *toward* an object; reconfigure *toward* a new musical measure, begin breaking down a "scene" of the choreography over *there*, manipulate the temporal flow of a graphical "score" based on *this moment* occurring now). But an agent framework allows the complexity, the multi- and variable dimensionality of the world, into the agents that exist inside it.

This said, we have seen problems and solutions to problems within this agent perspective that have the flavor of mapping to them — the long-term learning database of *The Music Creatures*, *page 143*, learns simple scalings from one domain to another; the motor learning of music creatures makes small self-organizing maps in order to understand the effects of its own motor control. Of course these are "small numbers" inside large systems — rather than large sys-

tems made up of small numbers. But what makes these different from the excruciatingly hand-made signal manipulations of classical mapping is the technologies that surround these magic numbers and transformations — technologies that allow these numbers to be indirectly and automatically set by a more "human" description or process of what those numbers ought to be.

Our simple, often one-dimensional self-organizing maps can learn a scaling from one domain into a particular, fixed range, while removing some of the static statistical features of the input domain, decoupling the *consumer* of this signal from some of the specifics of the *producer*. This technique, as much as it is useful, does not care about the temporal qualities of the signal. It does nothing with the temporal statistics of the signal (indeed, the first step in the learning algorithm for these maps is to randomize the order of and often down-sample the input signal) and it gives no interesting control over how the the output distribution changes. Perhaps, and especially as we move towards sharing a time and space with live dance, there is a role for indirectly specified maps that do propagate some temporal information.

As work for *how long...?* and *22* progressed it became increasingly apparent that we needed to build our own layered structure of perceptions of the movements of the dancers, stacking primitive upon primitive perhaps with parameters that could be quickly learned or reconfigured, rather than approaching the problem armed solely with a detailed foreknowledge of the choreography — foreknowledge that was impossible to obtain given the choreographic work schedule of *how long...?* and with the working practices and improvisatory nature of 22. Early on, as we began to sharpen the marker- and dancer-trackers, we began to look at other simple measures that we could derive from of, a set of markers moving in space that would be robust to both noise and choreographic decisions. The ultimate payoff for this work comes with a description of the works themselves, but the approach is so general as to merit a separate discussion here.

In 22 there are several properties of the imagery that become, at times, connected to motion on the stage — the obvious way to do this is to couple speed (of dancer) to speed (of playback of video, of movement of infinite lines). In *how long...?* there was a perceptible and yet ungraspable, unlocatable, vanishing rhythm to the movement that stood in defiance of the frame-rate and resolution of the motion-capture cameras. Immediately clear was that speed as "distance divided by time", as one would write it in high school, captured little of the motion of modern dance;with its dizzying curves and recursive foldings back on itself, this rhythm in Brown's motion simply was not to be found in such trivial coarse velocities.

Consider the problem, then, of *automatically* mapping the movement of a dancer to the movement of the virtual animation. Specifically, mapping the movement of a motion-capture marker-set to the movement through a particular, pre-made animation. We would like to specify as little foreknowledge to this "motion-scrubbing" problem as possible — because everything might be different in the next rehearsal or performance — and yet relate the complex temporal qualities of the dancer to the complex temporal qualities of the pre-made animation.

A simpler sub-problem is the mapping of a marker set animation to the "movement" of a single number. More precisely, can we find the motion of a single scalar quantity that most succinctly captures the qualities of that marker-set movement? This problem leads to the classic definition of *multi-dimensional scaling problem*, which is in this case equivalent to computing the principle component or leading eigenvector of the self-distance matrix. All multi-dimensional scaling techniques seek to find low dimensional spaces to embed high dimensional data-points such that **distance relationships** between the points are retained in the lower dimensional space. We could treat such an embedding algorithm as a black box, simply reading the literature and implementing one of the well known versions of the technique. However, a reinterpretation of the principle component analysis that underlies this technique will provide us with an algorithmic formulation that is more efficient for our purposes and much more flexible.

For a brief overview of multi-dimensional scaling: **J. B. Kruskal**, and **M. Wish.** *Multi-dimensional Scaling*. Sage Publications. Beverly Hills. CA. 1977

Our approach here is to take the **input signal** (the marker movement) I_t and a **distance-metric** (that gives a distance from any particular configuration of markers to any other) $\mathrm{d}(I_{t_0}, I_{t_1})$ over a range of time $t = a \ldots b$ sampled by N samples. This distance metric is the foreknowledge that we add. We can then compute the $N \times N$ matrix of distances D^I. The goal is to transfer this distance matrix over to a single output scalar quantity that is also defined over the interval $t = a \ldots b$ and also sampled by N samples. We do this by iteratively making the $N \times N$ matrix of self-distances for the output signal D^O increasingly like that of D^I. For scalars, where the distance metric is simply $\|x - y\|$ the iteration is:

for each element O_t; $t = a \ldots b$,

$$O_t \leftarrow O_t + \alpha \sum_{q \neq t} \left[(O_t - O_q) \cdot \left(D^I_{qt} / D^O_{qt} - 1 \right) \right]$$

By starting with a motion signal that shows some variation (even if it is simply some noise added to the signal) and repeatedly applying the above equation to each element of the signal N for some small α we will decrease the difference between D^I and D^O.

The above formulation of the problem is equivalent to an algorithm known as the power-method of finding the largest eigenvector of a matrix. For non-degenerate starting signal, it is extremely likely to converge and converges with a rate proportional to $\alpha\lambda_1/\lambda_2$ — the ratio of the first two eigenvalues of the distance matrix.

For a description of the power-method, and its convergence properties: **G H. Golub** and **C F. Van Loan,** *Matrix computations, second edition*, The Johns Hopkins University Press, 1989.

also **G. H. Golub, P. Comon,** *Tracking a few extreme singular values and vectors in signal processing* Proceedings of the IEEE Volume 78, Issue 8, Aug., 1990.

However, our almost pictorial interpretation given here seems to offer opportunities for special control that the text-book leading-eigenvector formulation does not possess:

A related, more principled, but less general iterative work is: **T. Morita, T. Kanade,** *A Sequential Factorization Method for Recovering Shape and Motion from Image Streams*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19 (8), 1997.

Firstly, we might want to find a **constrained solution** where a few particular O_n are fixed, or are less able to move — this is an opportunity for "top-down" control over the answer, perhaps the agent has already committed its body to some part of the solution and thus this part of our re-scaling cannot change.

Secondly it's easy to see how to **iteratively update** this system when a new piece of data arrives — turning an iterative algorithm into an incremental one — we simply need to pick a single new output scalar that minimizes:

$$O_{N+1} \leftarrow \arg\min_y \sum_{n=1\ldots N} \left| |y - O_n| - D^I_{n,N+1} \right|$$

Since O_N is a good starting guess for O_{N+1}, this is an $O(N)$ operation rather than $O(N^2)$ and also allows us to shape the preference for output distribution.

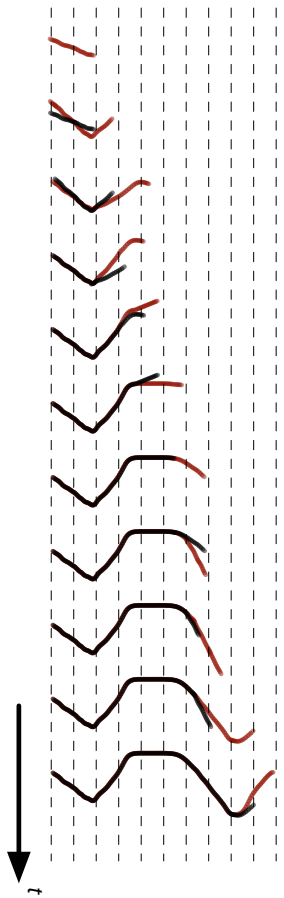


figure 64.
The distance mapping algorithm can be executed iteratively. Here, the solution to the problem given on the next page, *figure 66*, is slowly developed over time.

We can go further than this and, introducing a little latency in the output mapping, update not just one but the last few recent output samples, increasing the stiffness of this update as we go further back in time. This might reflect an increasing commitment to older values, perhaps because some other system has acted upon them.

Thirdly, and perhaps most interestingly, we can redefine the above picture in more abstract terms and operate on **non-scalar output spaces.** We define an operation blendedDistanceNorm(O_a O_b , d, α) that takes two elements of O — O_a and O_b and returns a version of O_b , O'_b such that the distance between O'_b and O_a is α closer to d . Of course, for such a general function we can say much less about the global convergence properties of this algorithm, but if this function always does what it claims to do, and never goes backwards, we know that we will always converge to a solution, if not the optimal solution. Finding optimality, in the face of many local solutions, is within the purview of the b-tracker framework which will meet this representation shortly.

We are now in a position to define a class of automatic, iterative, temporally aware mapping algorithms that are defined in indirect terms: specifically an **input distance metric**, and **output distance metric** and an **output blend function.** In certain cases where the input space has a particular topology we can automatically generate an input distance metric using the self-organizing map techniques previously described; otherwise there usually remains as a degree of freedom to pick a scaling between the distances of one space and the other.

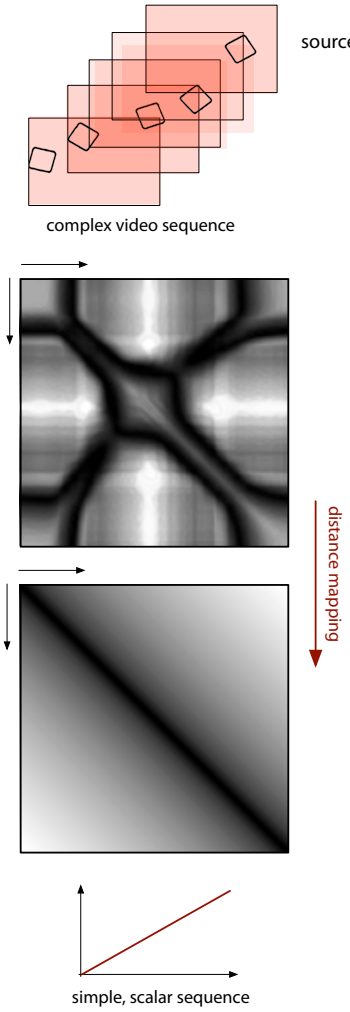


figure 65. The distance mapping algorithm takes one matrix of distances (here formed from a complex video sequence) and tries to the distance matrix of some other signal, in some other mathematical space, look just like it, by modifying that signal.

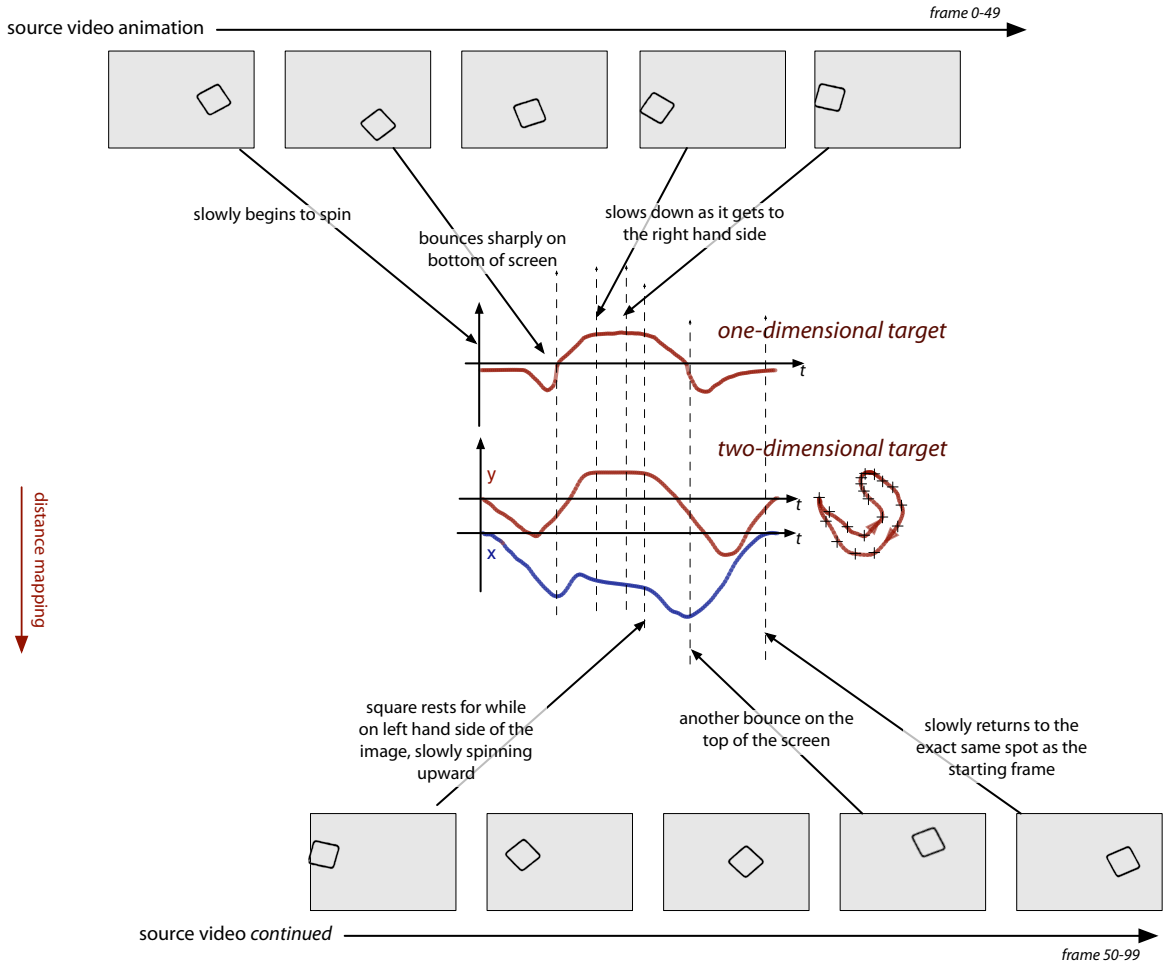


figure 66. Automatically mapping the video sequence of a rotating, bouncing square to a scalar or a 2-vector captures many of the aspects of the source video — the general symmetry, the slight asymmetry, the pause in the middle, the "rebound" upon the bounce. Further distance mapping analysis of the residual distance matrix yields a noisy, but informative, oscillatory signal that corresponds to the wavelength of the square's rotation.

We also note, in passing, that we have seen other representations that can satisfy these criteria — pose-graph motor systems have distance metrics and if needs be we can compute a constrained blendedDistanceNorm() function by allowing movement along graph edges; generic radial-basis channels' value representation can be used to formulate a blendedDistanceNorm() function if supplied with a distance-metric (the exact same interface is required for the competitive basis channels, *page 169*).

The motion-scrubbing solution

Thus we can formulate a solution to the motion-scrubbing problem with almost no tuning on our part. We need: a **distance metric** for motion capture marker data — for the case of tracked data, *page 305* this is a trivial sum of squared distances, for untracked, data we use the Hausdorff distance metric; a **distance metric** on the output space — the sum of squared vertex-position differences for the vertex animation data suffices; and a **blendedDistanceNorm**() function — we use an iterative algorithm that searches forwards or backwards in time along the fixed animation, is generic to any interpolatable time series (including the pose-graph representations) that already has a distance metric. In this case the inter-frame distances of the animation that the output distance calculation uses can be pre-computed and the whole iterative system runs in real time with a negligible computation burden; the core iteration is easily accelerated by modern vector processing techniques.

For a formal definition of the Hausdorff distance: **E. W. Weisstein.** *Hausdorff Measure*. From *MathWorld—A Wolfram Web Resource*.

http://mathworld.wolfram.com/HausdorffMeasure.html

Unlike the simple approach of mapping the speed (as in distance divided by time) of dance to the speed (as in frames per second) of video this approach has the following advantages:

it allows **backward motion** by the performer to change the direction of

movement of the video and similarly repetitive movement by the dancer repeats segments of video all the way out to the duration of the time-series windows. No amount of filtering a measured "dancer speed" could ever achieve the same effect for the information is simply not present in the instantaneous velocity of the dancer, but in the relationship to the body now with the body long past;

it is **less sensitive to noise** on the input signal than a first derivative calculation would, be while adding no additional latency;

it is **bidirectional** — the distance metric of the target representation also factors into how quickly we move through it. Rather than just playing out at a variable rate, should the video do something like "reverse" direction, our motion-scrubbing performer will have a much harder time keeping the video moving forward.

Mapping the moving of the dancer to the movement of the fiducial, "infinite" lines that mark the scrim in *22* is accomplished in a similar way — only here the distance-mapping algorithm is located inside a generic radial-basis channel and its output is blended with the influences of nearby geometry on those lines and the line's own momentum. It is useful to use the distance-mapping algorithm within such a process; in fact we can fade the distance mapping layer in and out depending on how good a job it finds itself doing at matching the output distance matrix with the input. This allows not just a mapping to occur, but the system to seize correspondences as the opportunity arises.

Finally, we can use incremental mappings into low=dimensional spaces as an input to higher level perceptual primitives. In particular we can open up the workings of the distance mapping algorithm to the view of b-tracker hypotheses that track a range of output signal trajectories, scoring them on how well their distance matrices correspond to the target input, creating new hypotheses by

perturbing the output signal — helping the system as a whole out of local minima caused by degeneracies of the output space. Alternatively, we can track individual extrema as the move through low-dimensional output signals on the basis that the input signals correspond to at least transiently interesting "poses". We shall see an example of these very algorithms in the *memory score* agent of *how long...?, page 368.*

Concluding remarks

In summarizing the distance mapping approach we might try to find a little space between it and more conventional slices through perceptual worlds. Although the distance "mapping" algorithm began as a direct approach to the mapping problem, with a goal of finding new ways of explicitly yet indirectly specifying mappings, in this formulation information is *compressed* into low dimensional signals but not necessarily *deleted*.

Indeed, we might note, in searching for a definition that separates the "analytical" of mapping from the analytical of more broadly used computer scientific representations is this compression. If we think of the analytical representations that have truly widespread use in synthetic arenas — for example, in computer music we have the short-time Fourier transform, linear predictive coding, the wavelet transform, or even just the radial-basis functions and neural networks of some of the more advanced mapping techniques — each of these representations began or were quickly adapted for the use of signal compression and reconstruction. Multi-dimensional scaling, related as it is to adaptive vector quantization and self-organizing maps, also shares this compression-aspect. And it is this, not any comparative complexity, that indicates a separation between such techniques and the simple averages and global measurements that visual, interactive artists have typically tended to gravitate toward.

This is, of course, a wide cross-section through computer music, for an introduction to most of these techniques as they apply to computer music, **C. Roads**, *The Computer Music Tutorial*. MIT Press, 1996.

For wavelets: **G. Evangelista**, *Flexible Wavelets for Music Signal Processing*. Journal of New Music Research, 30 (1). 2001.

Taken together — the broad, generic and adaptable b-tracker framework and the very specific, analytic distance-mapping technique have a number of features of critical importance to the working artist. They are general purpose: and thus lasting, and worth investing tools and visualizations on; they are generic: armed with these techniques I can build and test algorithms using simple data-sets, perhaps scalars, before exposing agents to the complexities, of, say modern dance; and they are open — they offer structure for thinking about perceptual problems and a variety of detail levels for interaction. These are the two approaches that allow my agents to enter into a whole variety of perceptual worlds, quickly and adaptively.

This section takes us from the works that use the c5 agent-toolkit to a new toolkit, called Diagram. *It introduces the first artwork to exploit this framework —* Loops Score, *the music for* Loops. *Diagram is a loose and interacting set of extensions to c5 that are designed to ameliorate the apparently persistent problems that artists employing complex agents face. In the technical discussion that follows, I respond to my previous critique of c5 and its use in alphaWolf.* Diagram *is also the set of technologies that will lead to* how long....

Chapter 6 — The Diagram framework & *Loops Score*

In the previous description of the complex, multi-programmer project *alphaWolf*, we discovered a number of inefficiencies in the way that it was assembled. While some of these problems may have stemmed from what one might call "impedance mismatches" between the components used to assemble *alphaWolf*, it seems more likely that the failings and weaknesses of this large assembly ideas from the c5 agent toolkit used for that work were more infrastructural than tied to any particular algorithm or representation. Both *Loops* and *The Music Creatures* dodged or postponed many of the issues identified — simply because of the size or goals of the agents involved, or the technologies deployed around their creation.

For the two works for dance theater that close this thesis — *how long...* and *22* — we had the great fortune of having two and a half years of notice before premiering the works. I could have spent all of this time constructing new fragments — new action-selection techniques, new classifiers for the perception system and new representations for pose-graph motor system. Instead I took

some time to consider how these fragments were being assembled and the "glue" systems that hold the other systems together.

1. Complex assemblages – the inversion (of the inversion) of control

The scope of these "glue systems" is both a little broader that the influential "design patterns" of software engineering and substantially narrower than a complete, integrated, academic AI system. Broader —for they are more concrete and more multiply instantiated than the abstract design pattern; Narrower — for they make no claim to be a complete or even partial solution to any particular AI *world* or *problem domain* by themselves.

The inability to draw a stable, hierarchical diagram of either control, instantiation, execution ordering or signal flow has been seen before in both published descriptions of hypothetical AI systems and throughout software engineering. We have seen it, in miniature, in our analysis of the source files of *alphaWolf*, *page 100*; we have seen it in the odd inversion of motor-system outside the colony of *Loops, page 117*; and we have seen it in the ad hoc reinterpretations of the perception / action / motor decompositions of *The Music Creatures*.

Indeed, historically, the very idea of the software agent has been motivated by the problem of creating heterogeneous assemblages of interdependent modules (agents) to solve complex tasks in complex domains— be they economies or soccer games — here the autonomy of the agent aligns again with the tractability of the decomposition of the complex task into interacting parts.

The so called "gang-of-four" design patterns book — **E. Gamma, R. Helm, R. Johnson, J. Vlissides**, *Design Patterns, Elements of reusable object-oriented software*. Addison-Wesley, 1995.

Primary texts on software engineering's ideas concerning the "Inversion of Control" are hard to come by. An overview of a broad variety of "framework integration problems" can be found in: **M. Mattsson, J. Bosch, M. E. Fayad**, *Framework integration problems, causes, solutions*. Communications of the ACM, 42 (10). 1999.

Online resources abound however:

The Apache project's Excalibur project: http://excalibur.apache.org/

the "HiveMind" project at Apache Jakarta: http://jakarta.apache.org/hivemind/ioc.html

The PicoContainer framework: http://www.picocontainer.org/

The Spring framework: http://www.springframework.org/

The tendency here, in both micro- and macro- conceptions of the problem, is to simplify the interconnectivity between these modules (agents). Hallmarks of this trend are extremely diverse, but one might re-read blackboard systems, artificial biochemistries, and various instantiation languages for behavior systems as searches for a minimal but powerful "glue" for signal flow, execution control or system instantiation. Each, I believe, is in insistence on the expressive role of the ubiquitous system diagram in AI's "system paper", with its complex boxes and thin arrows, each a desire for "modularity" re-articulated repeatedly.

Blackboard architectures have a long history both inside and outside the agent. For a software engineering perspective: **F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad,** and **M. Stal.** *Pattern-Oriented Software Architecture: A System Of Patterns.* West Sussex, England: John Wiley & Sons Ltd., 1996. Inside the agent: **B. Hayes-Roth,** *A Blackboard Architecture for Control,* Artificial Intelligence, 1985.

Synthetic biochemical communication: **S. Grand, D. Cliff, A. Malhotra,** *Creatures: artificial life autonomous software agents for home entertainment*. Proceedings of the first international conference on Autonomous agents, ACM, 1997.

Inside purer software engineering pursuits, similar problems and solutions are devised and re-devised. Most prevalent are the problems surrounding the instantiation of complex assemblages, and there is a thread of solutions that are typically referred to as the "inversion of control" or more tersely, IoC.

Many IoC systems propose a separate instantiation language (typically XML) for all material that is written in some other language — one programming technique for the boxes, another for the arrows. Still more IoC systems provide registry and retrieval mechanisms for objects to use in order to find the other objects that they ought to connect to — a central place for boxes to find their arrows.

XML is a ubiquitous W3C committee standard — http://www.w3.org/XML/

The goal in both cases is to *decouple* modules from each other, indeed, to keep modules modular, the boxes boxed-up and the arrows lightweight. The dreams of the modular cure many of the things that were so hard to maintain during the development of *alphaWolf* — separation of effect, incremental "testability", a late binding reconfigurability and the ability to reuse pieces of a work in the next.

IoC systems that use a separate configuration language, commit themselves to two positions. Firstly that the glue between systems is necessarily simpler than

the systems being glued together — that the box is bigger than the arrow —, and that this gluing does not require or deserve the complexities of a "full strength" programming language nor the environments that accompany them — that the organization of arrows is simpler than the contents of the box. Secondly that this assemblage is separated both temporally and (metaphorically) spatially from the cores of the modules themselves — these systems talk of a "design time", where the arrangement of modules is decided upon prior to execution, and a design space outside the modules.

Indeed, IoC solutions in general fail to allow the *inversion* of their particular inversion of control — or, less rhetorically, they fail to be particularly sophisticated about where the control (instantiation, execution or communication) ends up after it is taken away from the insides of the module. These systems are not failing their problem domain, but the challenges that they have been designed to face simply are not as complex as the architectural problems fundamental in making complex agents. The idea, therefore, that a module might, during execution, long after instantiation, reorganize existing modules, construct some anew and partially delete some more, is at best a rather long way from traditional IoC systems' point of departure. To deposit control into a central registry, a configuration file or a set of instantiation descriptions, is a fundamental dilution of the power of a module in the system over the modular, and, if our idea of the module is our receptacle for our introspection, our reusability and our extensibility strategies, this maneuver reinforces the problems of constructing complex assemblages of modules with complex life-cycles.

This of course isn't of much relevance for problems where the connections between modules are a simple affair and the modules themselves provide all of the power, but our agents are already dynamic and getting more so as we move from *Loops* to *how long*.... Perception systems grow, action systems grow, long-term authorship techniques cut across both, motor systems model what actions sys-

tems do, action systems set one structure up only to develop another later. Thus, artificial agents do not appear to be in the group of problems that allow the longer-term use of parallel instantiation languages.

It is worth noting in passing that early in the work of the Synthetic Characters group just such an instantiation language was created and used (for the large-scale installations *Swamped!*, 1997 and (void *) — *a cast of characters*, 1998). This design, which was part of the *Scoot* framework, was ultimately superseded and replaced by the current 'c' series of agent toolkits for a number of reasons, not least of which are the arguments presented above. Finally, we note of course that these IoC strategies run parallel to, but are in general more sophisticated than, the virtual wire of environments for digital art — which embody the fantasy of the system diagram — but in all fields the tactics are the same.

It is common to refer to the instantiation phase of an IoC container system as the time when "dependency injection", occurs. This is the time when connections between systems are forged, either pushed to modules from a central description of what the connections should be, or pulled by the modules from a central naming service. In the work that follows we reject the singular nature of the "design time" and look at structures that remain malleable across the life-cycle of the agent, and some structures across the life-cycle of the creation of the artwork.

This project shares, indeed inherits, the broad tactical goal of IoC systems — to find a low number of powerful, tractable, indeed author-able, strategies for controlling, assembling, ordering the execution and communication of modules. But our approach here differ from previous IoC attempts in that it admits immediately that that "low number" may be in fact greater than one; that the control, assemblage and ordering of modules are intersecting but not identical problems; and that these issues necessarily breach attempts to contain them

B. Blumberg, *Swamped! Using plush toys to direct autonomous animated characters.* Proceedings of SIGGRAPH 98: conference abstracts and applications., ACM Press, 1998.

B. Blumberg, *(void*): A Cast of Characters.* Proceedings of SIGGRAPH 99: conference abstracts and applications. ACM Press, 1999.

within the confines of instantiation languages, of "design time" or any other place which is essentially outside modules organized.

Related to modern IoC systems is the current excitement surrounding Aspect-Oriented programming. AoP, seeks to augment the module vocabulary of object oriented programming to include "concerns" that cut across several classes, instances or methods. Like "mainstream" AoP we are interested in finding alternative connective relationships between modules and alternative authorship strategies for maintaining these relationships. And we will use some of the same techniques that AoP under Java uses — instantiation time, byte-code injection and, most recently, load-time annotations. Unlike AoP we are not looking for generic solutions, but rather very specific ones for the problems that arise while authoring agents.

For an overview of Aspect-Oriented Programming issues, there is a special issue of the Communications of the ACM: **T. Elrad, R. E. Filman, A. Bader,** *Aspect Oriented Programming : Introduction,* Communications of the ACM, October 2001.

In spirit we are most influenced by the AspectJ project: http://eclipse.org/aspectj

2. The Context-Tree, a new "working memory" for agents.

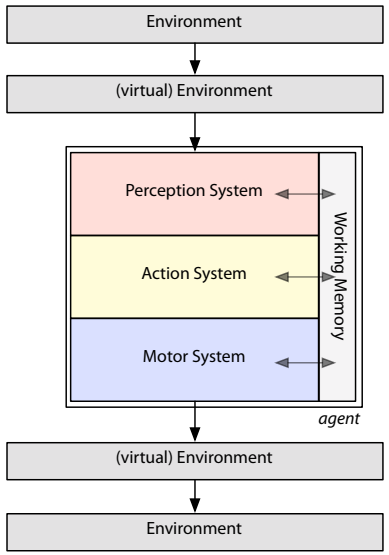


figure 68. The agent decomposed again. According to this diagram "working memory" acts as a conduit between all internal systems.

In the initial description of the c5 toolkit we drew a diagram, a decomposition into perception, action and motor systems connected by a ubiquitous "working memory".

This "working memory" serves as a communication channel, a persistent blackboard where systems could write and read, post and receive messages to each other, while remaining relatively uncoupled. Earlier, I refused to draw real arrows between these boxes — denying the implication that one particular thing flowed in a direction, coupled in a particular way or assumed control over any other. We can continue to refuse to draw arrows, but analyzing how separate concerns inside the agent-toolkit *couple* in these ways is, however, going to be an unavoidable aspect of taming complexities of large agents. In this section we develop a replacement for the simple blackboard-like working memory of c5, called the *context tree*.

The context tree is a tree of execution contexts, or scopes, that loosely follows the execution of the complete code-base involved in a particular work. In its simplest deployment the context tree has the following properties:

- for any given moment during the execution cycle, there is a single special level of the tree (branch or root) that is the "current context".
- each context has a name, and, optionally, any number of children.
- names are not unique throughout the tree, but are unique within a single group of children.
- each context has a parent, bar the top of the tree, the "root context".
- the tree is typically, but not always, fully explored in each execution cycle.
- the mapping from source-code line number to execution context is potentially one-to-many: if a line of code is revisited during execution it is not necessarily revisited in the same context. Context scope is thus a dynamic scope, rather than something that can be either statically or lexically determined.
- a context has any number of named elements, a mapping from name to object reference, and these are separate from namespace of children contexts. Named elements can be looked up with respect to the current context, should no element be found, successive parent contexts are searched until this element, or *key* is found.
- navigation through the tree, the creation and deletion of contexts is explicit.
- the other parts of the context tree can be referenced and searched both relatively and absolutely. The context-tree has aspects of a runtime database.

For information about dynamic scope in Perl
— L. Wall, T. Christiansen, J. Orwant,
Programming Perl, O'Reilly, 2000.

For an entry to the debate over Lisp's (early) dynamic scoping:
http://en.wikipedia.org/wiki/Scope_(programming)

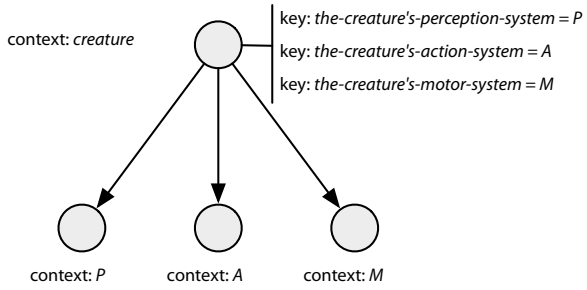


figure 69. The creature context and its three child contexts P, A and M.

Clearly my context tree shares many similarities with *dynamic scoping* found, for example, in some dialects of Lisp or more recently Perl. However it differs from these implementations mainly through its explicitness — dynamic scope is not implicitly woven from the language in which the code is written, but rather explicitly navigated, named and used by code that, in this case, may be written in any number of languages. However, given the above description of the context tree it should be clear that there really isn't very much complexity to it at all. An acceptable simple implementation of the context tree is simply a tree of named hash-maps. As we develop our use of the context tree, we will begin to weaken some of these assumptions: we shall see contexts with multiple parents, and language-level features for accessing and navigating the tree.

It's reasonably straightforward to see how to turn this tree into a solution for simple inversion of control problems. A creature needs a perception system, an action system and a motor system and we have a particular instantiated perception system P, a particular action system A, and a particular motor system M; we make a creature-level context creature that contains the key-value bindings *the-perception-system*:P, *the-action-system*:A , and *the-motor-system*:M. We'd like to write this with as little fanfare as possible, in Java (with the "keys" statically imported from definition interfaces):

```
thePerceptionSystem.set(P);
myP = thePreceptionSystem.get();
```

and in Python (using a class specifically designed for context tree access):

```
c.thePerceptionSystem = P;
```

We'll see an even more minimal interface to the context tree below, *page 242*. Typically, Keys like *thePerceptionSystem* are static, that is globally importable and accessible — they obtain that locality and module specificity not through the

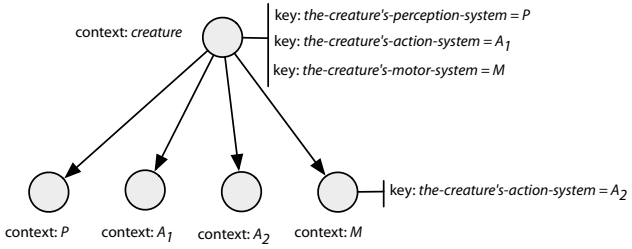


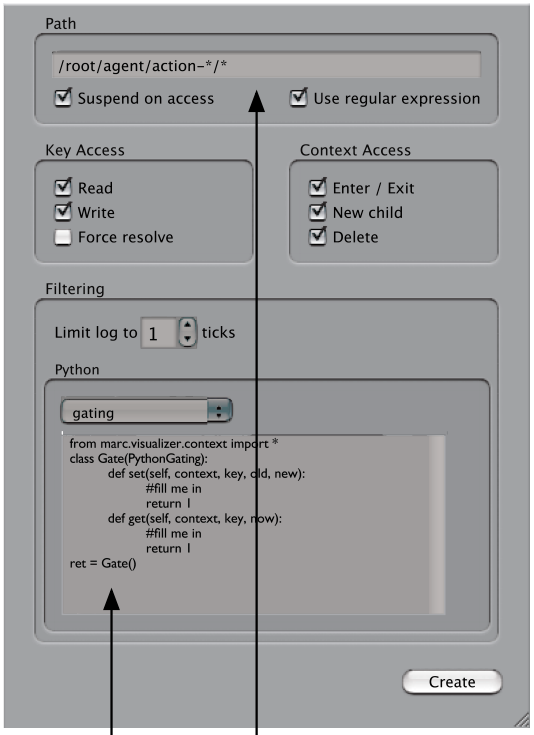
figure 70. One way of ensuring that M refers to A_2 is to write the reference directly into M, locally overriding it for this part of the context tree.

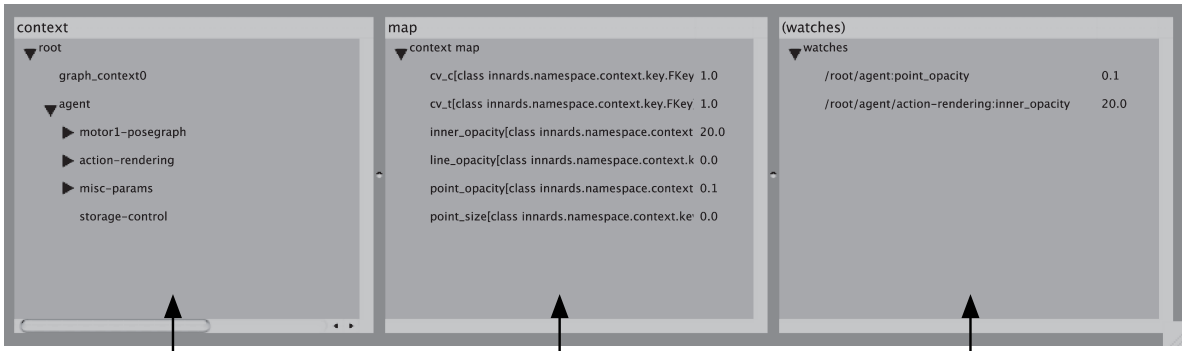
class-hierarchy-like instance fields but through the context hierarchy. However, later we might see storage that is both instance and context local, *page 220*. In other cases we might pick some other hierarchy structure to present as a tree — in the graphical user interface discussed later, we consider the hierarchy of views as a "context-tree-like" tree, *page 397*. No matter, for having constructed this abstraction, we are free to choose the granularity and the tree to apply it to. For the purposes of this discussion we shall assume the most typical case, there is a single, statically accessible context-tree shared by the entire runtime system.

Modules P, A, and M execute in their own, child contexts of creature. When the motor system M needs to find a reference to the action system it looks up *the-action-system* in its own context, although there is no binding there, there is a binding in the parent context creature and action system A is returned. The box thus finds its arrow.

Simple as this example is, there are a few key results to note:

this is a weakly coupling assemblage — this action system is never stored as an module-level variable (for example, it is never stored as an instance variable for any particular object), it can always be obtained from the context tree. The action system can change completely (to the point that the action system is actually a different object reference) without explicitly accessing or notifying any other system. For this to be useful to consumers of this information, context-tree lookup should be fast enough; this is easily achieved by a variety of caching mechanisms that turn a context tree search into a single hash-map lookup in most cases. For this to be useful to authors of systems there should be notification mechanisms that can provide hand-off between changing systems.





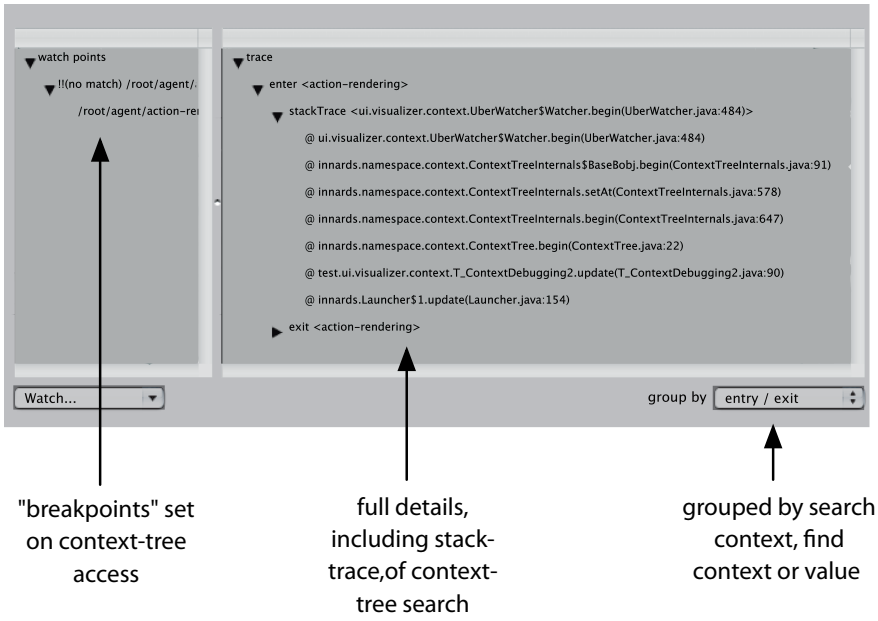


figure 71. The context tree becomes a place where we can focus the attention of custom debugging interfaces, to assist in the creation of agents. Shown here are the interfaces to the context-tree visualizer and the context-tree "breakpoint" interface. Breakpoints, which run on context and key access, are stored with the agent, and can be executed without any graphical intervention. Thus the boundary between "debugging" and "finished work" is blurred

the interconnect between modules is traceable — the first step of all inter-module communication is a call to the context-tree interface. If we want code to monitor, reflect upon or perhaps even interpose itself between, the action system and M or the motor system and A then the site of this intercession is clear. The result of this is that we focus a monitoring and tool building effort on the context tree and, for our tools and for our own navigations of the tree we can exploit the hierarchy's implicit "locality of effect": changes (be they modifications or bugs) to a context affect only children.

the modules remain mobile — the motor system can be executed inside a different context (for example a different creature) and its binding lookups from the point of view of M will change to reflect the new context. For this to be useful, it should be easy to cache and store state that needs to be context dependent as correctly context dependent. This is achieved by building higher-level container classes that are automatically backed by "context local storage" and by building language-level constructs that make, for example, "context locality" as easy to achieve as "instance locality" and "class locality" is in whatever object-oriented language that we have chosen, *page 220*.

This concludes the introduction to the context tree — my candidate replacement for an agents central blackboard, an alternative to the arrows in a system diagram. The section that follows simply takes this structure and builds more useful structures on top of it. Compared to blackboard architecture the context tree has more structure — a nest of searchable blackboards, an explicit and dynamic hierarchy of execution contexts that is tied to the execution of code. Compared to an "arrow" it has much less structure, a much less narrow focus, it provokes much weaker couplings between modules. Therefore the context tree,

as proposed, is a more complex compromise between these two simple solutions. It sacrifices some of the apparent simplicities of either of these two positions, hopefully, in exchange for simplifying the real problems that come with their use.

3. The uses of the context tree

However, in our two examples above, none of these decoupling "victories" are secure against all possible manipulations of the contents, execution cycles and the assemblies of P, A, and M. And the real power that the context tree has over a central registry is that its internal structure somehow reflects automatically the execution and use patterns of the model that refer back to it. We'll need to see some more complex examples for this power to be obvious.

For example: what happens if there are *two* action systems to communicate with A_1 and A_2? Where does the second go? How does A_2 get a share of the communication between A_1 and M? Something similar to the well known Façade pattern might be deployed to make two action systems appear as one to the motor system, but just as the façade pattern hides its presence from the caller it hides its presence from our previously traceable interconnect; it is a "local trick" and doesn't necessarily place control and share responsibility for its hidden manipulation in the right place. If we are going to perform such local maneuvers why have a central mechanism to begin with?

The actual problem behind this is an insufficient inversion of control — the motor system and a specific action system remain too strongly coupled together even when they find each other through looking each other up in the context tree. Better to note that much communication between the two systems can be articulated inside the context tree itself, with the posting and querying of results or elements that can be used to obtain results. Our pattern shifts then: rather

The Façade pattern is from: **E. Gamma, R. Helm, R. Johnson, J. Vlissides**, *Design Patterns, Elements of reusable object-oriented software*. Addison-Wesley, 1995.

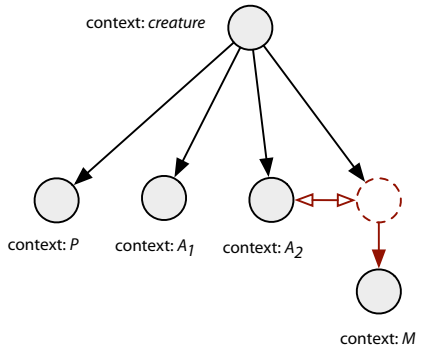


figure 72.A strong coupling between A_2 and M.

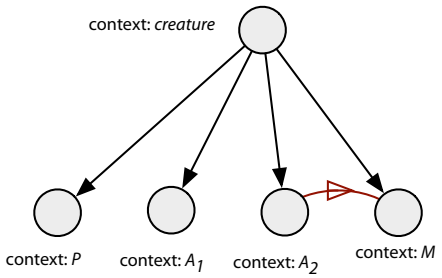


figure 73.A local coupling between A_2 and M.

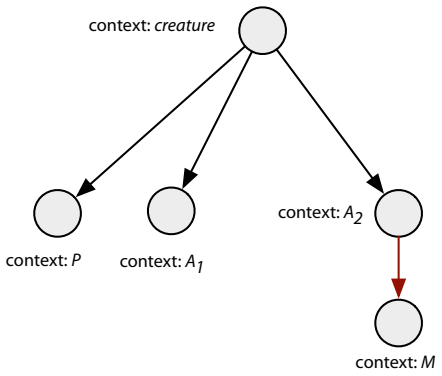


figure 74.A private coupling between A_2 and M.

than have a reference to *the-creature's-action-system* at the creature level (which is really the act to blame for our commitment to a single system at that level) we have our action system and motor system post results to and query information from the context tree — these results and queries are arbitrated by their own individual keys. Loose coupling can be achieved by using the context creature for this blackboard; results from both A_1 and A_2 are written here, in an execution dependent order.

Asymmetrically strong couplings can be achieved and with them a variety of execution independence:

A **stronger**, more detailed coupling, from A_x to M can be made by injecting a reference to A_x's context above M's context for use by M in looking up the results of A_x which are stored local to A_x — this coupling can be easily arranged by an assembly mechanism completely external to both A_x and M and independent of the ordering of A_1 and A_2;

A **local** coupling from A_x to M can be constructed by injecting the results of A directly into M's context — here A_x must come to know something about the existence of M's context, but nothing of the specifics of M's itself. Again this is (A_1,A_2) order independent; or

A **private** coupling by moving M to a sub-context of a particular A_x — this often makes A_x responsible for other aspects of M's life cycle including the ordering of A_x and M.

The context tree cannot make the decision as to which kind of coupling is more appropriate, but it does at least allow the decision to be made. And in each case, this serves the end of making the communication between modules more explicit, more standard (and thus observable by our generic context tree inspection tools) and, depending on how these results are described in code, potentially more declarative.

Looking back at the complexities illustrated in chapter 2 on the "large-agent" system *alphaWolf* we see a number that could be directly treated by using the context-tree as a general mechanism for coupling modules, and it is not difficult to hypothesize uses for each of the above couplings in this project. The coupling diagrams of figure 25, *page 95*, tell three stories of the development of *alphaWolf* and hide three more. One is the sheer number of connections between between disparate parts of code — which is in itself an argument for a strong and general purpose way of connecting things without boilerplate code. The second is the increase of connection density as the project grew. As modules "split" (presumably for the purposes of localization and testing) they duplicate and drag with them all of the previous connections to systems and add usually at least two more connections between the newly created modules — observe "fight action" and "toodle action" appearing out of the main "action system". Thirdly as the project develops, modules that were general purpose become specialized, and in doing so, other modules assume specific knowledge of that specialization — the transformation from "perception system" to "wolf perception system", and the leakage of this concrete implementation into other systems. What is hidden in these figures is are the "modules" that are never created because the coupling to their environment would be so strong, when expressed in method calls and shared instance variables, as to render the exercise futile. Equally hidden, and equally absent, are the "mock" objects that were difficult to create for the purposes of testing other modules in the presence of such detailed shared knowledge of the implementation of each module. Finally, hidden on the diagram, since it is difficult to represent pictorially, are the careful aggregation and ordering of one module's results by another in order to present this information in turn to other modules — for example, the perception system bundling information from the prioprioception system in order to pass it to parts of the action system.

The context-tree speaks to all of these problems. It offers a communicative complexity that scales with the number of modules, not the number of connections. It offers the potential of the utterly "code-free" fission of a module since the ability to find and connect to modules is stored outside the module being fissioned. We shall see that the context-tree offers module-external methods of shielding the implementation details of a module from others; a more detailed, yet potentially safer coupling that decreases the the threshold for modularity; and, as illustrated above, a module-external place where code can aggregate and treat the output of several modules before passing them up or along the context-tree for the consumption of other modules.

However in a more general sense, we have only begun the process of decoupling the modules of the agent — and we have simply deferred the problem rather than solved it if disparate parts of our hypothetical systems A or M need know about the specifics of the coupling arrangement — if special code need be written to implement these different classes of couplings. Rather, it should make no different to the insides of the modules to access a variable in each of these coupling scenarios. It is to the decoupling of this decoupling that we turn next.

Context-tree container classes

Clearly we are presenting the idea that we can have a language-level binding that appears to be a something like a "regular variable" (for example, in java an instance member of some kind of reference type, in python a class attribute).

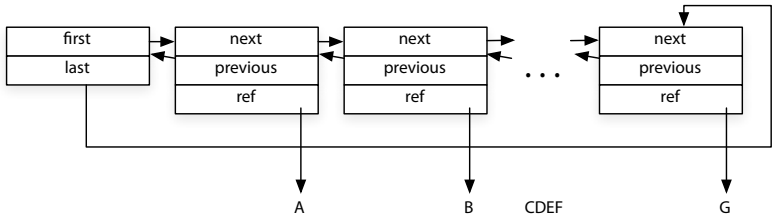
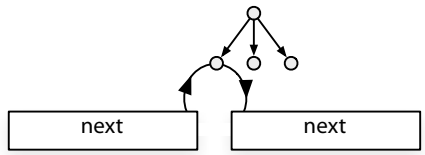


figure 75. The canonical linked list structure is an open network of references. We can replace the references with context-tree lookups to yield a context-tree-aware container class.



For example, in our most plain java we can define a context-tree key class with the following interface:

```
interface Key<t_value>{

    // looks up the binding for this key from the context tree
    t_value get();

    // sets this binding to be this value
    void set(t_value value);
}
```

This is the interface used for our previous examples.

In certain circumstances is possible to hide the presence of even this single level of indirection in Java, and in languages that are more directly malleable, such as Python, these meta-class level manipulations have been explicitly built in to the language. No matter, for even with this shim we can construct higher level data-structures where the references to objects have been replaced by these context-tree bindings.

For example we can take the canonical linked-list structure and translate the *next*, *previous*, and *object* references (together with the *head* references) to unique context-tree bindings. This gives us a list container class that at each level of the context tree acts as a list just inherits the semantics of the hierarchical context tree lookup. In particular, changes to the list in parent contexts are visible in all children contexts while the opposite is never the case. This holds both for non-structural (changes of what a particular list element refers to) and structural alterations of the list (deletions and additions to the list); when structural modifications present at parents and children overlap, then the list appears to follow the principle that the child context overrules the parent context for the purposes of that child context.

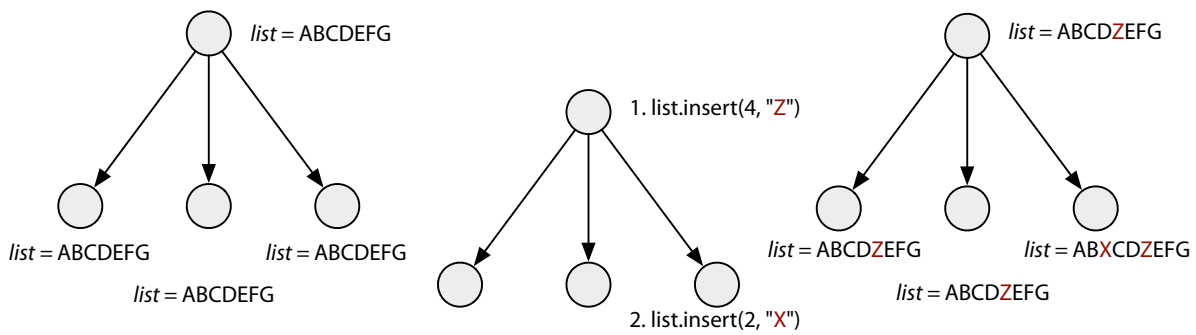


figure 76. The context-tree list extends the semantics of the context tree to a list container. Here elements stored in all parent contexts are visible in all children contexts. Similarly operations on the list, additions and deletions appear to occur at that level and all levels below.

The Java Collections framework is a standard toolkit of container classes — for a tutorial see:

http://java.sun.com/docs/books/tutorial/collections/

Similarly, "automatic" translation from instance-local to context-and-instance-local data-structures is trivial in the case of the (hash)map and the binary-tree, and thus is able to re-express the complete set of primary interfaces of the Java "collections framework". These context-tree-local container classes are then stored as conventional instance or class members, the further level of indirection afforded by storing these indirectly seldom being useful.

Programming in the interstices — code injection

These higher-level context storage containers allow the bootstrapping of yet higher-level programming techniques. When asked for its value our context-key local class asks the context tree for the value of a binding and returns it. When used as a blackboard for the posting and retrieval of results this is a site of communication between one module and another, and, as such, it is also a site of coupling of a different sort — one connected to systems' expectations of the semantics (what it means) of their communication rather than the syntax (where it is). We'll begin with a toy problem, although as we will see below this example occurred a great number of times during these multi-year projects, potentially threatening the very collaboration that was taking place.

A posts a value *A's result*:4.0 to the context tree and B looks for *A's result* as an important input. But the code for B was written a long, long time ago, and we've just compiled it again and mixed it into the agent in a hurry; A has been completely rewritten since then, and the scale of A's result has changed — how can we make it appear to B that A's units are twice as big?

One way to achieve the re-scaling is to execute code after A's posting but prior to B's reading that either rewrites the binding or injects a new binding into a context more local to B. These are clearly clumsy solutions, and neither of these solutions work well in practice. They introduce order-dependence where none previously existed and they interact poorly with a module C that is also interested in *A's result*.

A more interesting and maintainable technique is to open up the indirection provided by the context-tree key to the context tree itself.

Our key now looks like:

```
interface Key<t_value>{

    t_value get();
    void set(t_value value);

➡   void addToLookupStack(CodeElement<t_value> element);

}
```

where our CodeElement interface :

```
interface CodeElement<t_value>{

    t_value open(t_value filter);
    t_value close(t_value filter);
}
```

we can now restate key's **get** behavior in terms of this filtering stack, in python-like pseudo-code:

```
returnValue = nothing;

for element in codeElements:
    returnValue = element.open(returnValue)

for element in reverse(codeElement):
    returnValue = element.close(returnValue)

return returnValue
```

Keys come with a CodeElement that heads their stack that just looks the key up from the context tree as usual. And of course, we can write a CodeElement that has an close() that performs the unit correction between B and A in a matter of a few lines. Maintaining two methods open() and close() allow filters to choose to pre-empt the main lookup. These CodeElement fragments are completely independent and potentially persistable.

The final twist is to make this stack of CodeElements a context-tree-local linked list. With this we can add our re-scaling code element to the list inside B's context and inside B's context alone. From within this context it is part of the stack executed to get at *A's result* and it gets its opportunity to modify the value passed through it accordingly; from outside this context it is not part of the list.

We are now in a position to begin to see the relationship between the context tree and inversion of control container systems. Comparing these context-tree extensible, context-tree lookup's to conventional IoC's push or pull "dependency injection" — the connection of instantiated modules — we might be tempted to claim the term "code injection" for the more aspect-oriented context-tree system. For here it isn't (just) the references that are getting hooked up through the context tree, but the semantics of the actual reference types offered by the underlying language that are being subtly stretched from outside the module of

small amounts of code. This is neither strictly "pull" or "push" — these injections may come from other modules, typically parent modules, and one module's "central naming service" is another module's peer or child.

My context tree provides two other interstitial sites that are worth mentioning — *watches* and *traps*. Watches aid in the creation of context-tree debugging tools and allow code to be executed whenever slots in the context tree change. It is possible to implement this feature with zero additional cost in the case that there are no watches at or above a particular context, and this feature is designed mainly for debugging rather than self-monitoring. Traps, on the other hand, are called whenever contexts are entered, exited, re-parented, have children added to them, or deleted and are useful in maintaining caches of information (that may need to be updated if the topology of the context tree changes) or providing hooks for controlled shutdown in the case of context deletion. Later, I shall introduce structures that require this kind of caching in order to achieve good speeds at runtime, and we shall see some uses of the context tree for life cycle management in general, *page 231*.

Storing parts of the context tree — the technical support for naming

I've presented the context tree as a general purpose "working memory" for agents — a place where systems can post and read information and thus communicate in a loosely coupled fashion. This is an accurate description for its use in the agents in *The Music Creatures*, *Weather for an interactive window*, *Max, 22, how long...* and *Imagery for Jeux Deux* and all other agents within the most recent versions of the c5 toolkit. However, the context tree began life as a solution to a slightly different problem — not as a substrate for communication between processes but as a place for communication between artist and agent during the creative process.

The hierarchical outline of the tree makes it ideal for configuring rendering parameters broadly, before stepping down a few levels of the tree for local control over a specific agent, a specific shape or a specific line. And being able to distribute code throughout the interstices of the rendering is of course extremely powerful — for example, this makes it easy to programmatically *distribute variety* — of rendering styles, or behavior, *page 114*.

This technique works so well that I invested much time in creating graphical and textural interfaces for the context tree to set, inspect and manipulate these values and injected code. The context tree, with its structure dynamically determined by the execution of the agent, is its own "ideal interface" for distributing parameters. Creating *Loops* and *The Music Creatures* was just as much a matter of traversing the context tree of the colony and manipulating parameters as it was storing them, learning them and recalling them, *page 114*.

The filtration example above was presented as a toy example — and it certainly does look like a lot of trouble to go to for to multiply a number by two. However, this very problem appears frequently in practice. The context tree's scoped accessibility makes it an ideal place to put the large number of *ad hoc* parameters that get set and reset during the creation of an art work — be they line thicknesses, colors, noise parameters, filter coefficients. *how long...* moves around at least two hundred of these at various locations of the context tree, and almost everything that isn't specified in the action system in *Loops* is specified by these numbers — in the motor system, the graphics system and the global control of the colony.

However, there is a problem with storing these values. The resulting parameters aren't just numbers — they have been hard fought for and hard won; they might represent a considerable effort to tune the appearance of a line — e.g. *Loops* — there might have been a considerable amount of offline learning in-

volved on the value of a coefficient — e.g. *Music Creatures* — they might represent a considerable amount of consensus inside a collaborative work as to what a particular appearance should be, last month or even last year — e.g. *how long...*. Such is their importance in the development, one clearly needs a strategy for storing them past the life-cycle of a single execution of the work and past the memory span surrounding a rehearsal or an improvisation.

This storage of these parts of the context tree itself appears to be easily achievable by taking contexts and writing them to disk, and indeed this tree-like structure serializes to and from human-readable Xml extremely well. But these persistent numbers are not built from stable material, they are not referentially stable: line thicknesses are being developed at the same as the line drawer which today uses a selection of multiple lines rather than the simple one it did last week (*Loops*); a new agent wants to reuse the same rendering styles, on different geometry (*The Music Creatures*); learnt filter coefficients for one body now have to be translated into a new smoothing coefficients for a completely different body that as of last month hangs upside down — (*how long...*).

Where there is a *storage problem* there is a *versioning problem* waiting to happen: these numbers are not loaded back into the same system that saved them, and the effort devoted into finding these parameters cannot be allowed to slowly halt the further development of the system. If it could, we would be faced with choosing from two frustrating inertias — a forward inertia that would discourage us from changing the *process*, having made the *choice*; and a backward inertia that would discourage us from making the *choice*, until the *process* is "finalized". Both cripple the exploration of the field of potential developed by our complex systems just at the time when that potential might solidify into the particular; both threaten the collaborations that I have been involved in at, arguably its weakest place, my ability to regenerate material that was previously the subject

of consensus early in the collaboration, despite the provisional nature of the systems generating it.

Having realized the significance of the problem, the solution turns on two ingredients added to our context-tree techniques. The first is a database that acts as a repository for named, subsets of keys take from branches of the context-tree — we'll call these subsets **persisted, partial trees**. These partial trees are stored with versioning information that exists, crucially, on two levels: at the level of the partial tree, and at the level of the individual key. Named partial trees are additionally arranged in this "database" into types: parameters for our line renderer would be a type; a variety of named partial trees, each with different names, would form a set of rendering styles that we are interested in deploying throughout the piece; and a set of partial trees of the same name might be an ordered history of how this particular rendering style has been modified during the development of the piece.

Particular dates can be identified with names inside the database — "before the January rehearsal" or "before line renderer 2 got fixed". Historical information about a particular key is never deleted, merely superseded; databases for the works presented in this thesis, some of which were constructed over a period of 2 years, reach a size of several megabytes. This size is perfectly manageable without recourse to more heavyweight, truly "database" back ends.

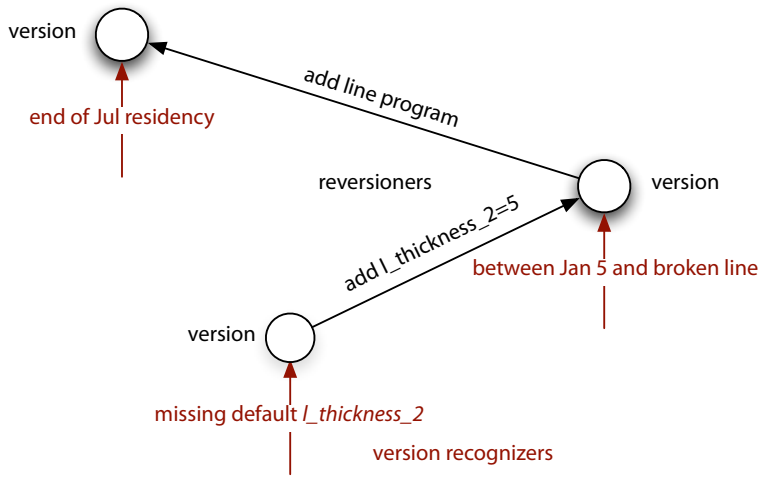


figure 77. Reversioners and version recognizers work together to ensure that persisted, partial trees are made up to date upon recall.

The second ingredient is a set of code resources that can first recognize when a key or a partial tree is "out of date" with respect to the current system and secondly do something about it. Version recognizers and "reversioners" can work in parallel and in series at both the partial tree and the individual key level. Neither recognizers nor reversioners can be made completely automatically, for the space of incompatible system changes resists standardization, but they can be made easily specifiable.

Version recognizers are generally quite simple, in most cases recognizing a specific date tag isn't the latest, or a particular key is missing from a persisted partial-tree. Reversioners generally act not by rewriting keys or adding previously missing keys but by injecting code into the keys or injecting code into new keys that tie their value to existing ones. The accumulated injected code allows old systems (or, more likely, old snippets of scripting code) talk the "old language" of the older keys while automatically presenting the newer interface to any module that cares.

Version recognizers are arranged and stored as vertices in a directed graph structure, with particular reversioners as edges of the graph, a path of accumulative "reversioning" is computed through directed search from the most recent (by date) version found from the database to the most recent (by date) version accessible to it in the graph. Should a reversioner $R_{1\to 2}$ that moves versions recognized by V_1 to those recognized by V_2 fail to turn a partial tree that is recognized as V_1 into a partial tree recognized by V_2 back-tracking occurs.

This "versioner graph" was successfully deployed in the pieces *how long...*, *22*, *Lifelike* and in *The Music Creatures* and an earlier prototype of the system was developed for *Loops* and *Weather for an interactive window*. Three of these five pieces were collaborations, two took place over a period of two and a half years. The version graphs for each of these dance pieces contained on the order of tens of nodes, some general, but some quite specific. In *Loops* and sometimes in *The Music Creatures* these partial trees are the very material from which the lowest-level representation of the generic pose-graph motor systems deployed in that work — these named "rendering styles" were in fact the named "body configurations" of the agents of *Loops*.

Indeed the small work, *Weather for an interactive window*, was mainly concerned with testing interfaces for live experimentation of rendering styles right up until minutes before the opening of the installation.

Although *Loops* exploited the context tree for its parameter distribution, it lacked a re-versioning graph system.

That the majority of these parameters were directly or indirectly related to rendering styles or body configurations probably stems from the the fact that these parameters are both the most readily placed as stored numbers or injected code and are the most likely site of tuning-while-running during a collaboration. However, it is also true that the results of offline and ongoing learning processes were stored and versioned in these persistence structures for maintenance across time-scales longer than the typical duration of the piece. Returning, once again, to *alphaWolf*, we can hypothesize uses for these techniques not just in tuning, say, the rendering parameters for the wolves, but rather in changing the way in which agents themselves were created. The partial storage of the context-tree could have provided a mechanism by which particular arrangements of the social learning of the wolf-pups that were proving problematic to debug could be stored and recalled independent of the ongoing development of the social learning mechanisms. The ability to quickly return, in the presence of structural changes, to the debugging "frontier" would have greatly assisted the tuning of the social "game dynamics" of the piece.

Looking back to *The Music Creatures*, *page 143*, and forward to a more general-purpose expression of this idea, *page 410*, this is neither the first or the last per-

sistence database that this thesis will discuss, and the theme of directly treating the historical development of a piece with custom tools continues: techniques that face up to responsibilities and consequences of long-term collaboration and thwart the encroachment of these choice / process inertias often developed when complex processes and artistic choice collide.

Authoring systems that change over time — the inverted context-tree list

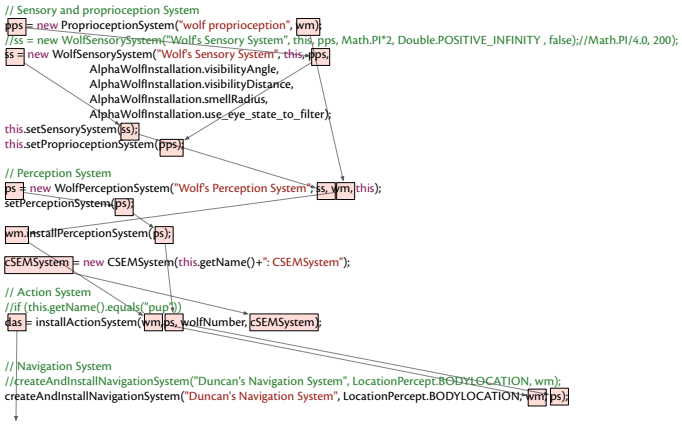


figure 78. A great many systems created and registered. The act of registering leaves a distributed, intricate trace that is hard to unravel.

The problems of constructing complex assemblages of interacting modules has motivated many of the interstitial techniques developed here. However, the problem of dismantling parts of these complex assemblages appears to be fundamentally even harder than constructing them in the first place. To see why this is the case, we should look back at the kinds of situations that appeared often in *alphaWolf*, where a great many systems were instantiated, registered and accreted.

Firstly we note that construction follows the execution of the code, but there is no closure around this or any construction in this object-oriented / imperative language. Each of those objects might wire themselves together and make more objects that keep references to others. This is easily written in an imperative style, and straightforward to execute assuming one has gotten the order correct, but the execution itself leaves no trace — it is possible and perhaps likely that these newly constructed, installed and registered objects don't have references to the objects that made them, installed them or maintain them as registered. Even if they do it's not clear that it is desirable for them to have the knowledge or the power to unmake, uninstall or de-register themselves. Such knowledge and access would represent a strong and, worse, diffuse and intricate coupling between disparate sub-parts.

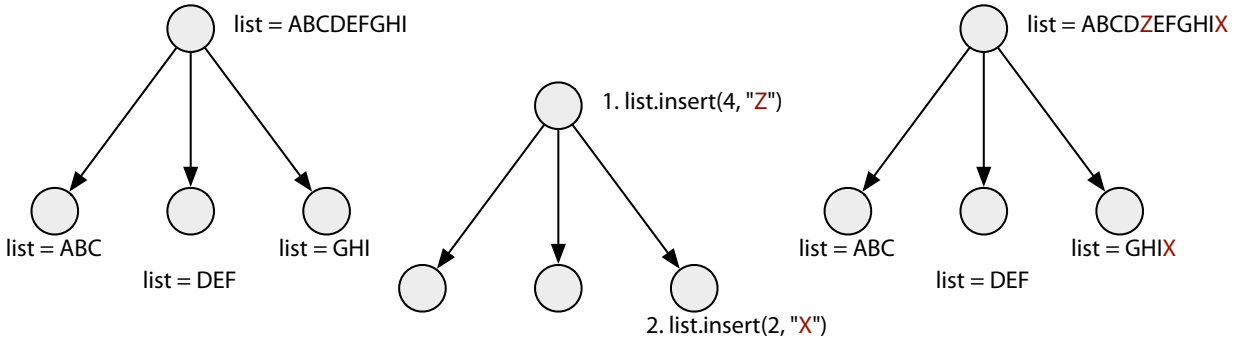


figure 79. The inverted context tree list appears to have the contents of a context and *all children* (as opposed to all parents).

Further complexity flows from the need to propagate a description of what is and what is not being torn down through the method calls doing the disassembly, it is not clear that this can this even be described without coupling the minutia of an assemblage to an external description of its boundaries. But we already have seen one description of the boundaries of a assemblage that doesn't need to be propagated anywhere — a branch of the context tree — and we have seen one species of container class that automatically reflects the state of the context tree. Perhaps these two ideas can be combined to make structural additions and deletions to the context tree equivalent to structural registrations and de-registrations.

Consider a new kind of list structure — the **inverted context-tree list**. Like our earlier linked list backed with context-local storage, its contents are actually distributed throughout the context tree, rather than stored in a particular instance or class field; and just like the our previous linked this this list is in actual fact a union of lists stored in a number of contexts. However, where the context-tree list was the union of all lists at the current context and its chain of parents, this inverted list is the union of all lists at the current context and all of its children. Unlike the previous linked list structure we cannot construct this list by

replacing object-references with context-tree key lookups, for context-tree key lookups proceed, in the absence of code injection, upward through the tree. However, in practice we certainly can reuse the same caching mechanisms that make access to the context tree fast to maintain this inverted list of lists at each level of the tree.

This list is now a primitive from which we can construct registration lists, execution orders and notification queues. Whenever there is a list of objects that need updating or notified when events occur, this list should be an inverted context-tree list; whenever there is a map of known services that might be called upon by a module, this map should be an inverted context-tree map. Delete part of the context tree and all references to objects installed at that level and below disappear, execute methods on this module in a different part of the context tree and a different set of objects to update or services are available. Further, upon deletion, the references disappear in a particularly orderly and consistent fashion: namely, simultaneously. And this simultaneous destruction occurs at a particular moment: at the last transition out of the deleted context they are never seen again. This is a time when, by definition, there is no code running that accesses these data-structures. Together with a few "language level" programming tricks, this facility can be made broadly usable, removing much of the boilerplate code that is involved in both de-registering and even registering systems' connections, *page 247*.

This idea, as already described, is very useful beyond simply enabling the disassembly of systems — indeed it permits the rapid assembly of the kinds of complex assemblages that the context-tree promotes from going unexploited in systems that need to delete parts of themselves during their execution. Often in the face of creating an agent or an artwork the focus is on assembling something, testing it, tuning it; having reached a point of confidence that that thing is heading the in right direction, one puts this piece down and takes up another

part of the work. Only when these pieces come together do their life-cycles begin to get complicated: when we need to go from one piece to another, incorporate one in another, instantiate three things rather than one.

By no means, however, is this kind of tear-down essential to the creation of graphical, interactive agents — the installations *Loops*, *Dobie*, *alphaWolf*, *The Music Creatures*, *Lifelike* all ran without any structural deletion occurring during their life-cycles. Similarly, most interactive works — be they authored in Max, Isadora, Director, or Flash — seldom change structurally much after initialization time — data flows through pre-made networks of modules, pre-loaded resources are moved to the fore or hidden.

These environments will be the subject of much discussion in chapter 8 — see the references therein.

However, in each of my early works, the problems of structural deletion made their presence felt — *Loops* became an infinite piece about the finite materials from which it was constructed; *Dobie* learns without bound, without forgetting; *alphaWolf* faced difficulties of such magnitude loading and deleting its constituent wolves that after their five minute growth cycle they were swapped around rather than deleted and reinitialized. As I moved to agents with more complex parts, their lives, and the simulations they inhabit, grow shorter: *The Music Creatures* only lived for around 7 minutes before dying — and with their accumulation of models and graphical material they might not have made it much past a few hours if left to run; and *Lifelike* ran for the duration of a 30-minute dance work, but the accumulative flux of points, lines, graphical resources, and behavior systems were so great I fear it would not make it past an hour. Is this the necessary price for live structural change?

Instead, these works (and commercial interactive programming environments), find a solution space (or a duration) where they do not have to confront the issues that come with tearing down previously constructed systems that may

have changed structurally; but at the same time, we are clearly interested in works that do change structurally and remain running over long time-scales.

So the above techniques, the context tree and the containers, enable *how long...* to have a parade of overlapping, interactive agents that come and go, that model and stop modeling, that add and subtract geometry. As an artwork there is no secret reason why it could not run indefinitely. If it were appropriate, we could loosen the scripting of the entrances and exits of agents to create an endless chance juxtapositions of bodies and perception systems. Perhaps this flexibility will be demonstrated an installation context at some future point. Regardless, the flexibility paid for itself in rehearsal.

Therefore together the context tree and its container classes allow for the assembly and automatic disassembly of modules. However, there is an alternative interpretation of this power that has implications squarely in the domain of artificial intelligence, not simply software engineering. Since the addition and deletion of things are tied to the context tree and since contexts nest we can use context-trees to implement structural *closures*. That is, we can open a child context, try some computation and, if we don't like the results just delete the context and it is as if nothing happened. If we do like the results, we need to propagate the contents of that child context up to the parent — overwriting slots that contain simple values, merging slots that contain lists etc. This kind of **speculative execution** allows systems to hypothesize about what would happen if it changed in a certain way. Unlike languages with built-in support for closures it is up to the programmer to decide what is and what isn't closed in. This is a problem since it is prone to error unless using context-tree-local storage is simple (see the annotation library, *page 242*), but it is a benefit because it allows us to choose what does and what doesn't get rolled back on this context-tree-level "undo". This technique has a number of in the works: *The Music Creatures* used an early version of this technique to speculatively close a sensitive period that

might in fact need to open again; *Loops Score* uses it in a number of places to see if performing a musical action will produce a series of notes that can be related in some way to what has already been scheduled, *page 251*. I expect that this technique will find an increasing role in future agents that undergo long execution, long-term structural change.

Further, such a facility — had it been offered by the toolkit at that time — would have radically changed how projects such as *alphaWolf* were authored. Not only would it have been possible to unravel the kinds of structures indicated in figure 78, but this error-prone glue code itself would disappear. By preserving the modularity of parts of the agents one could also prevent the unfortunate collapse of all three "kinds" of agents (pup, adult and caretaker) into a single action system — allowing the set of action tuples present in the system to grow and change during the life-cycle of the creature rather than creating a single action system with parts "shorted-out". This offers an improvement not just of computational complexity of running the action systems (which, in itself is dwarfed by the graphics and animation tasks) but of the complexity of debugging these systems, visualizing their execution, even just thinking through the results of modifying the source code (see, for example, the number of "optional sections" in the *alphaWolf* behavior code in figure 80, *page 238*). Further, a real deployment of the context-tree as a universal coupling between parts of an *alphaWolf* creature might have permitted modular testing, in particular the profiling and creation of the perception system and motor system independent of the action system, allowing in turn the multiple programming collaborators on that project to work more independently (directly treating the problems illustrated in figure 25, *page 95*).

The context-tree and system creation — sub-classable complex systems

The above sections have considered the context-tree-based solutions to the problems of constructing, manipulating, and disassembling complex assemblages of interacting "modules". This last section addresses the related problem of re-using them.

Often in object-oriented programming languages one expects to be able to use inheritance to provide a set of good base classes that will speed the implementation of, and reduce the amount of code required for, common specific classes. In an agent toolkit one expects to be able to create a simple agent in code by perhaps overriding a few methods from a base class that aggregates the machinery required for a graphical agent, an agent with a motor system, a perception system and an action system etc. Unfortunately, in practice, building a toolkit that offered such a super-class template has proved to be extremely difficult, and throughout the collaborative development of the 'c' series of agent toolkits, there has been an ongoing tension between powerful "abstractions" and useful "base-classes" . Here a apparently irreconcilable tension appears: between concretizing all-too-abstract elements in order to make specific agents, or abstracting all-too-concrete previous works to make the next. AlphaWolf falls, in my opinion, squarely inside the gap between these two poles — little of the code piece gets reused in the works that follow, yet the generic nature of some of the elements used in its creation can do little to prevent the growth of the behavior system files.

In this quandary, nothing less than one's technical development as an artist is at stake — the tension is between maintaining a broad ranging set of elements that are difficult to turn into free-standing artworks, versus accumulating knowledge, and experience, but no tangible tools or code.

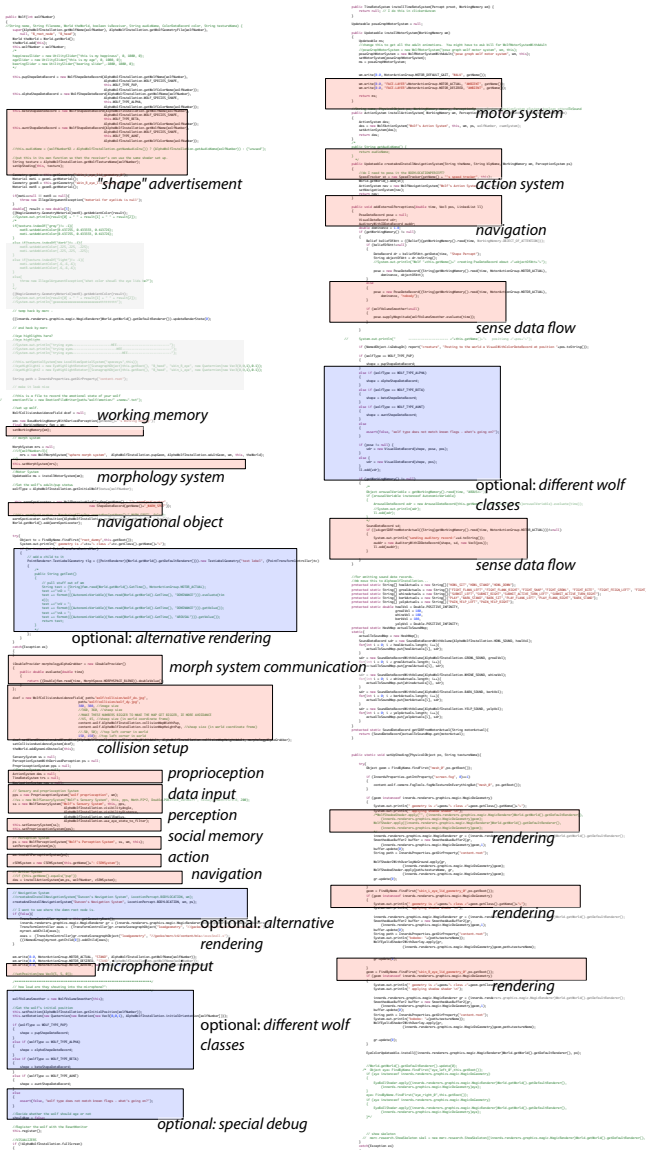


figure 80. Optional blocks (blue) and class instantiations (red) are distributed broadly through the *alphaWolf* action system.

Let us look at this complex "sub-classing" problem in detail. Perhaps the problem stems from the number of *options* that this super-class aggregation needs to present to its possible sub-classes. Rather than reducing possibilities as we descend down the inheritance chain to get closer to specific uses of the agent toolkit, the opposite happens: flexibility increases exponentially as we aggregate systems that are suffering from the same problem. So, unless there exists a mechanism for distributing defaults and overrides amongst these systems, either inopportune choices are made and the inheritance hierarchy is abandoned, or the base-classes provide so broad a base that they are hard to understand.

So far we have tried hard to allow the creation of complex assemblages that do not couple to their connections — essentially, their "parameters". The context tree defuses some of these tensions. We can have constructors for these aggregating classes that are written in a normal fashion — passing through, incorporating and storing parameters — as well as mutators that are coded almost as normal; all using the context tree. However, these classes do still couple in a number of ways to the classes that they instantiate. This instantiation is one way that super-classes commit to limiting the options for their potential future sub-classes and if this instantiation cannot be defaulted and overridden then we are very much in the situation outlined above.

The standard design pattern used to avoid this in practice is another inversion-of-control technique: the well known factory pattern. The factory pattern interposes logic in the name resolution and construction of classes. Rather than asking the language for a new instance of a specific class, one asks the factory for a new instance of a particular interface. This factory *decides* on what class should be instantiated and with what *parameters*. There are two problems associated with trying to deploy the factory pattern widely: providing enough information to the factory for that decision to take place and providing those parameters once that decision has occurred.

The context tree can clearly help with the distribution of parameters across a set of systems — this is the very purpose to which we have put it in the examples above. This ability could be pressed into backing a factory system — we could distribute the names of classes to instantiate just as easily as we can distribute other parameters. But perhaps there is a more apt and more flexible meeting of the context tree and the factory pattern.

We can use a version of the interstitial context-tree programming technique described above, *page 222*. Instead of shielding systems from changes that are incompatible with previously saved data, by installing context-specific filters on access to that data, we can install context-specific factory functions to instantiate classes.

We build up the language very similarly — and we should build it up, for it is going to become a common *programming* unit for a whole variety of systems. They key programming element is the the CodeElement from before:


```
interface CodeElement<t_value>{

    t_value open(t_value filter);
    t_value close(t_value filter);
}
```

These elements are stored and executed in context-local lists inside ExtendedKeys. This class offers what Key offers plus a number of convenient versions of the add method that help write more factory-based context tree programs. The most important of these methods are:

call(Object *o*) — wraps any object (that is presumed to have only one publicly accessible method) in a CodeElement and adds call to this element on the stack.

lookup(Key<t_value> *o*) — looks up a key from the context tree.

is(t_value *o*) — just returns this value into the execution order of the stack.

with(Key<t_value> *key*, t_value *value*) — places a CodeElement that temporarily sets a particular key to a particular value on the stack.

makeNew() — puts a CodeElement on the stack that will instantiate classes that are passed to it (runs in close(...))

A little care must be taken with how Java's generics and type system are used to make this syntax work. In general we admit not only t_value, the underlying value type for the keys, but <? extends t_value>, and Class<? extends t_value> to be passed into these methods. These methods are omitted for brevity. Internally, of course, Keys are free to ignore the parameter on the type, since the implementation of generics is not strictly part of Java's type system.

This lets us declare a default instantiation, perhaps in a creature base-class, like:

```
navigationSystemFactory.is(DefaultNavigation.class).makeNew();
```

And then from some other location, perhaps in the action system, where we need a navigation system:

```
navigationSystemFactory.get();
```

will suffice. This is the most straightforward of all possible examples. We can use the context tree to pass "keyword parameters" to this factory:

```
navigationSystemFactory.with(bodySize, 10).get();
```

We can, far away from this invocation or declaration, manipulate the factory lookup such that this action system gets a special navigation system, perhaps in a subclass of the base creature class:

```
navigationSystemFactory.inside("action-sys/").is(GraphicalNavigationSystem.class);
```

Or, less radically, we could just provide a better default:

```
navigationSystemFactory.with(bodySize, myBodySize);
```

In order to distribute these throughout the partially assembled context tree it is useful to have a path expression language to refer to contexts different from the current one. Rather than invent an expression language that handles hierarchies of attributed objects we borrow an extremely well thought-out, standardized expression language made for searching xml —XPath — and map its object model (which is usually the tree-like structure of elements and attributes in xml) onto the context tree (a structure of contexts and keys). This allows simple searching for contexts by name over the whole tree (looking for "action-sys" regardless of where we are):

XPath is a w3c spectfication — http://www.w3.org/TR/xpath

The flexible Java XPath engine used for this work is Jaxen — http://jaxen.org/faq.html

This project specifically allows XPath expressions to be evaluated over "custom" object models (here, the context tree, later, the *Fluid* view hierarchy).


```
navigationSystemFactory.inside("//action-sys/")...
```

as well as more complex expressions that may match multiple contexts (looking for a context that contains a value for *direction*):

```
navigationSystemFactory.insideAll("//*[ct:containsKey('direction')]")...
```

The methods that round out this interstitial programming environment are:

and(Object *o*) — call-s *o* should nothing have been found yet.

beforeAnd(Object *o*) — call-s *o* first in the stack, and only calls the rest of the stack if it doesn't come up with an object.

importing(Object *o*) — imports the context given by *o* into the stack at this point.

There is a strong similarity between these methods and the complex, multiple dispatch of the Common Lisp Object system. However, here, dispatch is extended in a context-centric way *external* to the class or method structure target. This borrows, then, the flavor of Aspect Orientation but is specifically focused on the problem of instantiation and allows configuration behavior to be modified live, on a per-context, that is a per-hierarchically-defined-module, basis reusing the functional groups that the context tree represents.

While these techniques do not make the problems of extending complex code assemblages evaporate they do make a great deal of different to the problem. Unlike the tangled "excesses" of earlier agents that, while informing the work that followed, were, as a set of code, ultimately abandoned after premiering, the agents of *how long...* were constructed both with more generic element and more simply. Thus, we will see these techniques exploited in the agents of *how long...* — in the LineAcceptor system, *page 325*, and the "subclasses" of the agents built between workshops and in the evenings between rehearsals would not have been possible without this mobility. Intense, but open collaborative practice requires a solution to the abstract and flexible / concrete and useful dichotomy — one's prior work to entering the theater must be useful if there's work to be done, but it must be flexible if there's a collaboration that is to occur.

3. An annotation tag library for context-tree use in Java

Although it is certainly possible to implement the above context-tree key class, and use the context tree itself in pure Java, the works discussed in this thesis have gravitated towards one of two alternative paths. They either provided extended, syntactic "sugar" for the context tree in the form of a pre-processor

language for Java or, more recently, a set of method and member annotations and a custom, byte-code injecting classloader that manipulates classes based on these annotations.

This latter implementation, which has only become possible with the most recent revision of the Java language, has the considerable benefit of standardization, allowing one to maintain an utterly conventional tool chain in the presence of even radical alterations to the semantics of Methods and Classes. Because of this, it will be this implementation that will be described here. It is in this "tag library" that we are the closest to "Aspect Oriented Programing", but mainstream AoP lacks the idea of a context-tree.

Java's annotations (similar in implementation to the annotations of C#) are essentially programmer-defined, structured "comments" in code that can be read at run-time, or in this case, class-load time. We start with our most basic annotation that tags classes:

@context

this informs our custom class-loader that this class has the potential to have the other tags from our library. It is a class-load-time error for subclasses of such classes to omit this tag and debug-time error for other tags to be present in a class without this tag. It provides both safety and optimization for the load time.

@dynamic(*name*=optional prefix)

marks this member variable as being context-local. subsequent Write and read access to and from this variable is rewritten to go through a context key. This context key's identifier defaults to the name of the member + the name of the class. Currently, only private object reference members of classes may have this tag.

@subcontext(*name*)

places the contents of this method inside a child of the current context called "name", creating this context if necessary. Crucially this has three properties that are hard to get right without language support: the tag exhibits the expected behavior for overridden methods, specifically their method bodies are wrapped and wrapped once regardless of the existence, location and number of any calls to methods in the superclass; secondly the context is correctly unwound should the method exit abnormally; and lastly this tag has the expected behavior even when the method annotated is a constructor.

@inside(*contextdescription*, *creationdescription*=default)

a more flexible form of "subcontext" that allows specification of two helper classes that define how to find the subcontext and how to create it should it not be found. In addition to the helper classes that "subcontext" uses that finds a named child context of the current context and creates one if it isn't there, other helpers have been found to be useful. One acts as "subcontext" does once and then from that point on goes back to that exact same context, regardless of the current context. The current context is restored upon the exit of the method. This is useful, for example, to ensure that methods of an instance are executed in the same context that was present when the instance was constructed.

@deferUntilEntry(*contextdescription*, *queuedescription*=default)
@deferUntilExit(*contextdescription*, *queuedescription*=default)

these two tags **defer the execution** of the body of the method until a specific context is entered or exited. These tags are only to be used on methods that have no return value. The queue description describes a class responsible for storing the deferred method calls, which need not be a simple list: subclasses that concatenate multiple method calls together into a single call are useful, more complex deferrals will be discussed, *page 275*.

These annotations clearly give a lot of power to a class to alter the scope surrounding the execution of a method in a way that maintains a coupling between object and execution context. They are not arbitrarily powerful — for example, it is impossible with these tags (but not with the underlying interfaces) to change the context in which a method executes based on a parameter to that method.

@autoUpdates

this tag marks a constructor of a class (or a whole class, thus marking all constructors). Constructors marked this way register the constructed instance with a **central auto-update list** fetched from context-tree local storage which, by default is an inverted context-tree list, *232*. Instances constructed this way will have their update(...) method called when the auto-update chain is updated. The interface for Updateables is trivial:

```
interface Updateable{

    void update();
}
```

A more complex life-cycle interface is optionally:

```
interface Task extends Updateable {

    void init();

    void update();

    void shutdown();
    void forceShutdown();
    void isShuttingDown();
    void hasShutdown();
}
```

This fuses two models together, optional shutdown and forced shutdown. A life-cycle object that has shutdown() called on it may opt to return *true* from isShuttingDown() and may ultimately return *true* from hasShutdown(), guaranteeing that this object will never have update() called on it again from this context. Alternatively, should the updator call forceShutdown() this component must expect never to have update called on it again (again, from this context). In this case, components that absolutely must shut down over a period of a few calls to update() should arrange for their update() to be called by some other means.

These life-cycle interfaces are implemented by a wide range of classes throughout *how long...*, 22, and *Loops Score*; the agents of *how long...* themselves implement these interfaces, as well as the rendering tasks that require graphics resources, and the individual graphical elements and scripts in the *Fluid* environment.

The deletion of contexts, which would silently prevent subsequent calls to update() for autoUpdatable-s stored in inverted context-tree lists without any call to shutdown(), and therefore violate the implied contract are monitored for by installing "traps" at the current level of the context tree.

@doesAutoUpdating
@doesAutoUpdatingOverContext
@doAutoUpdate

These tags are for classes that do the updating — instances that will be containers for other classes. The first tag marks a constructor of a class (or a whole class, thus marking all constructors) as owning part of the context-tree local auto update tree. The constructor (including super constructor invocations, and all inherited subclass constructors) is effectively wrapped in a pair of save(...) and restore(...) calls for the context-tree local storage for the central auto-update list. The second tag uses a inverted context-tree list rather than a conventional list to back the storage of this local update loop. This is useful in the case where a single instance is expecting to be updated in several times in different contexts. The third tag performs this auto-updating in the body of the following method in a "overriding safe" manner.

Of course, a majority of classes that are marked @doesAutoUpdating are marked @autoUpdates as well — they are "updatable" containers that create things that are updated in turn.

Several more tags will be added to this library through the same mechanism to complete the Diagram system, they integrate some of the features of that work back into the language — transforming what looks to the caller to be a method call into an object that is deferred, executable, inspectable, modifiable.

In *Francis Bacon: the logic of sensation*, French philosopher Gilles Deleuze presents the diagram as a stage of artistic creation, citing Bacon's example of a brush stroke which reveals that the mouth of a portrait could cut across the entire face, suddenly increasing the sense of distance and transforming the figuration. Deleuze affirms the role of chance in this act — the lines of the diagram are "irrational, involuntary, accidental, free, and haphazard". The diagram exists on the boundary between preparatory work and the act of painting proper; its chaos must be transformed into a new form of figuration.

G. Deleuze, *Francis Bacon: The Logic of Sensation*, D. W. Smith , T. Conley (trans.), University of Minnesota Press, 2003.

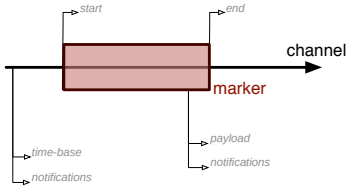


figure 81. The basic anatomy of a marker and a channel.

The Diagram framework, in this work, synthesizes many of the technologies described above to generate a canvas on which unsynchronized or uncoupled processes, perhaps action-selection systems or perceptual traces, can mark and out of which new figuration, here a new temporal patterning, can be found. It is simultaneously a loosening of techniques like the c5 action-group, hierarchical scene-graph-based graphics and the pose-graph motor system and an opportunity for more direct temporal specification.

There are two extremely minimal core elements to the Diagram framework: the **channel** and the **marker**. A marker is an object with a time and duration with respect to some time-base. A channel is an set of markers, ordered by onset time that shares a time-base with its markers. A channel has a unique channel context — that is, a part of a *local* context-tree that is associated with the Diagram system. All diagram system storage is local with respect to this context tree, thus all marker creation, modification and deletion is potentially speculative. Markers are always part of one and only one channel.

This micro-structure is generic enough to permit the re-expression of many of the data structures seen thus far. This channel / marker structure is a minimal subset of the representation behind the generic radial-basis channel, and behind the visual elements laid out in a sheet in Fluid, it will become our instantiated action, and our scheduled pose in the pose-graph motor system.

The first layer of Diagram focuses on building channels and markers extremely well. In particular it is occupied by building notification mechanisms (for the addition, change and deletion of markers) with respect to whole channels or individual markers. These notification chains support batching and event coalescing, *page 275*, and form the basis of the efficiency and efficacy of the algo-

rithms that use these channels. Both channels and markers are mutable, but can provide immutable views as well as views onto windows of time or segments.

A detailed discussion of these inner details here would be unmotivated, so we will wait until the uses of the framework come into focus in making *Loops* Score before describing the implementation aspects. The goal for the Diagram system is not to provide another action selection algorithm (although we shall see one, *page 251*), nor another perceptual framework (although, *page 259*), nor a replacement for the pose-graph motor system (although, by the time we see the *Fluid* environment Diagram will have turned the pose-graph inside out, *page 349*). Part of the goal is to simply re-articulate the previous structures in a form that reduces the barrier for interaction between them. But in doing so I will present a new "recomposition" of action and motor patterning. The goal is to build a support structure around these algorithms that extends their power, specifically in the realm of supporting complex temporal patterning, specifically for the use of the agent metaphor in time-based art. It renders explicit, indeed *graphic*, what was previously implicitly hidden in action system initialization code, the trace of execution through a motor system or the pattern of activation in a perception system. It allows the re-coupling of algorithmic processes that cut across systems and algorithms in the temporal domain.

Some principles, however, guide the development of the Diagram-based works and the uses to which we put the channels and markers — *Loops Score*, *how long...*, and to a lesser extent *22*, *Imagery for Jeux Deux* and *The Music Creatures* (which developed an older version of this Diagram work).

These principles are:

marker manipulation should be **reversible and inspectable** — we shall see algorithms for marker production (essentially nothing more than production systems implemented with an explicit time axis) and manipulation,

but very little information is ever irreversibly removed or irretrievably hidden in any of these manipulations. Rather than translating a marker by overwriting its position, a relationship is set up and maintained that actively moves a marker to the left. This relationship is added to the channel, visible in the diagram, inspectable by other processes. Where action-selection systems allow code to compete for expression, the Diagram system additionally allows processes to compete for the modification of expressed lines or channels. Notification and compression support make reading and storing (and strategically forgetting relationships) computationally tenable.

transient, experimental computation — on the other hand, the computational impact of this agglomerative network of relationships is bounded by the transient nature of these channels. Often they are only representing the near future, and at one end there is the present — a scheduler horizon — and at the other there is an increasing uncertainty about the future — a planning horizon. The detailed relationships are meant to be transient, compressed and discarded. Slicing channels and modifiable sub-views onto channels help with writing code that efficiently deals with windowed portions of time. Context-Tree backed storage allows structural modification to channels to be attempted in sub-context, experimentally conducted only to be effortlessly discarded.

highly accessible — the channel / marker idea would be abstract beyond the point of utility if it wasn't for the common set of "glue systems" that make Diagram-like computations fast and allows them all to share a common language. The idea itself is useless, and in addition not very diagram-like, should these channels and markers be inaccessible to systems that have little commitment to the Diagram system. In the work that follows the channel / marker system is forced into systems by extending the programming language, *page 242*, aggressively finding commonalities between

it and the visual tools (*Fluid*, *page 412*), between it and more traditional computer musical concerns (*Loops Score*) and between it and the way that motor systems are structured (*Loops Score*, *page 267*). Diagram will blur the separation between action selection (or more strictly action *layout*), the kind of motor sequencing that agents tend to do and the previous working memory / context-tree "blackboard" that was used as a communicative glue between systems.

As the description of the work that is based on this kernel continues it should be more apparent just what it is that is diagram-like about this diagram framework. The Diagram system's channels become a field, a rapidly receding canvas where multiple systems leave initially uncoordinated marks in time only to have their chance-like relationships enforced, reorganized or reshaped by other processes that induce patterns out of them and their histories.

Action selection in the Diagram framework

Diagram, as described up until this point, is missing a key part of the action system story — an action-selection strategy. While we would be free to take a c5 implementation of action-tuples and have its actions mark channels, it should be clear that this algorithm isn't quite taking full advantage of the channel / marker representation. Firstly, this approach knows little and says less about any time other than the present. Secondly, it doesn't then open up its action selection to external modification — its action-selection results are neither reversible or inspectable. Thirdly, at the core of c5, as it is typically deployed, is an assumption that there is only one, always one and exactly one active action at any time. Of course, creatures constructed with this approach are free to have multiple c5 action groups operating in parallel — and we have seen such creatures in *alphaWolf* and even the colony as a whole in *Loops* — but in Diagram we are particularly interested in this problem of multiple action selection in order for

these multiple actions to be coordinated. Specifically, it isn't clear that delegating multiple action selection to multiple independent and continually overlapping groups isn't simply deferring a problem of coordination rather than solving it.

My approach is to disassemble the c43/c5 + action-tuple organization and then reassemble it in a slightly different order, ultimately exposing its internals as a diagram-like representation.

While the action-tuple is a convenient shape to draw and a convenient chunk to think about we will have to split it into two to prepare it for multiple simultaneous actions (additionally, in the overview of c43/c5 we already saw that an action-tuple had to expose its .trigger() .value() and .doWhile() methods separately. The unity of the action-tuple is clearly already under attack).

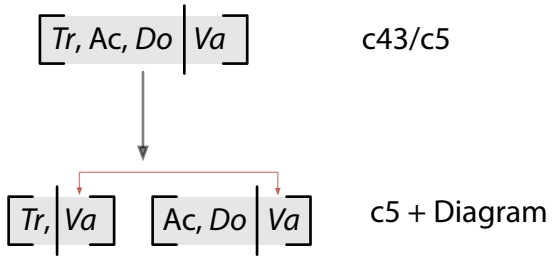


figure 82. The action-tuple is split up into two parts, a trigger factory and an produced action.

The *trigger* becomes, in addition to the place where we obtain the instantaneous relevance and expected value for the action, the factory for the *action payload* — an object that, when asked, is capable of producing an action which then exists independently of the *trigger*. This formalizes the split described above— although many factories and payloads collaborate after creation — and it allows triggers to effectively "group" parametric actions together and possibly instantiate multiple versions of them simultaneously, or even just one after another.

The second modification to c43/c5 is to explicitly allow multiple simultaneous actions — assuming the trigger factories are willing to allow it. We define another object that represents the *action budget* — the number of simultaneous actions that this action group is willing to maintain. With the help of this object, the core c43/c5 proceeds as normal — the central heuristic of only selecting actions when the action-group's triggers and do-whiles have changed both significantly and relevantly remains, only here an action selection may result in an action-tuple being added to the active set rather than replacing it, or the out-

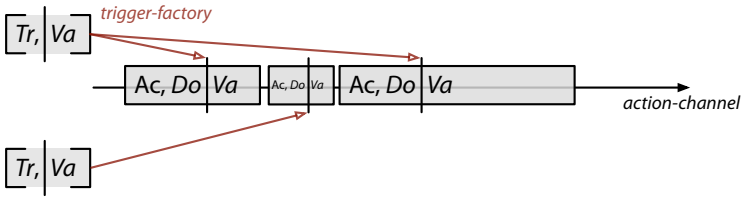


figure 83. The trigger factories schedule produced actions into a channel which is then read out over time to execute the actions.

going, loosing action being deleted from the active set according to the demands of the action budget.

The final twist is to both store and present the results of the action-selection strategy in a publicly accessible channel. Running actions are markers, triggers produce markers when they are selected for. Indeed triggers do not simply instantiate actions, they **schedule** actions by writing into the channel, and are free to place the results of their winning selection in the future. All storage concerning whether an action is or is not active — in particular, for the purposes of maintaining this "budget" of multiple actions — is maintained with respect to this channel, and a number of other, auxiliary and independent processes can act on this channel without fear of disturbing the action-selection process. These processes fine-tune, realign, filter, recombine and even delete the scheduled actions.

Many of the subtle timing and organization issues of the c43/c5 selection algorithm either evaporate or become explicit and visualizable rather than implicit. Through the trigger / action split we have made explicit the possibilities for actions to participate in more than one group and now have a focused location for more complex cross inhibition. Through the open channel / marker representation, temporal coordinations that would have to be implicitly coaxed out of the temporal dynamics of the triggers and do-whiles can now be specified as quite self-contained processes orthogonal to action selection itself — and we shall see many examples of processes that cut across the results of action selection, clean them up and constrain their relationships.

These post-selection modifications need not be kept strictly downstream of the trigger instantiations — by pushing all of the action-selection and budget storage into the context-tree we can even allow for speculatively executed, *page 231*, action selection, and more usefully, action instantiation. An example: triggers

The c5 + Diagram Action Selection Algorithm

state (all state marked @dynamic)

startles — a set of action triggers that get special privilege to interrupt others.

triggers — a set of action triggers inside this group.

lastValues — a mapping from tuple to real number

budget — an object that will handle the balancing of the action budget and hold the "list" of current actions

algorithm

if the maximum *tuple*.expectedValueOf() over all of *startles* is greater than zero then
nextOffer becomes the
greatest of these.

▸ otherwise,

construct the new map *nextValues*[*trigger*] = *trigger*.expectedValueOf() for all *triggers*

if any trigger has *nextValue*[*trigger*]>*lastValues*[*trigger*] and
nextValue[*trigger*] > 2* min(*budget*.currentlyActive()) then select a new action

if *budget*.isUnbalenced() then select a new action

▸ if we need to select a new action:

if all of *nextValues*[...] = 0 then nothing is done

▸ otherwise,

sample *nextOffer* from a normalized version of *nextValues*[...]

success = *budget*.offer(*nextOffer*)

▸ if success:

nextOffer.instantitate()

and set *lastValues*[...] = *nextValues*[...]

can retract their instantiated actions, "unscheduling" them from the channel. They might do this because, after the application of the processes that would clean up the scheduling of actions in the channel, some condition is not true. The only signal propagation out of this speculative execution "closure" is the the trace that the *trigger* should not attempt to re-fire.

Finally, by creating a persistent trace of the action execution, we formalize some of the ad hoc nature of the deferred credit assignment that we found necessary for trainable characters, *page 87* (a direct use of this flexibility, *page 343*); one can imagine other processes scavenging inputs from the history of execution — the channel of markers past.

In general, this c5 / channel actions selection system, together with a battery of post-selection filtration and manipulation processes, can be read as hybridizing a "reactive" approach to action selection with something that might seem more "deliberative". Indeed, in the separation that coincides with the interactionist / classical AI division, both sides of this debate have constructed systems with impoverished vocabularies concerning the patterning of time. Reactive architectures, even more so than pure c5, allow the precise temporal patterns to emerge out of the interaction of tens or hundreds of elements — while techniques to control the speed with which these systems change their selection often earn them places towards my "low temporal complexity" and "low temporal uncertainty" axes, precise and certain control over patterning can become a quite complex affair. On the other hand, classical AI systems — which can have strong symbolic representations of time, and allow sequencing and planning — have had relationships with "real-time" that are often tenuous at best. The c5 / channel action selection technique combines a reactive, an interactive, action selection algorithm with a short window where the temporal patterning strengths of deliberation, "planning" and sequencing can be brought to bear.

Although it should be clear that, at least in principle, the availability of post-selection, *ad hoc* filtering processes increases the both expressive range of action-selection within the c5 toolkit and the vocabulary available to the agent author for articulating that range, it is worth pausing to review what such an extended action-selection algorithm might have meant for the *alphaWolf* project. The Diagram framework explicitly treats many of the problems that caused profound complexity in the creation of both the action systems of the wolf-pups and its interface to the motor systems.

Firstly it is important to realize that a pup action system is in fact an example of a multiple, parallel action system — each consists of an "attention" action group and a "main" action group. However, while the execution number and order of these units are fixed by the limits of the toolkit at that time (the attention group always runs first, and both groups only select once), the order that makes sense for the problem that the behavior designer is trying to solve changes depending on the part of the interaction being authored. Sometimes what the pup should pay attention to is limited by the action that the wolf pup is performing (e.g. it should look at the thing that it is fighting); at other times the actions that ought to be performed are constrained by the object of attention (e.g. the pup can't fight a navigational marker on the ground supplied by the person directing the pup). What raises the stakes of the problem is the paucity of ways in which these two groups can be coupled — the first group to happen to make a decision gets priority and one ends up fighting against the protection against dithering inherent in c5 — the mechanisms to bias the rolls of the dice of these action groups are misused to ensure that the right set of dice gets rolled first. This is the spindle around which the complexity of figure 26, *page 96*, weaves itself.

In Diagram the coupling is much more easily expressed as a filtration process on the results of a single, much simpler action group. Weights or biases on the trig-

gers of actions remain just that — gone are the additional triggers that manipulate what group of actions get a chance to select. A scan of the ultimate behavior system file deployed in *alphaWolf* shows that, in the absence of short-circuiting triggers that act in this fashion, the complexity of the action selection process, as measured by the number of triggers connected to all action tuples, reduces by 60%. Additionally, it appears that several action tuples, that act as silent placeholders for actions that subsume control over both groups, would disappear entirely.

Secondly, and more speculatively, are the other hypothetical simplifications that Diagram could have offered the designers of *alphaWolf*. Most significant is the ability for the channel mechanism to represent, schedule and revoke sequences of actions. As I have stated, *page 98,* the creation of chains of actions and the deferral of actions while others run seemed fundamentally difficult in *alphaWolf* — difficult to both express, and to express without damaging the action-group's ability to prevent dithering. Sequences where pups explore their environment, by moving between random locations; where adults move away from the pack only to return later; and indeed the primary interaction of moving towards a pup, fighting it and winning or losing, form the backbone of this kind of directable agent's "scriptability". The deferral of directed escape while being attacked also suffers from being only indirectly expressible inside the wolf-pup's action system — a set of triggers that "latch" user interaction can be seen throughout the code. Finally, it might have been possible to unify the top-most layers of the wolf-pups' motor system with parts of the action system in a single set of Diagram channels. The patterning of the motor programs of *alphaWolf* often reached points of extreme complexity due primarily to the navigational complexity of the environment (the need to move pups towards moving targets). If both scheduled actions and the resulting motor programs could have "seen" each other, and more importantly been seen by the navigation system, issues of persistence — either running actions until the pup actually arrived at a point where

interaction could occur or eliminating attention switches made impossible by the current motor programs — could have been avoided.

While it remains, impossible to verify these claims of the utility of the Diagram system for the *alphaWolf* installation, it is equally clear that if these problems can arise even in a system in many respects ill-suited for their treatment, it will not be long before they arise in other works. Indeed, in *how long...* and *22* we shall see ghosts of the complexities of *alphaWolf* appear again — "modal" action systems that move through constrained phases, often patterned by the flow of time through the work; actions, be they the drawing of lines, or the manipulation of existing graphic elements, that unfold over small, scripted periods of time; and a general blurring of the boundary between action selection and motor patterning. Prior to deploying Diagram in earnest in my pieces for live dance, however, I completed an installation that was very much about the complex and "precise" patterning of time — a live composition, *Loops Score*.

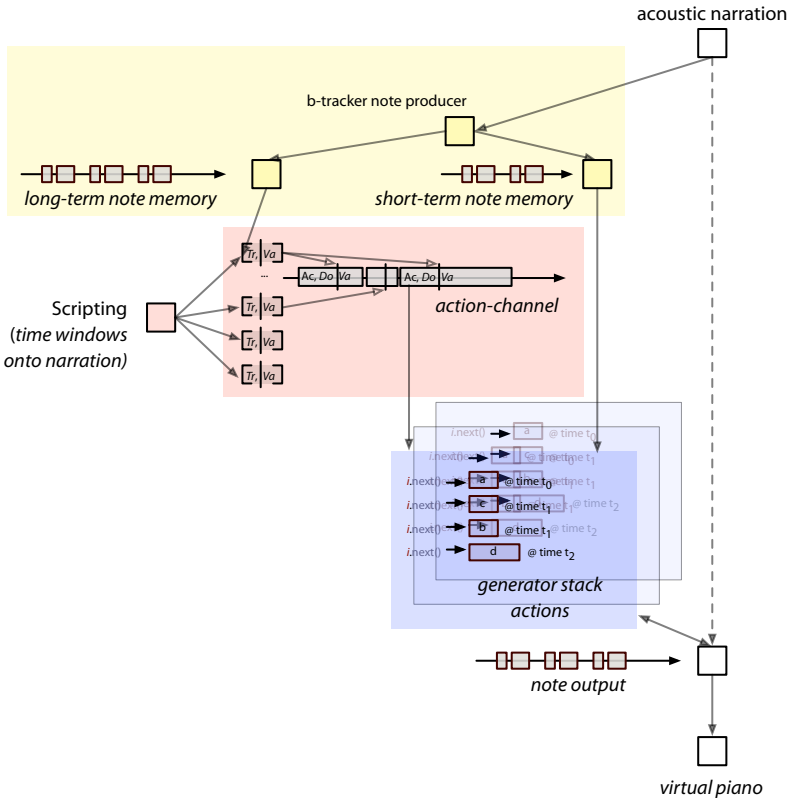


figure 84. *Loops Score* agent overview. Each of these boxes will be "unpacked" in the main body of the text.

Loops Score is the music that was made to accompany *Loops*. It was constructed by analogy with the visuals two years after *Loops* premiered. Just as *Loops* constructs a set of interacting processes that observe and recast the motion of Cunningham's hands, *Loops Score* takes a set of interacting musical processes that listen to and restate the sound and language of Cunningham's narration. Cunningham provided a twenty minute recording session, independent of the motion capture session, consisting of him reading from his 1937 diary — his first visit as a young man to New York. The sonic palette for the work was found in a high-fidelity sample set provided by the John Cage Foundation of a prepared piano, prepared in accordance with Cage's instructions accompanying his sonatas and interludes. By selection, filtration and pitch shifting this prepared piano was turned into set of seven pianos, each with a different, but radically expanded, timbal range.

Unlike *Loops* there is a single agent at work in the production, and, in contrast with *The Music Creatures* this agent has no visual form and its musical output comes without virtual movement analogies, but simply as a set notes sent to a set of virtual pianos. However, *Loops Score* was a work that continued many of the technical themes that *Loops* started, in particular: what it might mean to score one or more action systems and how might one create a work that navigates alternating layered structures of emergence and control. In particular the ability to general complex structures in channel / marker representations and then have competing tasks slice across these channels, organizing and culling them, is in itself the layer of open emergence and detailed specification.

The Music Creatures was a unique exploration of the possibilities of relationships between the body of an agent, the sound that it makes and the sound that it understands. But as far a mainstream computer music is concerned this ap-

peal to virtual physicality was a digression. *Loops Score* approaches computer music more directly. Where *The Music Creatures* have agents with complex perceptual and motor skills and simple, almost script-like developmental action systems glued together with a few automatically learned parameters, *Loops Score* is on ground more common with the traditions of the interactive music work: the complexity is in the scoring, and in the interpretation of this score — specifically the "action system" of the agent. There is, no doubt, a future installation to be made that has comparable richness in each of the three parts of the agent decomposition, *page 441*, but taken together I believe that these works do a good job of creating a preliminary sketch of that installation's territory.

The open process score

Early in the creation of this piece we rejected the, perhaps rather Cunningham-esque, idea that the meaning of the words might be technically unrelated to the processes that act upon the sounds of them, opting instead to find the "process score" for the music out of the text itself. The search was to find structures in the text that had both musical and linguistic function, that were 'half way' between forms found in music and forms bearing linguistic meaning. A number of forms were found, and each form was constructed as a loose template. They included **lists** inside the text both large and small; **comparisons** and spatial relationships; markings of the **passage** of time; **returns** to previous locations.

These templates operate on the word level, and one could perhaps imagine given a robust enough speech-to-text system performing this template-matching in real time. This was not attempted and not required given the finite and predetermined narration, but remains an enticing possibility for future work. Rather, an analysis of the text of the narration was coupled to sound of the narration through marking of the onset and offset of each word in the narration. Thus for

each present atom in each instance of each template, we can provide a time period this atom is "listening" to the sound of the narration.

Each of these structures, gathered from the text, marks a channel-based *script* which opens windows for actions to occur. Actions inside *Loops Score* are triggered by the presence of narration material inside these windows; in the end some 150 windows are marked on the score, cross-linked to sixty-five actions. The vocabulary of possible actions corresponds metaphorically to the kinds of structures that trigger them: **list-actions** repeat their triggering elements while searching for a stable rhythm inside the elements; **comparison-actions** state their elements and then emphasize the differences between them; **passage-actions** state their first element and then continue to look for material that is sonically related until the close of their marked passage; return-actions compress material from their triggering element all the way back to where they "came from". The actual programming of these musical cells is constructed using the manipulations of the marker / channel structure that will be described in the following section. These actions schedule the notes to be played into a channel which is itself exposed to the actions. Rather than "ballistically" writing the notes to be played to the scheduler, they opportunistically align, modify and perhaps even fight for space and relationships "on output".

Since the script is densely packed with overlapping processes, these actions compete using the action-selection mechanism described above. The budget for actions is dynamically set based on a smoothed version of the amount of musical material that eventually makes it out of the agent — providing a long-term, self-regulatory production.

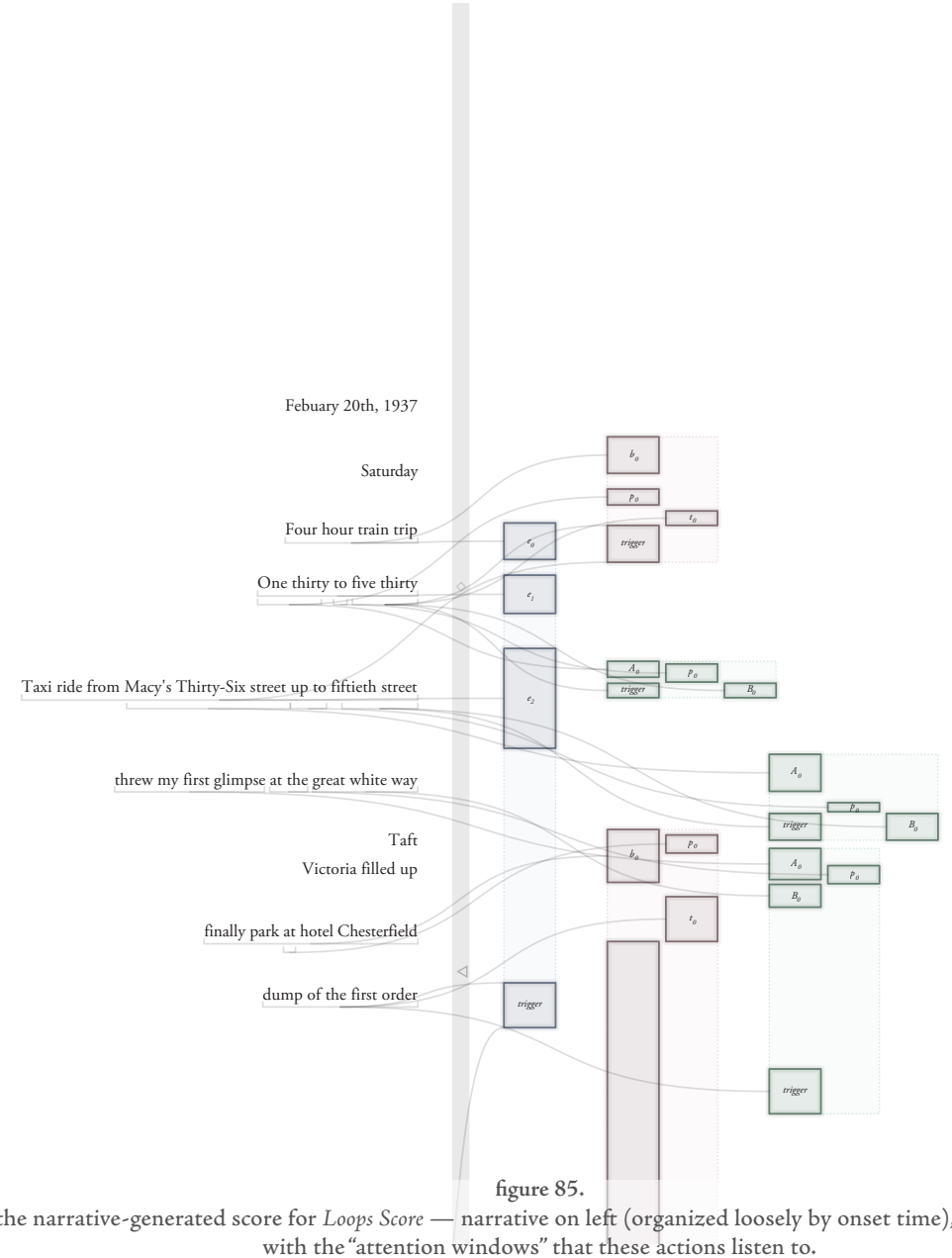


figure 85.
An extract from the narrative-generated score for *Loops Score* — narrative on left (organized loosely by onset time), potential actions on the right with the "attention windows" that these actions listen to.

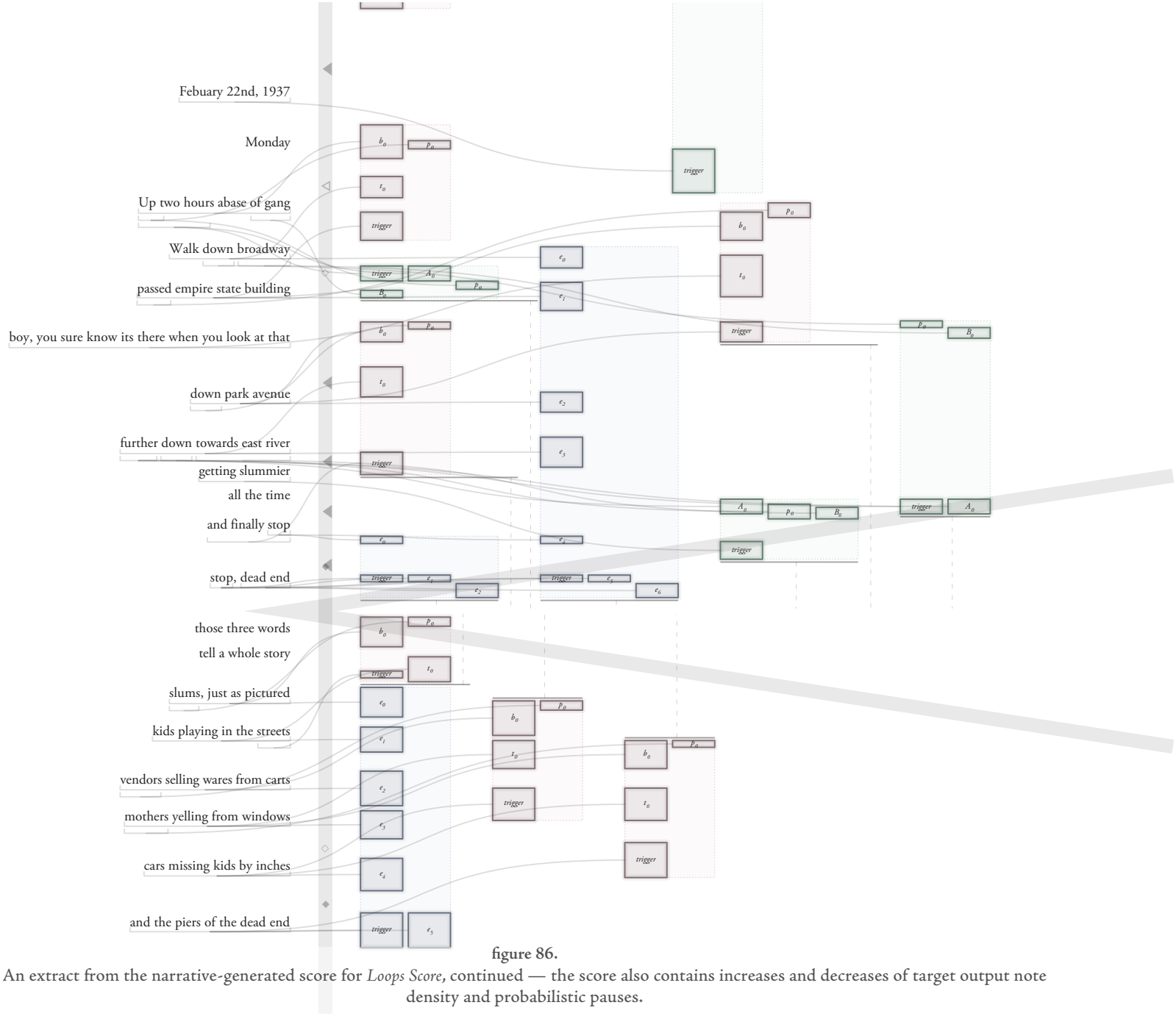


figure 86.
An extract from the narrative-generated score for *Loops Score*, continued — the score also contains increases and decreases of target output note density and probabilistic pauses.

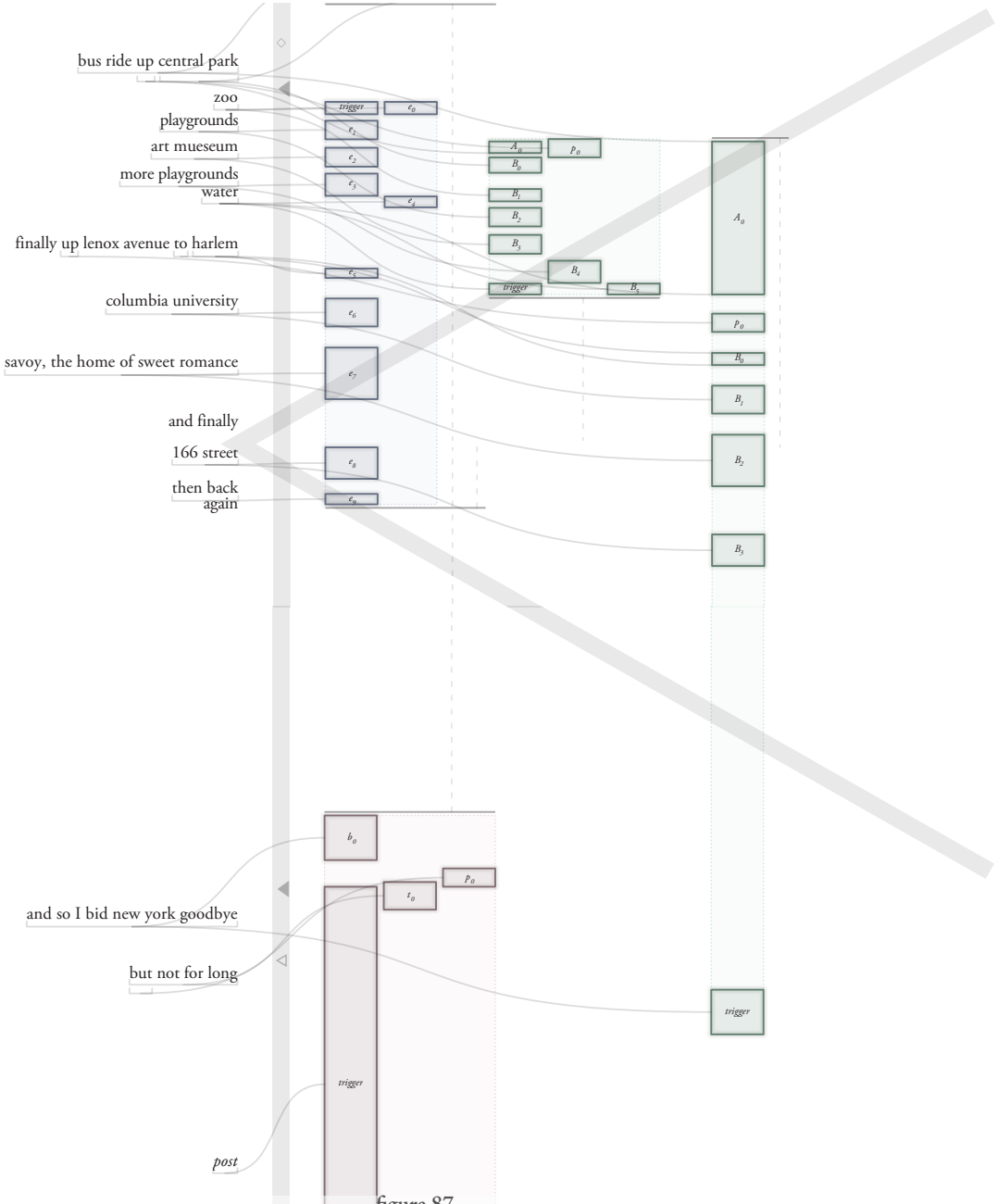


figure 87.
An extract from the narrative-generated score for *Loops Score*, continued — upon ending, the score loops. However, the memory of the notes played is made available to the processes in the next iteration, allowing actions to trigger in advance of all of their attention windows.

Thus, the agent of *Loops Score* exists in a perceptual world dominated by the sound of the narration annotated by this script. The perception system here transforms the audio into a series of overlapping note events. And we find ourselves again in a problem domain half way between that of music and that of speech. The conversion of raw unstructured audio to quantized notes is a problem that has, of course, received considerable attention — it is in essence the inverse problem to the forward task of synthesizing sound from a score. And, particularly in monophonic worlds, solving this problem is often an important initial step in interactive music systems. However we are in an adjacent but different domain here — converting speech rather than music to musical notes — a less grounded domain. The coarsest version of this problem might be the extraction of prosodic contour and the segmentation of voiced and unvoiced parts of speech and this too has received some attention. Our goal then is more musical detail than that afforded by speech-based approaches, and less musical fidelity to a ground-truth with a more complex input than polyphonic music-based approaches.

For what I believe to be the state of the art in this problem, **A. T. Cemgil**, *Bayesian Music Transcription*, PhD dissertation, Radboud University of Nijmegen, 2004.

One of the earliest score followers used unstructured audio: **B. Vercoe, M. Puckette.** *Synthetic Rehearsal: Training the Synthetic Performer*. Proceedings of the 1985 International Computer Music Conference. San Francisco: Computer Music Association, 1985.

This pitch tracker is an implementation of the lightweight voice-pitch tracker: **L. K. Saul, D. D. Lee, C. L. Isbell, Y. LeCun**, *Real time voice processing with audiovisual feedback : toward autonomous agents with perfect pitch*, Advances in Neural Information Processing Systems 15. NIPS 2002.

Because of our need for musical accuracy we forsake the simplicities of a pure-monophonic pitch tracking-solution. We recast this perception problem as a as a tracking problem, tracking peaks on successive overlapping Fourier transform frames, seeding our b-tracker ongoing model population with the results of a lightweight monophonic pitch follower. We use the b-tracker perceptual framework to track these peaks and convert them into musical "notes".

To use the b-tracker framework we need to supply the following underlying process details: individual tracking hypotheses are represented as individual frequencies F with frequency "velocities" F_v and amplitudes; they predict that the next Fourier frame will contain a strong peak at the same frequency, and a

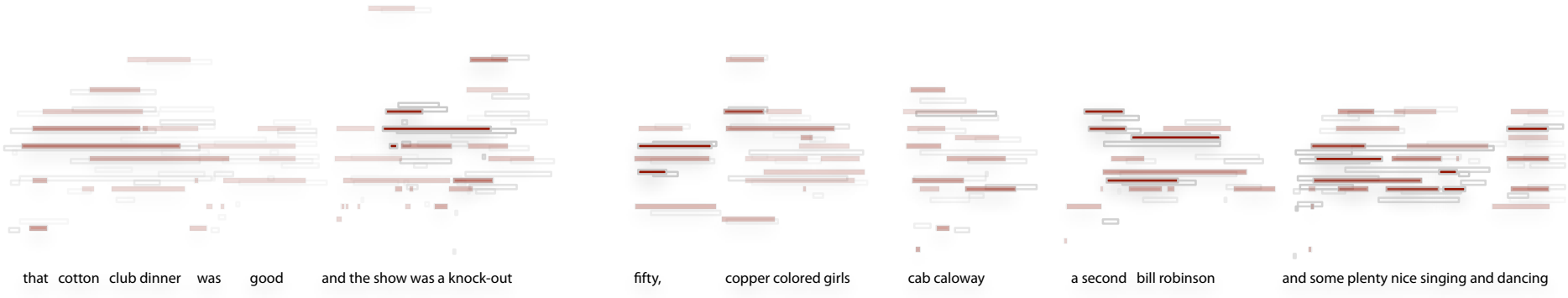


figure 88. The notes created (grey) and filtered (red) from the b-tracker analysis of the short-time Fourier transform windows from Cunninghams's narration.

frequency one Fourier bin higher $F + R + Fv$ and one bin lower $F - R + Fv$; thus the untrimmed branching factor of the forward search is three. Successive Fourier frames are overlapped by a factor of four, to provide precise frequency information in the case that only one frequency is present in a bin. In the case that multiple frequencies move in and out of a bin we'd expect the kind of crossing hypotheses that can be disambiguated with the frequency velocity information. The monophonic pitch tracker constantly seeds the tracker with hypotheses at its output pitch, should it determine that voiced speech is actually taking place.

Once a hypothesis has survived the culling process of the b-tracker for four successive frames (four frames overlap a single location), it has the opportunity to emit a note. To do this we need to convert the pitch history of the hypothesis to a musical note. Since we have no underlying pitch grid from the underlying sonic material we have to adapt one as we move along. We can represent this pitch grid as a set of hypotheses that get adapted by the pitches emitted by the lower level tracker.

Setting the size of the bins to be multiplicative increments of $2^{1/12}$ gives us a adaptive chromatic scale. Choosing larger increments gives us access to poten-

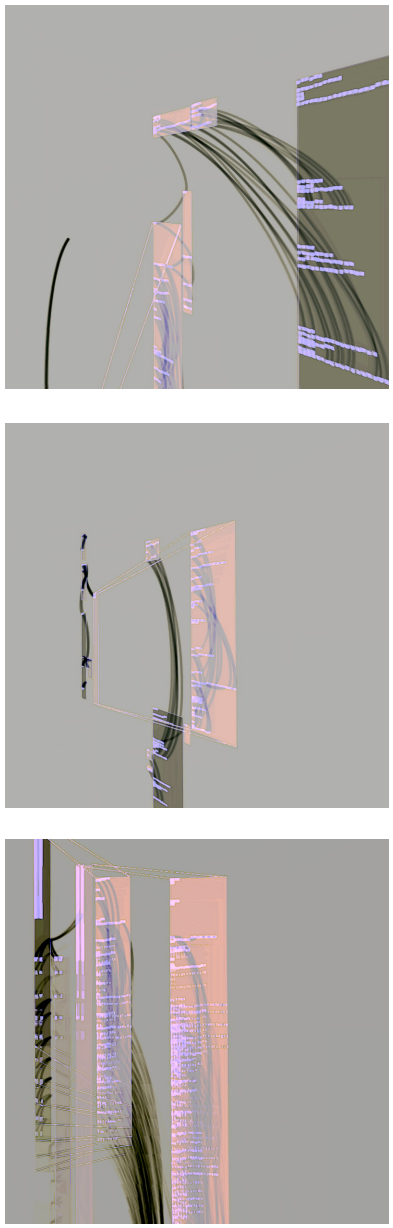


figure 89. The diagram visualizer (which showed alongside *Loops* and *Loops Score* in a separate, synchronized interactive kiosk). These images are quite literally the markers and channels generating the music.

tially interesting hybrid modes — *modes* because the scale is typically quantized more coarsely than a full evenly-tempered chromatic scale, *hybrid* because the locally coarse quantization grids are allowed to be globally misaligned. This increment, and the speed of adaptation / forgetting in this layer are free parameters. And in a few places the score forces a flushing of this memory — a modal break accompanies the change of day in the narration. This granularity is easily expressed as a "cleanup" process on the ongoing model stage of this b-tracker implementation, *page 189*.

These hypotheses, labeled with note values, amplitudes, and ultimately with durations, are the "output" of the perceptual layer of the *Loops Score* agent. These hypotheses are injected into a short-term memory (of around five seconds), which maintains the full merge histories of the hypotheses and a long term-memory (of around thirty minutes, twice the duration of the underling narration) which maintains just enough information to write a musical score.

This resulting post-perceptual surface is thus easily presented in the channel / marker representation. This is the raw musical material that triggers and enters the actions of *Loops Score* and my discussion will now turn to how the raw *algorithmic* material of these actions are constructed. These materials, although introduced here in a "computer music" use are all specific only to the marker / channel representation and the fundamental support that Diagram offers — they are *of* music, but not musical themselves. They form the basis for the odd "musicality" of movement and interaction that I have sought in my dance pieces and beyond.

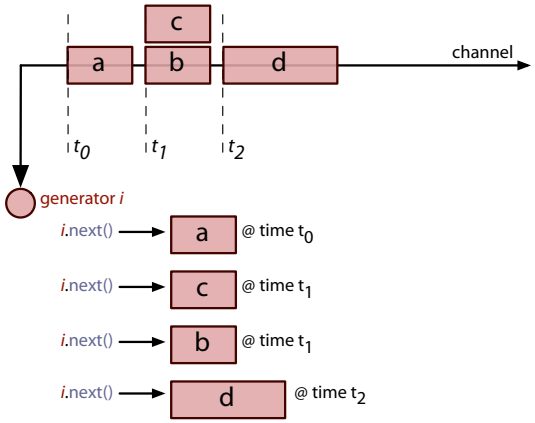


figure 90. Converting between channels and generators is easy. Here a generator "reads out" a channel in order.

The most important is a diagram channel *generator* — this is an extension of the co-routine / resource framework introduced for *The Music Creatures, page 156*. In that work the framework was constructed to allow small imperative programs to run concurrently while allowing cross-program interaction and dependency in the shape of resources. Typically, these small programs got one chance to execute (or "update(...)") per update cycle of the music creature in which they were embedded. Here we allow our co-routines to "return" a sequential series of diagram channel markers, and move programs forward not once per update cycle but until the markers that they start returning reach a certain point in the future. These programs are responsible for keeping the diagram channels' description of the future filled up and are called upon to generate more material, if they can, when they are needed.

Musical material generating processes, both simple and complex, can be written inside this style. For example (this time in Python, the Java is similar to the co-routines previously discussed):

```
def scale(start, step):
    n=start
    while true:
        n = n + step
        yield noteMarkerFor(n, quarterNote(n))
```

produces an ascending chromatic scale. Parallel and series composition of processes are achieved using the same kinds of continuation-composition that the co-routine / resource framework already allow.

```
def contrary_motion():
    yield parallel( (scale(0,1), scale(100,-1) )
```

Clearly, we can turn channels into generators quite simply (this generator is

nothing more than sorted channel marker *iterator*), and we can turn generators into (potentially infinite) channels through rendering out the markers that appear from the generator until a sufficient time in the future is reached. Nothing prevents generators from returning markers that do not increase in time along the channel, although such a practice is discouraged. "Flattening" generators can be applied to processes that cannot produce their output in time order, with increasing levels of latency (i.e decreasing levels of minimum future) being introduced for increasing levels of safety.

Generators and channels are complementary: generator-level composition is particularly good at producing memory and time-efficient processing of action-level musical manipulation; channel-level manipulations are good at producing transformations that cut across many channels or many time epochs and are suitable for memories and intermediate buffers.

The goal was to create from such a language framework a system that allowed the rapid development of incremental, real-time musical processes that contain as little latency as is needed to maintain their computational integrity inside a dynamic environment but no less — a system that blends the interactivity of reactive systems and the complex multi-temporal scheduling and planning of material that more deliberative systems achieve.

To flesh out the description of how this framework was used for *Loops* Score, we need to give some kind of overview of both generator-and channel-level manipulators from which the actions of the agent were created. Since, as premiered, *Loops Score* used thirty-five primitive generators, eight channel manipulators together to make three versions (each a generator) of each of the four sub-linguistic templates described above, we should group these generators and manipulations together into some kind of taxonomy.

An **abstract balance** is a generator-level Diagram facility that takes a channel that is having markers added to it and filters these events such that a target event rate is met — it requires that something, perhaps the markers themselves, can provide a scalar *value* for a marker. We have seen something similar, but less dynamic, in the persistent long-term learning of *The Music Creatures*, *page 143*. The task is to find some horizontal partitioning cut-off for a channel such that the rate is maintained if markers that are "bigger" that this cut-off. If we have a sorted list of N markers m_n in the source channel we place the cut-off $c_{\alpha,N}$ that lets the top α fraction through at position αN or, more accurately:

$$c_{\alpha,N} = m_{\lfloor \alpha N \rfloor}(1 - \alpha N + \lfloor \alpha N \rfloor) + m_{\lceil \alpha N \rceil}(\alpha N - \lfloor \alpha N \rfloor)$$

In practice the distribution of marker-values isn't necessarily stationary. Handling arbitrary non-stationarities is, of course, arbitrarily hard. The following gives a variable forward momentum generated by recent history:

$$c'_{\alpha,N,\beta} = c_{\alpha,N} + (c_{\alpha,N/2} - c_{\alpha,N})\beta$$

for $\beta > 1$ this extrapolates out how the cut-off is changing with time. It's easy to modify this approach further to prefer more pessimistic or optimistic (or rather high-value or low-value) extrapolations.

The balances can turn a channel of valued-markers, which might represent actions or perceptual events, into event streams with particular general *rates*. Often, it seems, it is easy to come up with a good metric for perceptual events, but much harder to understand how this metric transforms the underlying temporal behavior of what it applied to. Without these indirect connections, such as the abstract-balance, one begins to start tuning the metric itself in response to

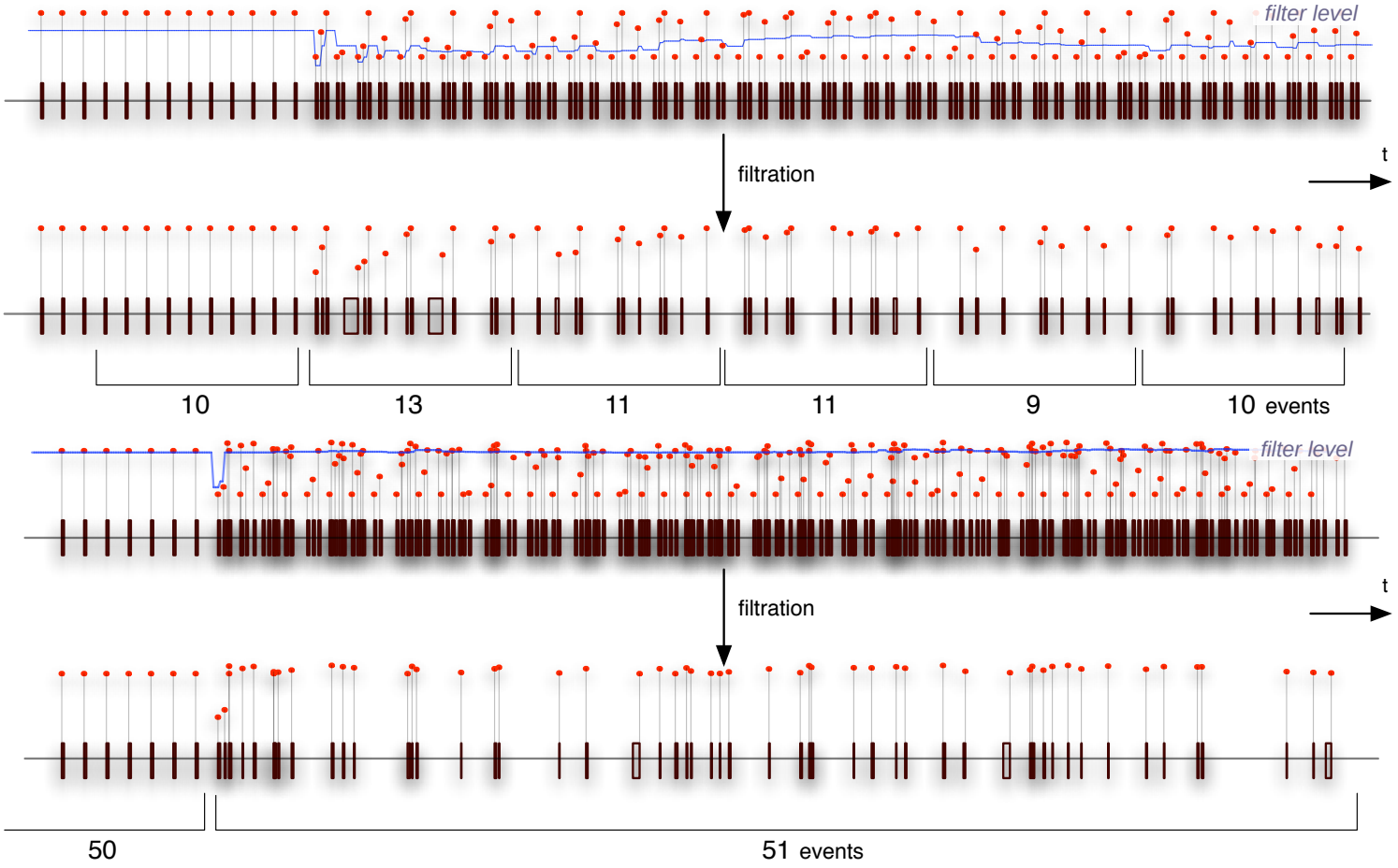


figure 91. The abstract balance correctly adapts its threshold to maintain a very event similar rate while capturing only the highest value events.

the temporal dynamics of the material that it is exposed to. That these connections should be more indirectly, yet more explicitly specified is argued strongly in the development of *The Music Creatures, page 175*. Such a blurring of principle (the metric) and pragmatic deployment (what it happens to be used on) is a generally unacceptable level of coupling between two parts of a system and is often how a commitment to a particular input or processing of input gets buried deep within a system that consumes it.

The complete abstract balance has two more parameters — a maximum latency and a minimum refractory period. The first controls the "look-ahead" window size and controls how far back in time the generator will be returning events. High latency will allow a better tracking of the target rate on highly non-stationary input. A minimum refractory period blocks events from being generated too close together, and, further, masks these events in the memory (the sorted array above) such that they do not feature in the computation of the threshold.

Generator-level operations - filtration / perceptual partial re-tracking

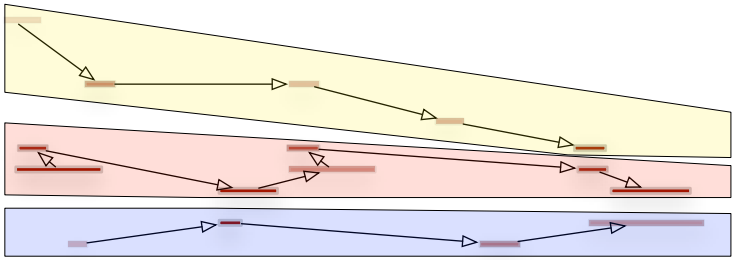


figure 92. A fragment from the short term memory of the *Loops Score* agent "re-tracked" by a new b-tracker process, segmented into three registers, corresponding to three b-tracker hypotheses. Note that no constant segmentation threshold would give this result.

Another generator-level manipulation of the musical material captured from the transformation of Cunningham's narration is the perceptual stream tracker. This layers another level of the b-tracker framework on top of the musical material — a much coarser level than the original partial analysis was conducted on — that produces a number of generator streams of material. Each b-tracker hypothesis is a perceptual stream — a pitch value, and a pitch momentum. Although a number of attempts at musical perceptual stream segmentation are present in the literature, the existence of the b-tracker framework makes the implementation of another perceptual stream follower almost as simple as filling out a form: a hypothesis representation (pitch, pitch momentum) = (p, m), a distance metric between hypothesis and pitch class datum d $|p+m-d|$, a hypothesis predictor $p \leftarrow p+m, m \leftarrow \alpha m$. Now further generator-level manipulations can be performed on the lower (the lowest good hypothesis) and upper (the highest good hypothesis) lines of the transformed material.

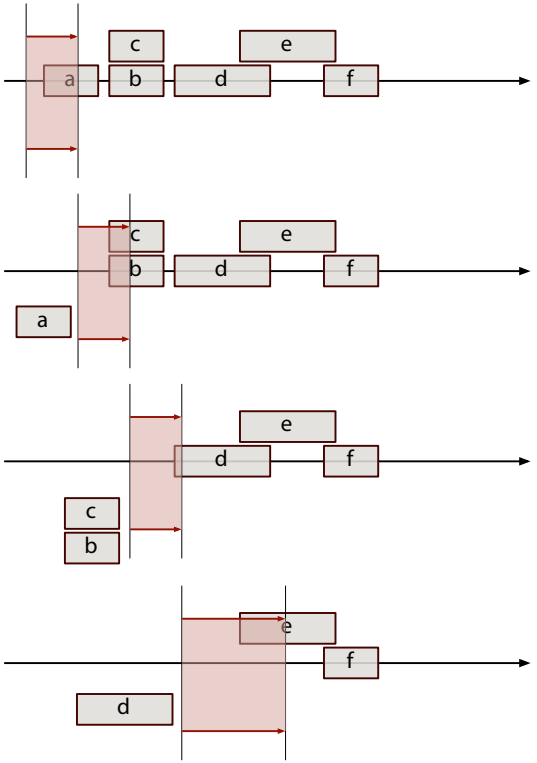


figure 93. The rolling culler is responsible for the past "forgetting" of markers in channels. It correctly handles both variable update rates and moving markers.

One simple, but ubiquitous channel-level manipulation is the rolling culler. This manipulation can be put to two uses, the first is to incrementally remove from a channel markers that have fallen past a particular time horizon, freeing up the space allocated to them and allowing the channel's memory to forget them. The second use for this rolling structure is to read out, according to some time-base the contents of the channel — like reading a score, (or playing a Fluid score, *page 393*) — by intersecting the time-base interval between updates with the contents of the channel. This is, therefore, one way of turning a fully laid-out channel into a marker generator. Rolling cullers are used to run the Diagram framework into a scheduler of music (with note events as markers) or an organizer of movement (with pose-graph or other instructions as markers).

In *Loops* Score, Diagram markers represent actions, planned out in time, that when traversed by the moment, result in notes being played, or parameters being set. The abstract balances are ways of culling sets of actions, based on criteria — metrics of value — thinning them out to a particular rate. There are other ways of thinning actions, that are more important if these channels are going to be coupled with other channels.

As a representation of future, scheduled actions, the Diagram marker channel appears to notate a case where actions are independent and that, baring any constraints, markers can be swapped or dropped independently. What if actions remained atomic, but were able to form molecules inside their channels?

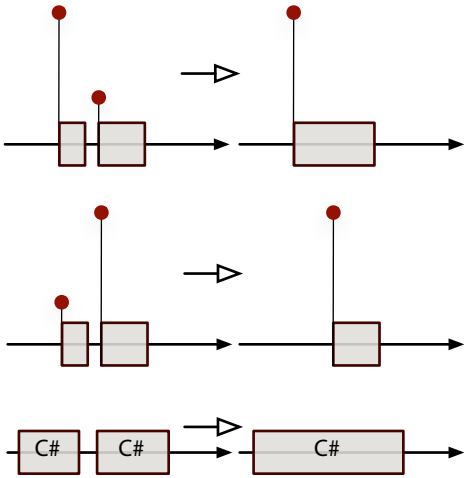


figure 94. Three example fusion filters take from the text.

A Diagram fusion filter takes a perspective onto the markers in a channel and looks for patterns that fit templates that, once matched, cause the replacement of the components with a new marker or markers. If we can find a succinct way of specifying these channel "chemistries" then they offer a powerful way of filtering or developing channel contents.

We can write production templates for perceptual phenomena:

forward masking: $|_{t_0}\text{loud} + |_{t0+\Delta}\text{quiet} \rightarrow |_{t_0}\text{loud}$

backward masking: $|_{t_0}\text{quiet} + |_{t0+\Delta}\text{loud} \rightarrow |_{t0+\Delta}\text{loud}$

local quantization cleanup: $|_{t_0}\text{c\#}|_{t_0+\Delta}\text{c\#} \rightarrow |_{t_0+\Delta/2}\text{c\#}$

On the left-hand side there is a template to be matched — certain markers with certain properties with a particular temporal relationship (and in particular, within a certain window of time); on the right hand side are the markers produced. This notation is not complete — it does not specify whether intervening markers are either ignored or prevent the match from occurring.

Programmatically we are free to construct these windowed template recognizers a number of ways and many recognizers are simply coded "by hand". We shall see later a less general, but more compact notation for describing these templates, *page 278*.

There are other abstract production calculi that are useful to define in general. One models a species of marker whose action is to set a variable to a value:

redundancy deletion: $\left[\left|_{t_0}(x \leftarrow a) + \left|_{t0+\Delta_0}(x \leftarrow b) + \left|_{t0+\Delta_1}(x \leftarrow a)\right] \rightarrow \right|_{t_0}(x \leftarrow a)\right.\right.$

for: $\forall a, b$

Such filters, applied over short windows, remove transient values that are set and then unset from a stream.

Such techniques fall very firmly within the domain of production systems and, as written, their use in either AI is far from new. However, in important previous uses of production rules in AI the goal has been to create a complete system using this structure — essentially recasting the complete action selection and / or motor system problem in terms of competing or ordered production rule systems. In the framework here, the production system is not the material from which other system are constructed, rather they available to other structures and representation. What is different here is the strategy not the tactics, the framework that this idea is embedded in and the principles that govern its deployment.

For example, the C.L. Forgy, *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem*, Artificial Intelligence, 19, 1982 continues to have new implementers today: http://drools.org/Rete

The use of production systems *per se* in generative computer music is a little more ephemeral.

reversible, live and aware of time— no production rule deletes any material, even if it appears to "consume" its triggering markers, these markers remain in the channel (annotated as "consumed") and linked to the material that they produced and the production rule that coordinated it. This allows two important flexibilities: firstly, production rules that have fired are updated live when the details of their left hand sides are updated;

Notification inside the Diagram Framework

The batching of notifications is one form of *deferred method execution*. Rather than, say, the addition of a marker to a channel causing the notification of every system interested in that channel we allow modifications to channels to be bracketed by calls to beginModification() and endModification(). Only at endModification() are notifications propagated (calls can be safely nested). One thing that is important to get correct in this implementation is how structural modifications — deletions and additions — should be handled. Many processes that execute on modifications are much easier to code if notifications are delayed until not only the addition of a marker to a channel has taken place, but the marker has finished having its attributes set and its relationship with other markers configured. Batching, as written above, achieves this delay.

However, for deletion of markers the issue is a little trickier. Again, consumers of notifications typically want to be informed of the impending deletion of a marker before the actual removal of the marker takes place, for it might have information stored with respect to the markers role in the channel. If these consumers only hear about the deletion after the marker's information has been deleted then they must be written to maintain a copy of all of that information. Maintaining this duplicate information, in turn, places a heavier burden on the notification mechanisms. So rather we make it easy to place the final removal of markers at endModification() time — there is a removeAtEndModification(*marker*). This defers final deletion until notifications have had a chance to act.

Finally, deletions need to be fused in the notification batch with any additions that take place to short-circuit notification cascades for markers that are added-changed-deleted or changed-deleted in one single batch. Ironically, such fusion of method calls would be an ideal use of the channel based fusion filters, were those filters not being constructed out of the use of this very notification mechanism.

secondly, production rules can choose to "un-fire" and delete their right-hand side should their conditions for acting cease to exist. This fluidity is vital in the case where the contents of the two sides of the production rule vary continuously, which is the case since the marker temporal positions are continuous, unlike the typical symbolic-level AI production systems.

a domain of low computational complexity — although in general even a carefully ordered set of production rules can become computationally burdensome (and a non-ordered set can, of course, run forever) we note that in most cases there are a number of factors on our side: the rules are not being used to structure particularly deep or broad computation — they are for cleaning, recognizing and embellishing small amounts of incrementally specified material, more complex computations can be achieved by other means; the computation is limited by the present time in one direction and by a configurable time horizon on the other — there is little point computing things far before the present, and far in the future can wait.

The computational difficulties that remain are ameliorated by careful batching of the notifications that cascade out from a channel when it is modified. Indeed, the batching of notifications becomes vital for keeping the whole Diagram system working at high speed.

high availability — one or more fusion filters can be attached to any channel; however, because these filters are often used to filter executable markers they are particularly useful for channels which represent destinations for deferred method calls. This is of such general applicability that we add to the context-tree annotation library for Java a new annotation tag (that triggers load time byte-code injection):

@deferIntoChannel(*channelDescription*, *parameterDescription*)

this tag marks a method (which must return void) as being deferred into

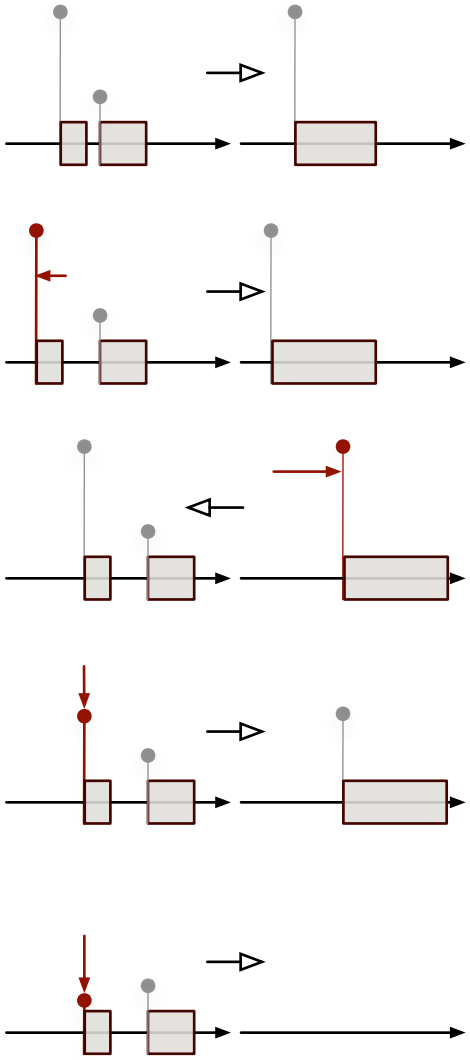


figure 95. By an intricate network of notifications, fusion filters are happy to collaborate with changing marker inputs, changing their output positions or attributes, or ultimately retracting their products altogether. The relationship is bidirectional.

a channel. Calling this method no longer results in the method body being executed; rather, a marker that will execute the method body is created and inserted into the channel at a particular time in the future. *channelDescription* points to a class that describes how to find the channel in question and the channel's time-base (this is typically either from the context tree or from a fields in this instance). *parameterDescription* points to a class that describes how to take the parameters to this method and transform them into attributes on the marker — attributes that will presumably be read and matched by fusion filters and other channel-level manipulations.

Fusion filters and @deferIntoChannel are the basis for an important part of my tool "Fluid". When visual elements (which can be though of as actions) are activated by multiple processes (which can be thought of as action-groups) we would like them to continue to see a stable life-cycle transition — start(), continue(), stop() and always in that order— despite multiple processes, spread throughout the code-base, starting them, continuing them and stopping them without coordination. The solution is to defer these start(), continue() and stop() calls into a channel and have a fusion filter clean up the overlapping messages into a diagram that expresses whether the action is in fact running or not.

$$|_{t_0}\text{start} + |_{t_0}\text{continue} \rightarrow |_{t_0}\text{continue}$$

$$|_{t_0}\text{stop} + |_{t_0}\text{continue} \rightarrow |_{t_0}\text{continue}$$

$$|_{t_0}\text{start} + |_{t_0}\text{stop} \rightarrow |_{t_0}\text{start}$$

If some latency in stopping and starting actions is acceptable we can elide stops and starts that occur too closely together into unbroken "continues":

$$|_{t_0}\text{stop} + |_{t_0+\Delta}\text{start} \rightarrow |_{t_0 \rightarrow t_0+\Delta}\text{continue}$$

This is useful, perhaps even vital, in the visual programming case where one needs to construct visual programs that are robust with respect to infinitesimal changes of visual element positioning, *page 393*. Away from the visual environment this offers a neat and self-contained (with respect to where the logic is located) solution to the problem of non-reentrant actions having multiple parent action groups. In *how long...* there is the *weaving* agent, *page 371*, whose "motor system" is entirely constructed out of such Diagram channels — the motor actions taken by the creature involve reorganizing a notation of the stage. The interface between the action system and the motor system evaporates, and only the programming language — the method call — remains.

Loops score uses fusion filters at a number of levels. Firstly, to further clean the output of the perceptual abstract balances, implementing coarse perceptual masking effects (for the processes that benefit from capturing a few strong and structurally important notes) and removing repeated overlapping notes from the transformation and sampling of the narrative. Secondly, to propagate, and more importantly repeat-with-modification, these samples of musical material.

Channel-level operations — modified time view

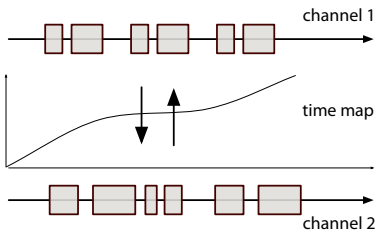


figure 96. Two channels coupled together reflect each other's contents, but present different temporal views — useful for the compressed memory structures of *Loops Score*.

The narration for the *Loops Score* is not a linear story, but rather a series of independent fragments, between which there are pauses of random duration.

The first channel-level manipulation is a simple one — the temporal distortion. This takes a channel and makes a view onto it that has a remapped time. With a bi-directional time remapping, both the view and the source channel are both live — markers can be added to either one and appear in the correct place in the other. *Loops Score* uses such a channel pair as its primary memory of its outputted musical events to compensate for these randomly generated pauses; the material that falls in the gaps between narratives is stretched or compressed when written and stretched or compressed again when recalled.

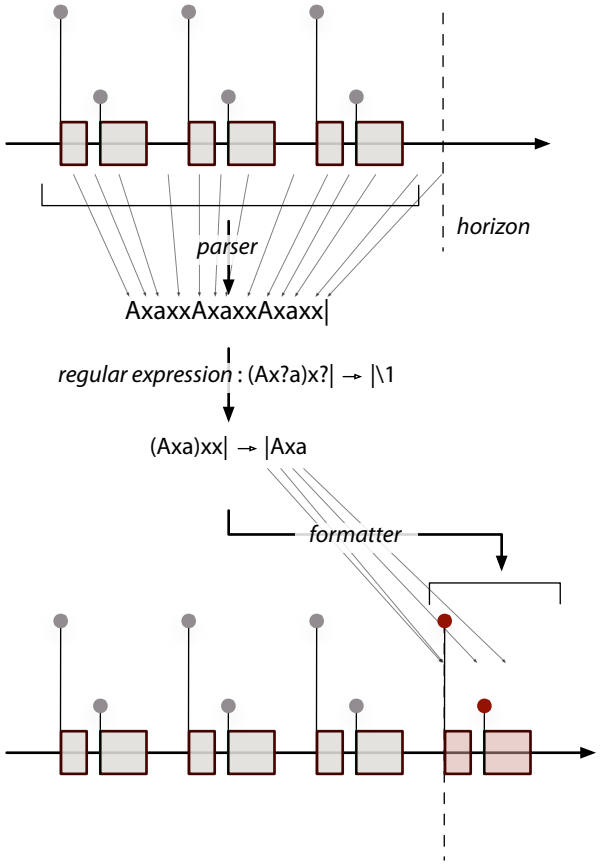


figure 97. An example "continuation momentum" written using a regular expression.

In the fusion filters above I loosely wrote "production equations" such as:

forward masking: $|_{t_0}\text{loud} + |_{t0+\Delta}\text{quiet} \rightarrow |_{t_0}\text{loud}$

We can code the recognizers (the left-hand side) and the producers (the right-hand side) by any means. The only constraints are that the recognizers must work incrementally — in response to batched notification updates from the channels — and producers should work non-destructively and reversibly — installing the request notification inside the source markers to maintain their relationship should their source markers change, or delete the production should their source markers be removed. However, for the sake of quick experimentation and tuning if nothing else, sometimes it is more conceptually useful to appeal to an intermediate and less general notation — particularly in the case of rhythmic cell generators.

So I will define a smaller set of channel listeners, which I will call channel momenta. These classes are responsible for taking the contents of a channel and continuing it out a little further in time. They are useful for maintaining a metrical grid, repeating otherwise idiomatic phrases for the purposes of proto-rhythmic structures. In *how long...* they are ways of injecting regularity into the movements of an agent (*accumulation, page 349*) without either choice or rigid duplication. In *Loops Score* they are responsible for continuing a motif such that other processes can effect the repeated gesture. Their left-hand side is always bounded by a temporal horizon (the "long past") of fixed size or number of elements and by the last element in the channel (the "now"). This window contains the pattern that is to be matched by the channel momentum object.

Of course, there is one pattern-matching domain where mainstream computer science can give us a significant head-start — so called "regular expression" matching. Thus there is an intermediate subspecies of fusion filter acts on annotated strings using an extended regular expression library. If we can produce a channel *parser* that takes a windowed area of a channel and transduces the markers and the gaps between markers into strings of annotated characters, then we can formulate our production rules in terms of regular expressions. Most regular expressions (and all regular expression engines) match portions of strings of characters using expression encoded in other strings. In Diagram, the target characters are annotated with information connecting them to the markers, or the spaces that created them. These annotations do not affect the matching power of the regular expression, but they do follow their characters along for the ride. Produced markers can then be created in terms of the "captured groups" (strings of now annotated characters) of the template regular expressions by channel *formatters* which have an opposite role to channel parsers, turning these marker-annotated characters into new markers in the future of the channel.

In this work I was assisted greatly by the open source regular expression engine that ships with Java 1.5 — http://java.sun.com

Channel-level operations — opportunistic alignment

Most of the channel manipulations described above cut across multiple times on the same channel. There is an important class of channel manipulations that link markers across channels, typically markers that are temporally proximate. Of most use in *Loops Score* are the alignment manipulations. These take the markers in a source channel and try to make the markers in a target channel line up. At first blush, this is is not too dissimilar to the object-based alignment of popular graphical tools (such as Adobe Illustrator or Apple Keynote).

Adobe Systems Inc — http://www.adobe.com/illustrator

Apple Computer, Inc — http:// www.apple.com/keynote

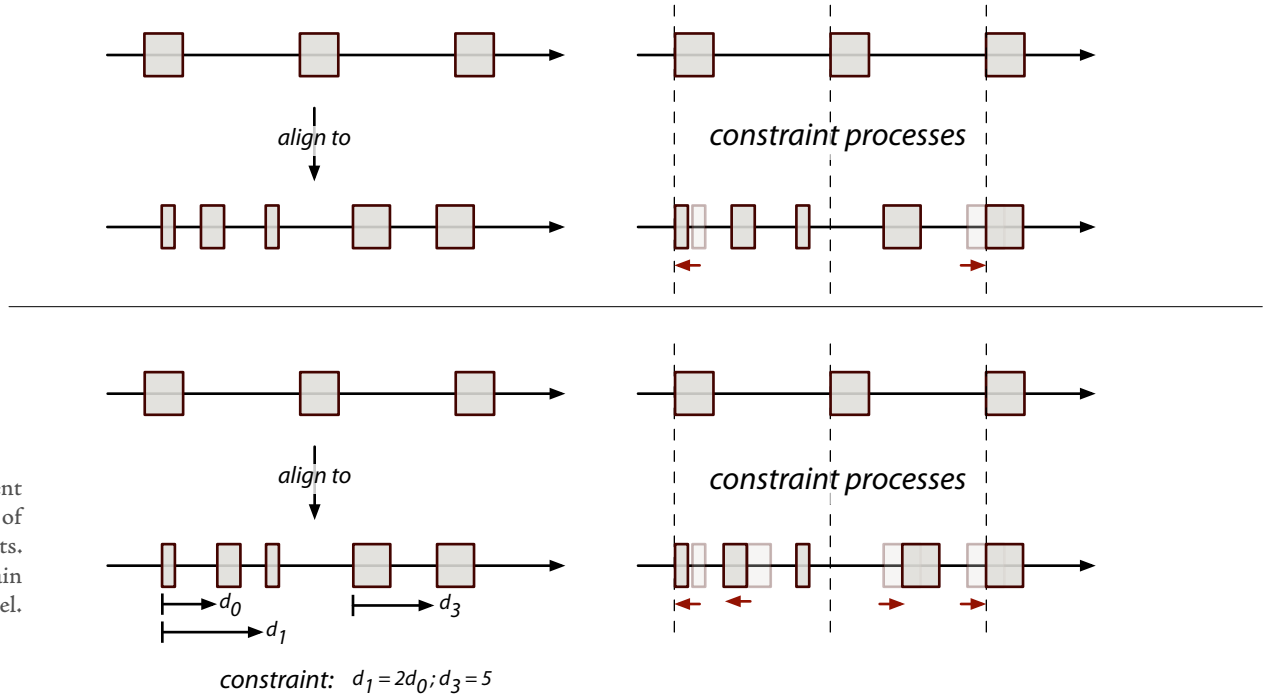


figure 98. Opportunistic alignment processes try to keep the contents of two channels aligned, within limits. They can work with any constrain processes active in the channel.

The fastest implementation is the most obvious: specifically, when a pair of source/target markers fall close together in time, we change the location of the target marker to align with the destination marker.

However, again, we return to the principles of the Diagram system. In particular, these alignments should be reversible and live. The key to maintaining these principles is to store the the marker positions in a framework that blends multiple, overlapping and persistent opinions about what a position should be. We have already seen one such framework — the generic radial-basis channel. Specifically, we create a class of marker that stores its position and duration in a generic radial-basis channel, *page 152*, that has a (position, duration) value repre-

sentation. Now we can re-implement the above example. Rather than setting the location of the target marker to be the position of the destination marker we create a posting that expresses and maintains the constraint that the target marker should be the same as the source marker, whatever that value should be. Now external processes are free to move the source marker around and our pinned target marker will follow — therefore, we should also express the conditions in which the constraint is violated and "removed" (contributing to the radial-basis channel with zero weight). This makes the alignment reversible (no information is deleted) and live (the operation is actively maintained).

Such alignments are extensively used throughout *Loops Score* to manage and reduce the otherwise chaotic rhythmic complexity that otherwise comes from a number of independent musical processes taking musical material and subjecting them to repeated transformation. Indeed the alignment of markers before transmission to the virtual piano is the central source of rhythmic arbitration at the output "stage" of the agent. Unlike standard rhythmic quantization there is no global grid imposed upon the output, but a local complexity limit for the temporal patterning.

Four extensions make this alignment more useful and more detailed for musical purposes. Firstly, when aligning with extremely sparse channels one might wish to use a fusion filter to generate virtual markers "between" sparse markers. In our graphical tool metaphor this is equivalent to aligning not just to the edges of the page, but to the center as well. Secondly, we might choose to find the temporal distortion for un-aligned markers in terms of the nearby aligned markers. For the purposes of efficiency *Loops Score* uses a simple linear blend for unaligned markers that fall between their nearest aligned markers; *Imagery for Jeux Deux* uses a a radial-basis function solution after a straight line fit to couple the note-level channel output of a score follower to a set of video keyframes. Thirdly, there is no reason not to make the constraint bi-directional; rather than

forcing the target marker to have the same onset as the source, we force both markers to the same, intermediate time. This is the technique used in the output stage of *Loops Score* (where, additionally, the relative weights on the movement come from the amplitude of the musical event represented by the markers).

Finally, we note that in the generic radial-basis formulation we can also encode additional constraints into the positions and durations of the channel — perhaps some markers must be in the middle of others, often the ordering of markers are significant and cannot change. This representation will see a much more use when we discuss the visual counterpart to the channel marker: the Fluid graphical system.

Concluding remarks

Loops Score is a densely overlapped work of computer music that, like *Loops*, moves between areas of shocking clarity — piano mimicking the sound of Cunningham's voice — and periods that are propelled and sustained by its own obscured but palpable logic. Like *The Music Creatures*, the piece with its general strategy of capture and repetition-with-modification, produces clearly perceptible intentional development, a deeply rhythmic movement with no stable pulse or tonal center. More than *The Music Creatures* it is an oddly self-balancing yet unbalanced music, culling the variety of the over-prepared prepared pianos and often offering slow and audible development of material.

Technically *Loops Score* set out to be an exploration of the recently created Diagram framework in a setting closer to traditional computer music than *The Music Creatures*. That it offered a thorough "work-out" of the technology at the lowest level is undeniable — the architecture survives the instantiation and destruction of tens of thousands of actions and perhaps millions of notes without intervention. The re-coupling of the strategies afforded by this work and a

more visual, perhaps a more "agent-like", set of concerns occurs in the new dance pieces, in particular the work for *how long...*

I believe that in this virtual choreographic domain the emphasis, borrowed from my musical concerns, on the complex patterning *of* time and the authorship of rich, open forms *in* time provides a fresh perspective on the creation of live time-based media. The Diagram framework successfully hybridizing the reactive, or "interactionist", tendencies of shared by both the agent and mapping perspectives with the more ponderous and typically off-line strategies of non-real time and perhaps even non-algorithmic music.

We should step back and return briefly to the axial decomposition of action-selection techniques discussed earlier, *page 89*, so see where the diagram framework fits in. Clearly, the Diagram framework core action-selection algorithm pushes the c43/c5m action strategy into allowing multiple simultaneous actions. However, as a supporting framework, Diagram also reduces the burden of responsibility on the core selection technique — no longer is it responsible for all of the high level temporal patterning of an agents actions. No longer is action selection the final structuring step (with the details to be filled in by a motor system) in the creating of temporal structures. Rather it is the first step, with the results of action selection to be further crafted by collaborating processes. By allowing post-hoc manipulations of future, scheduled actions and interactions, multiple processes that cut across the results of action selection can both clean up and constrain them. This acts to reduce the "temporal uncertainty" faced by the author of an agent, while at the same time offering hybrid strategies, orthogonal to action selection, that allow for more complex patterning of action.

In standing back and surveying *Loops Score*, the Diagram framework and the smaller "glue systems" that this chapter has developed, we see that this "complex

patterning of action" — what I might call the *choreography* of the digital agent's actions — is in fact the goal of this family of techniques.

The context-tree permits the kind of modularity that defuses one of the central methodological contradictions of making art with a complex process — how choices act upon a process that is simultaneously being developed, how the *names* of parameters, styles, behaviors, states, movements retain power over the complex processes without fusing solid the agent's inner workings (and with them our creative process) prematurely. Through decoupling elements, through code injection, and through carefully made persistence frameworks, we have complex processes that can support long-term collaborative practices. I am tempted to call this the complex patterning of *collaboration*.

Is this choreographic? The Diagram framework, in its explicit articulation of action selection and scheduling, makes a computational *representation*, in the sense of the introductory chapter — a site for further, agglomerative, transformation, for the collision of processes and constraints. It is these structures that allow for the small, nimble agents of my work in dance theater.

These "language interventions", the multiple uses of the context tree, the agents made up of changing parts, the explicit patterning of action in the diagram framework, all are motivated by the need to author and contain intricate processes. One cloud remains, however — how to take the techniques developed in this chapter cast them in such a way that they can truly be deployed "live", as it were, in a collaborative creative process. This thesis's answer to this will be given in the last chapter, on *Fluid*, the graphical environment that responds to this problem.

This chapter concerns two pieces for dance theater: 22, *with Bill T. Jones, and* how long does the subject linger on the edge of the volume...? *with Trisha Brown. In addition to the unprecedented use of real-time motion capture,* 22 *presents a novel class of non-photorealistic rendering techniques.* how long... *is a more sustained and lasting work, and most of this chapter is devoted to its overlapping and interacting agents.*

Chapter 7 — *22 & how long does the subject linger on the edge of the volume...*

Two works for dance theater conclude this thesis — *22*, with choreographer/performer Bill T. Jones; and *how long does the subject linger on the edge of the volume...?* with choreographer Trisha Brown.

The pieces were constructed during a series of intense, week-long residencies throughout the two years leading up to their premiere. Given the considerable expense involved in maintaining a dance company in a theater, and the speed with which choreographers are able to manipulate the movement of their dancers, there is a considerable responsibility for the digital visual artist to develop a working style and a working tool-set by which algorithmic "material" is prepared ahead of time and deployed, tuned and remade live in an improvisatorial setting. My aim upon entering these collaborations was to find a working area similar to that of *Loops* — where the initial algorithmic ideas, the tentative and tactical "formal systems", should be layered and surrounded in such a way that not only could they surprise us as visual artists with unexpected correspondences and developments, but that these surprises could be seized, assured and folded back into the work and its development. The aesthetics of effort, intention and transience first developed in my *Music Creatures* was to be our point of departure

for new agents that caught fragments of motion from the stage. And the techniques, and the very musicality, of *Loops Score* was to be transported into an animated realm.

1. *22*, an overview

22 is the live imagery created to accompany a new dance work of the same name by choreographer / performer Bill T. Jones. Like *how long...* which was developed in parallel, this work is for real-time motion capture in front of a live audience. Unlike *how long...* this piece is an improvisation for solo performer, and although it uses similar processes to understand the movement of the dancer on the stage, it ultimately represents a strictly simpler use of the agent metaphor. Therefore, the focus of the discussion here is primarily on the non-photorealistic rendering techniques developed for this work and, later, secondarily on the examples *22* provides for the Fluid graphical environment.

22 took, as its point of departure, Jones's 1980s work *21* which combined spoken narrative with a series of *21* poses drawn from magazines, film and everyday life. In 21, these initially cryptic poses become iconic when purposefully labeled in a fixed expository sequence early in the work. As Jones begins to speak, telling stories from his past leading up to the present moment, this fixed sequence of named poses exists as a parallel medium, sometimes hidden in more dance-like movement, sometimes surfacing as an almost clock-like circular motor, always threatening to intersect the meanings of the narrative or disrupt the re-telling of the story.

The story of the working-class mother feeding her daughter to her abusive husband also appears in Jones's memoirs — **B. T. Jones,** (with **P. Gillespie**), *Last Night On Earth*, Pantheon, 1997.

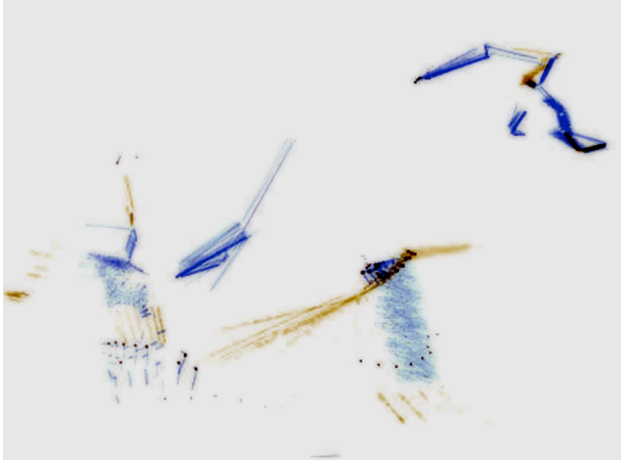


figure 99. Parts of the work *Lifelike* began to explore the passage between the abstract and abstraction — here is a horse figure, based on the decidedly "offline motion capture" of photography pioneer Eadweard Muybridge, see: **E. Muybridge,** *Animals in Motion,* Dover, 1957.

In *22* the imagery (and the music, by composer Roger Reynolds) enter the work as another parallel track that can coexist in the same space as the performer's movement and words and threaten the flow of the other media, only occasionally, but very clearly "interacting" to form new meanings with the other simultaneously presented elements. As in *21* we reduce the imagery to a self-prescribed palette and ordering of material — 7 "stations"; as in *21* these stations may be obliquely related to the narrative material — the retelling of a story told to Jones as a child (of a rather gruesome black working-class Thyestean feast) and a retelling of a story heard on the radio (concerning a photo-journalist's trip to Rwanda) — but create this ambiguous relationship mainly through their generic nature.

Unlike my other work related to choreography — *how long...*, *Loops*, or much of *Lifelike* — these stations are definite, particular and recognizable, and demand to be treated as such. Our stations are, in order: ladder, viewfinder, window, door, box, table. Conceptually this recognizable material is to be pushed around by the movement and the narrative on stage. Sometimes looping, appearing pinned in a machine-like cycle only to break free as the story progresses, and other times coexisting and arriving at a point of unison as if by chance.

The technical task, then, was to find a rendering method that allowed a fluid passage between the photo-real and the abstract and one where this passage would retain the sense of underlying movement from the material. To this end, I looked for a rendering technique that could incorporate the undeniable mimetic advantages of video with the control offered by synthetic, real-time computer graphical movement.

The re-projection renderers

The Music Creatures, *22*, *how long...*, *Max* and *Imagery for Jeux Deux* introduce a new class of real-time "non-photorealistic" graphics renderers — the "re-projection renderers". The core of a re-projection technique is the idea that the previous rendered frame is used to texture geometry that will be drawn in the current. Crucially, texture coordinates for these pieces of geometry are automatically generated to distort the previous rendered frame such that it maps onto the new positions of the geometry or newly extended geometry. Moving geometry thus carries the image of how it was previously rendered; its history is re-projected back onto its present shape.

A commonality throughout most of the works described in this thesis is the addition of various amounts of noise to the geometry drawn and the overlapping of transparent successive frames — a successive frame based "motion-blur" technique. In a re-projection context, this overlapping noise creates grain-like texture *on* the geometry that is controllable *by* the geometry because it stems *from* this very geometry.

This geometry is no longer computationally isolated from the rest of the scene, or the rest of the image that it is rendered to (as in traditional static or dynamic texture mapping) but actually has a relationship to nearby elements — the graphical canvas is no longer smooth and error free. Since pieces of geometry that overlap necessarily share the same texture — there only ever is one previously rendered frame — overdrawn areas of the screen end up interacting on the image plane outside and inside the drawn geometry as if, say, one line were carefully rendered taking into account all of the other lines in the scene (a computation that would probably be computationally prohibitive). The accumulated noise of a re-projection rendered scene may have areas where low frequencies dominate, but this textural *blurring* is unlike most of computer graphics.

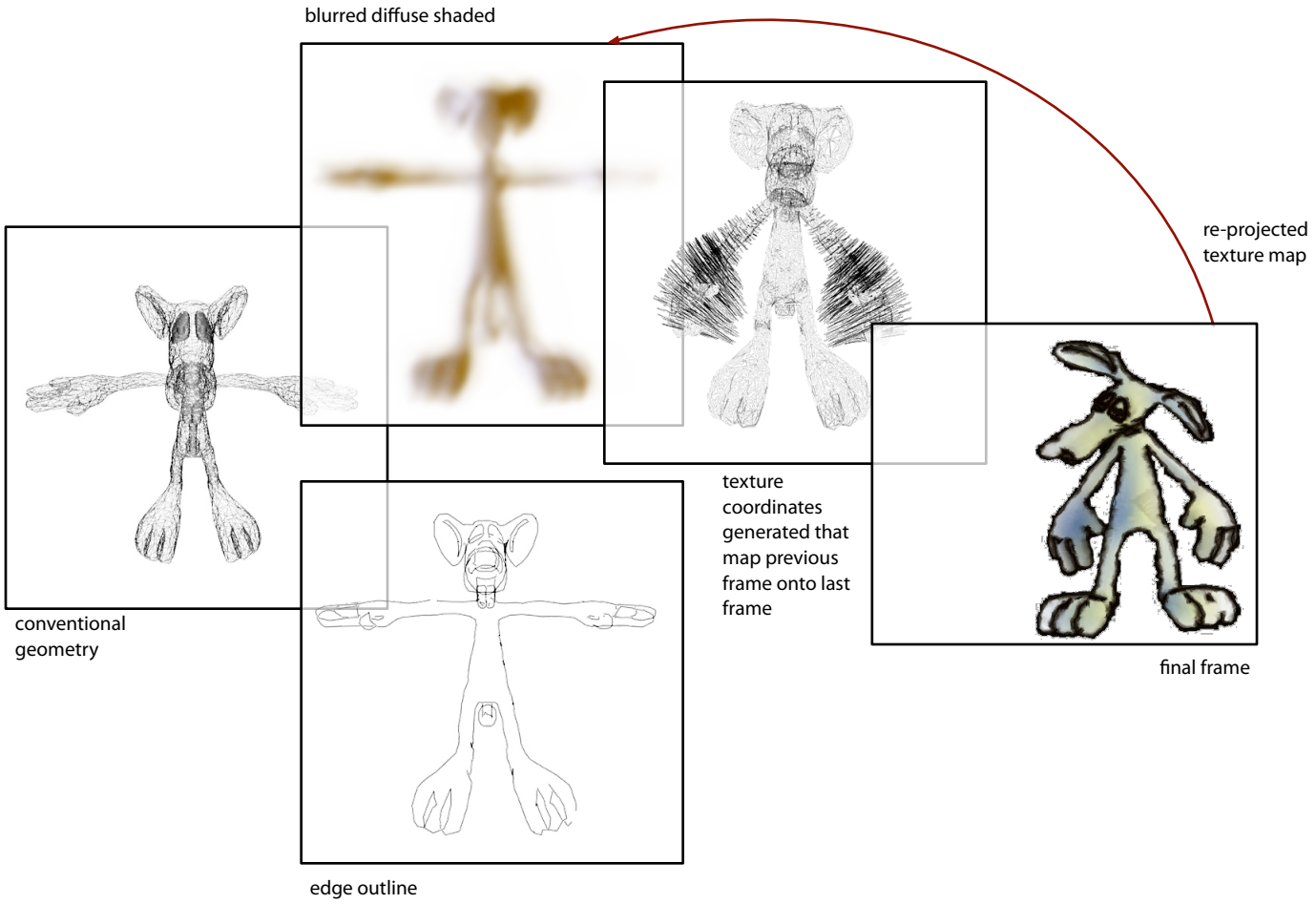


figure 100. This re-projection renderer character — *Max* — illustrates the basic re-projection shading and edge-tracing setup.

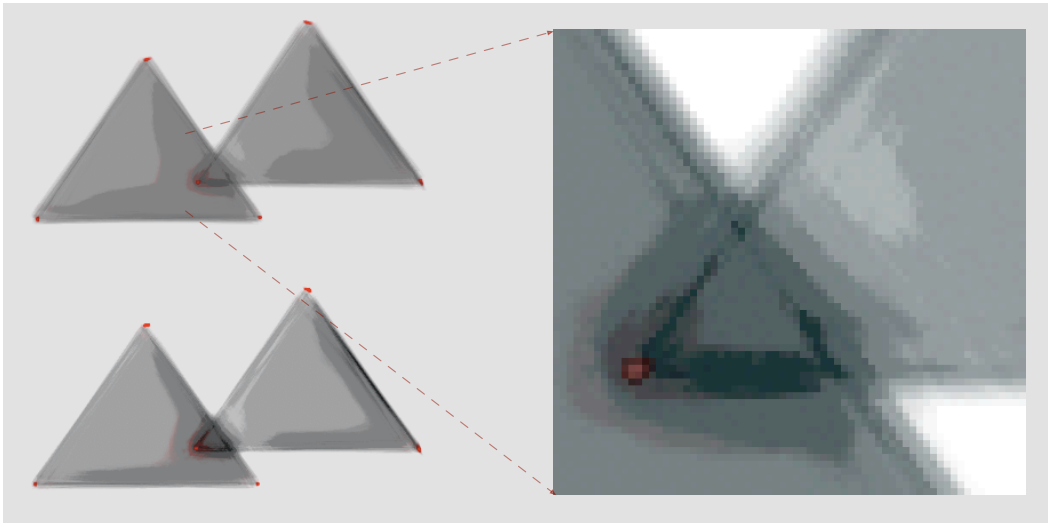


figure 101. A simple pair of triangles outlined in white and filled in using a re-projection shader. Note the interaction between overlapping pieces of geometry, and note the internal shading — coupled to the edge rendering and the overlap itself.

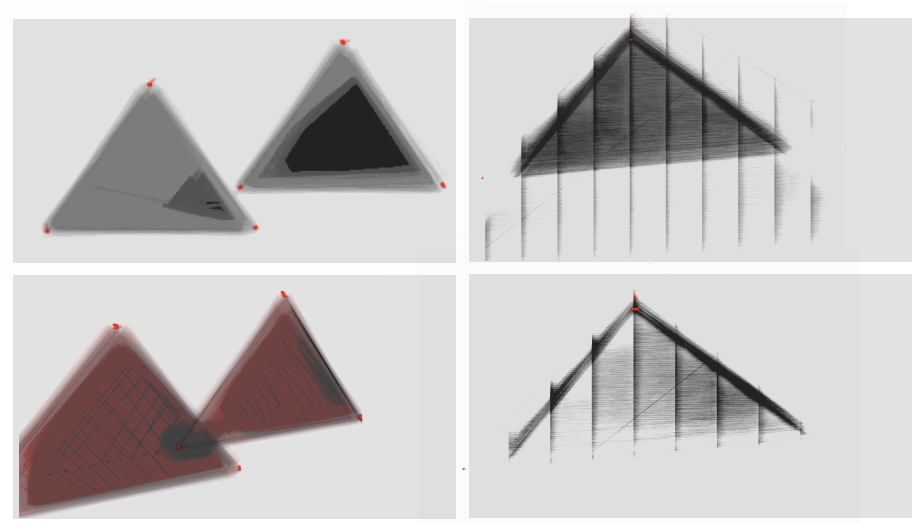
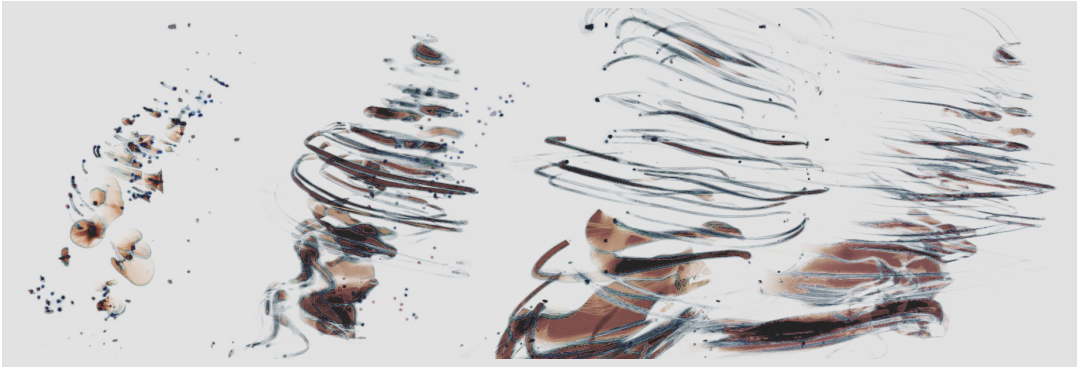


figure 102. Counter-clockwise from top left: 1. the same triangles as above, have separated — traces of their previous encounter remain; 2. one triangle moves rightwards over the other — trace of movement slowly fades; 3, 4 a moving vertical line has disappeared from view, and a trace of movement remains on the surface of the triangle.

figure 103. A more complex example (motion studies from video-tracked Jones motion). This geometry is rendered using a single linear primitive — all texture comes via re-projection.



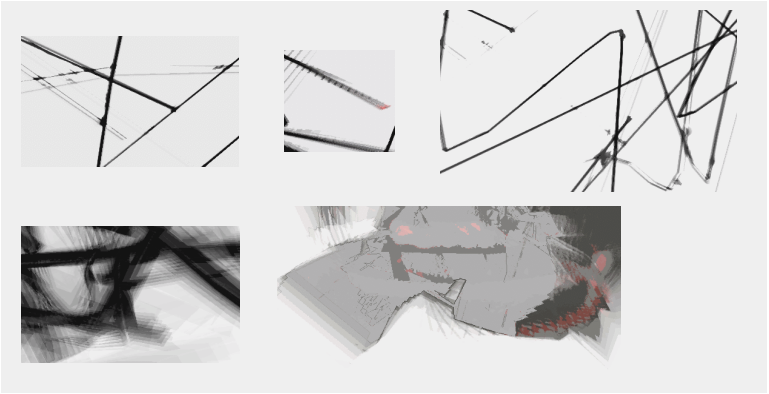


figure 104. Close-up of line fragments — the history of crossing lines interact with geometry on screen.

figure 105. Study for 22 — multiple frames from "table" sequence in a variety of video+ re-projective styles — note the use of distorted color space.

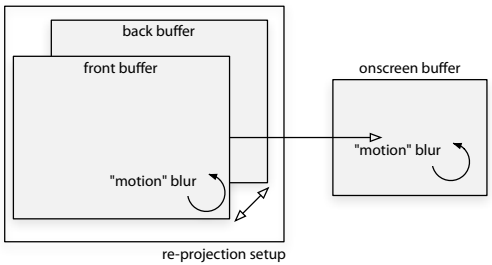


figure 106. The standard re-projection setup renderers offscreen at a high resolution (with a separate level of motion blur) . The back buffer is used to texture material in the front buffer.

Autodesk / Discreet —
http://www.discreet.com/3dsmax

Firstly, it is completely tied to the geometric content of the scene and secondly, it generated directly by the history of rendering and the accumulation of geometric material (and not by hiding the absence of process information in the picture).

Further, and this is especially evident in the "gestural" lines of *how long...*, geometry carries with it the history of its making. At the end of a "gesture" the "texture" that remains on the thickened line is inextricably linked to how that line was drawn over time. The image plane is not created afresh with every frame.

The final rendering technique for *22* augments a re-projection renderer to incorporate pre-made video material generated from pre-made animation sequences. These sequences were constructed in a conventional 3d modeling, key-framing, and rendering environment (3D studio max / character studio), and exported as both geometry and rendered video. By correcting for the offline-rendering system's lens parameters, this video could be synchronously projected back onto the moving geometry as its animation was played back in the real-time renderer. Thus, if undisturbed, this real-time renderer could appear to display all of the photorealistic techniques that the offline renderer of 3d studio max could provide — soft shadows from many light sources, complex materials, high resolution meshes — at the minimal expense of decompressing video.

Unlike in the case of video, however, we know much about the underlying motion present in what it is that is being projected onto the geometry. In *22* each station has the possibility of a human figure acting inside it — there is a man climbing a *ladder*; a child pushing a *box* etc. — since the video is projected onto geometry, we can re-synthesize camera movements with respect to specific body parts of the figure, or around important parts of this virtual set, or dynamically compose the stage picture of Jones and this virtual presence. While the geometry, in clothing itself in the texture, exploits video for its "fidelity" and "realism", video

here takes the geometry as a perfect annotation of the positions and processes that generate it. The technique is thus fully hybrid.

Of course, there is a catch: this "trick" of projecting video over synchronized geometry only "looks right" in a particular special case: if the virtual projector for the video has the same lens parameters *and* position as the virtual camera had in the offline renderer *and* the virtual camera of the real-time renderer is also at this position. While the scene can stay relatively "photo-real" with small violations of these conditions, for larger movements away from this ideal position, the coherence of the scene disintegrates.

However, what is less obvious is that the underlying movement of the geometry remains readable during this process of abstraction — the movement of the virtual figure in particular outlasts its figuration — and only in rare conditions are the mechanics of the actual technique itself legible. Because these shards of video intersecting with moving form are also visually interesting, the final full, *22* renderer incorporates this shading layer of video re-projection with another more typical re-projection of the previous rendered frame. The now standard noise and motion blur complete the set of graphical techniques used. This makes *22* the most complex "shader" produced for this thesis.

While the visual imagery for *22* exploits this shader to generate its graphical flirtation with figuration and abstraction, the full flexibility of this shader has not been seen in this work to date. In particular, little space to explore the *rhythmic* possibilities of desynchronizing the typically coupled movements of underlying geometry animation, underlying camera movement and projected video could be found during the dance work. Since less than one fifth of the prepared geometry/video library made it into this particular work, it is expected that these further uses of this re-projection shader will be explored in a separate installation.

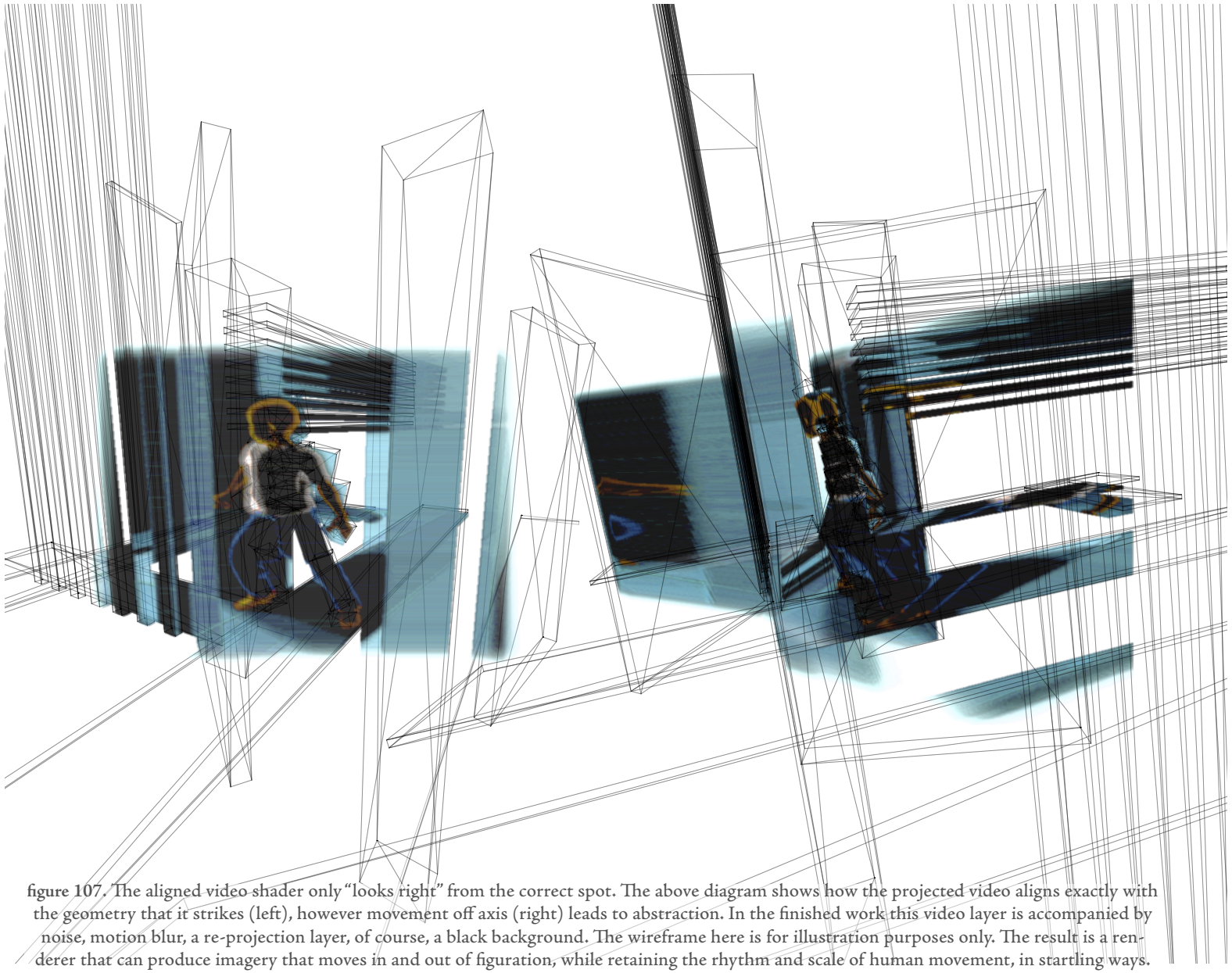


figure 107. The aligned video shader only "looks right" from the correct spot. The above diagram shows how the projected video aligns exactly with the geometry that it strikes (left), however movement off axis (right) leads to abstraction. In the finished work this video layer is accompanied by noise, motion blur, a re-projection layer, of course, a black background. The wireframe here is for illustration purposes only. The result is a renderer that can produce imagery that moves in and out of figuration, while retaining the rhythm and scale of human movement, in startling ways.

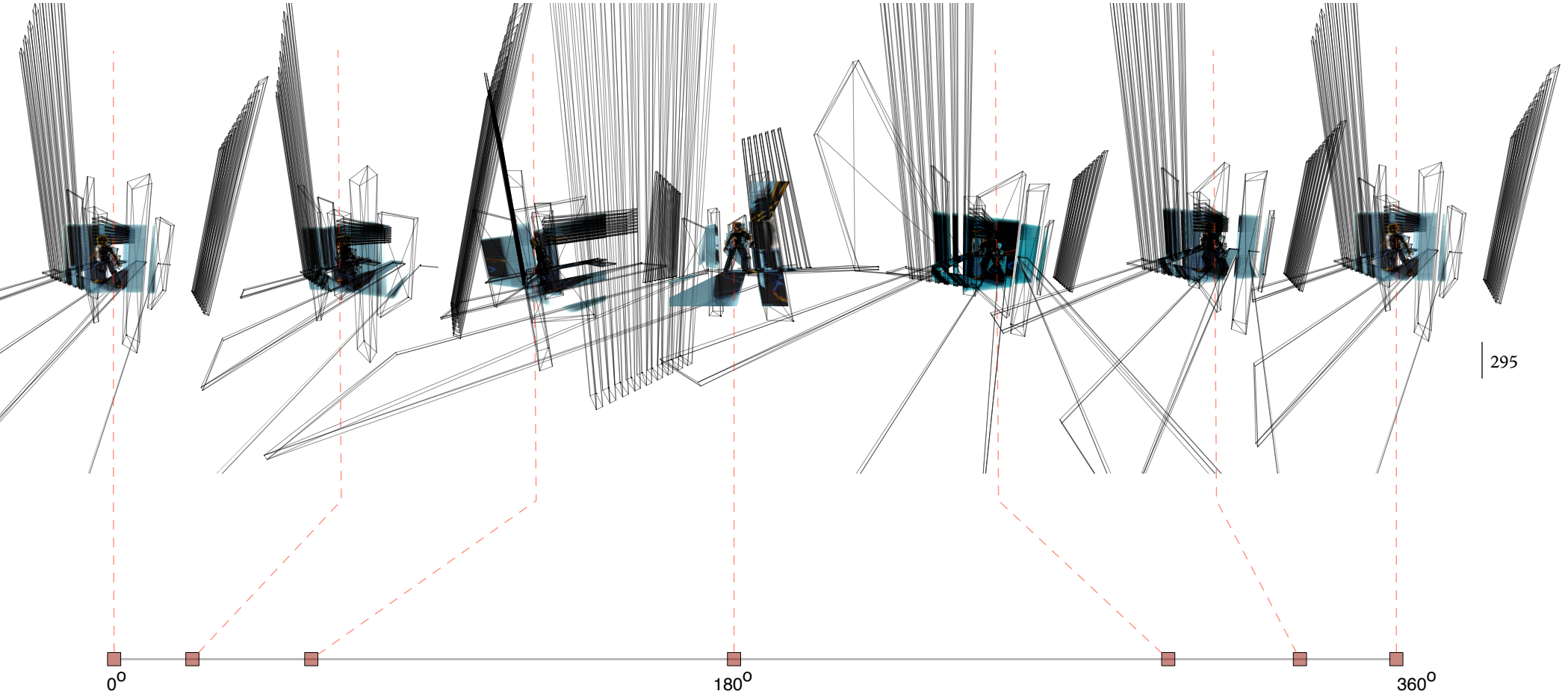


figure 108. The same scene as the previous diagram, slowly rotating around the child figure — which moves in and out of clarity.

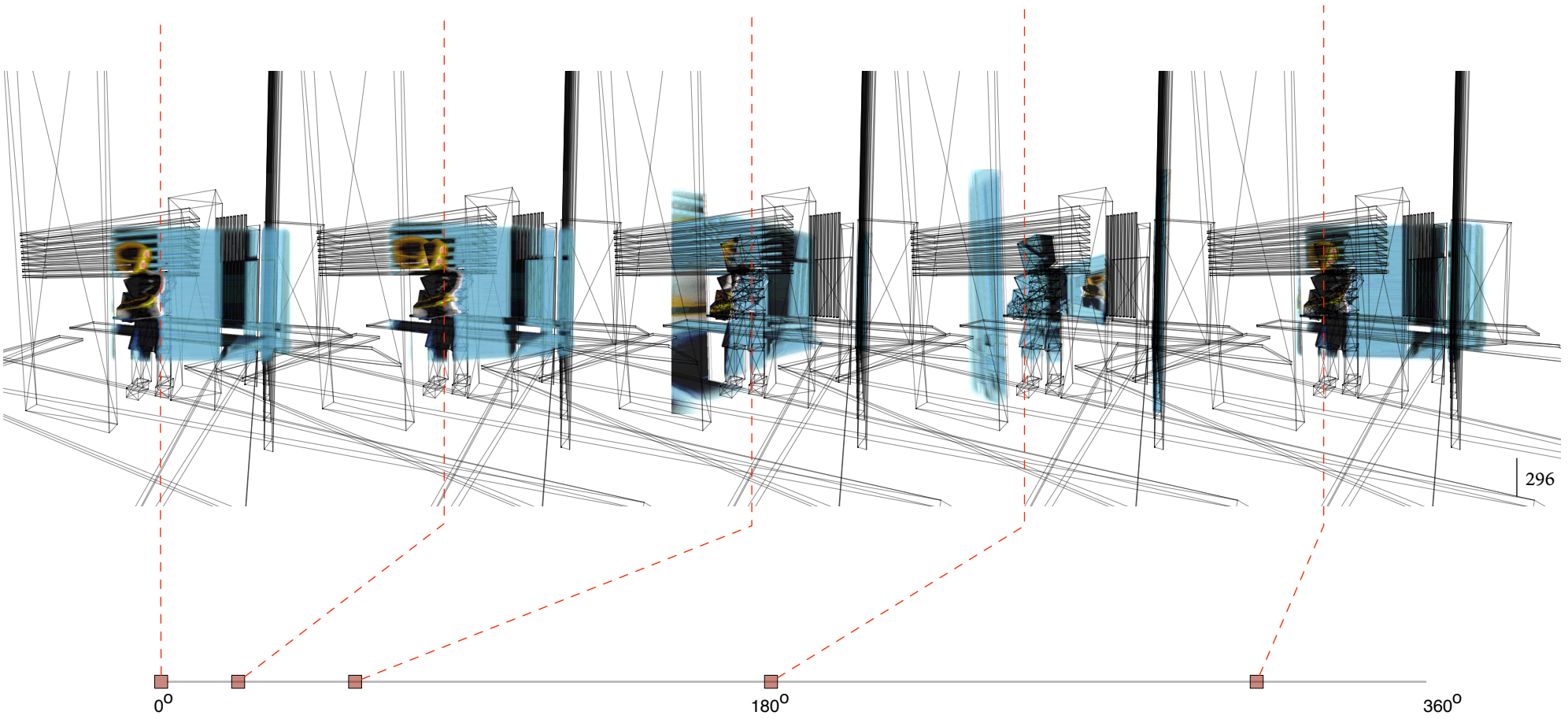


figure 109. Virtual camera movement is not the only degree of freedom that this shading technique offers. Here we rotate the virtual projector around the scene.

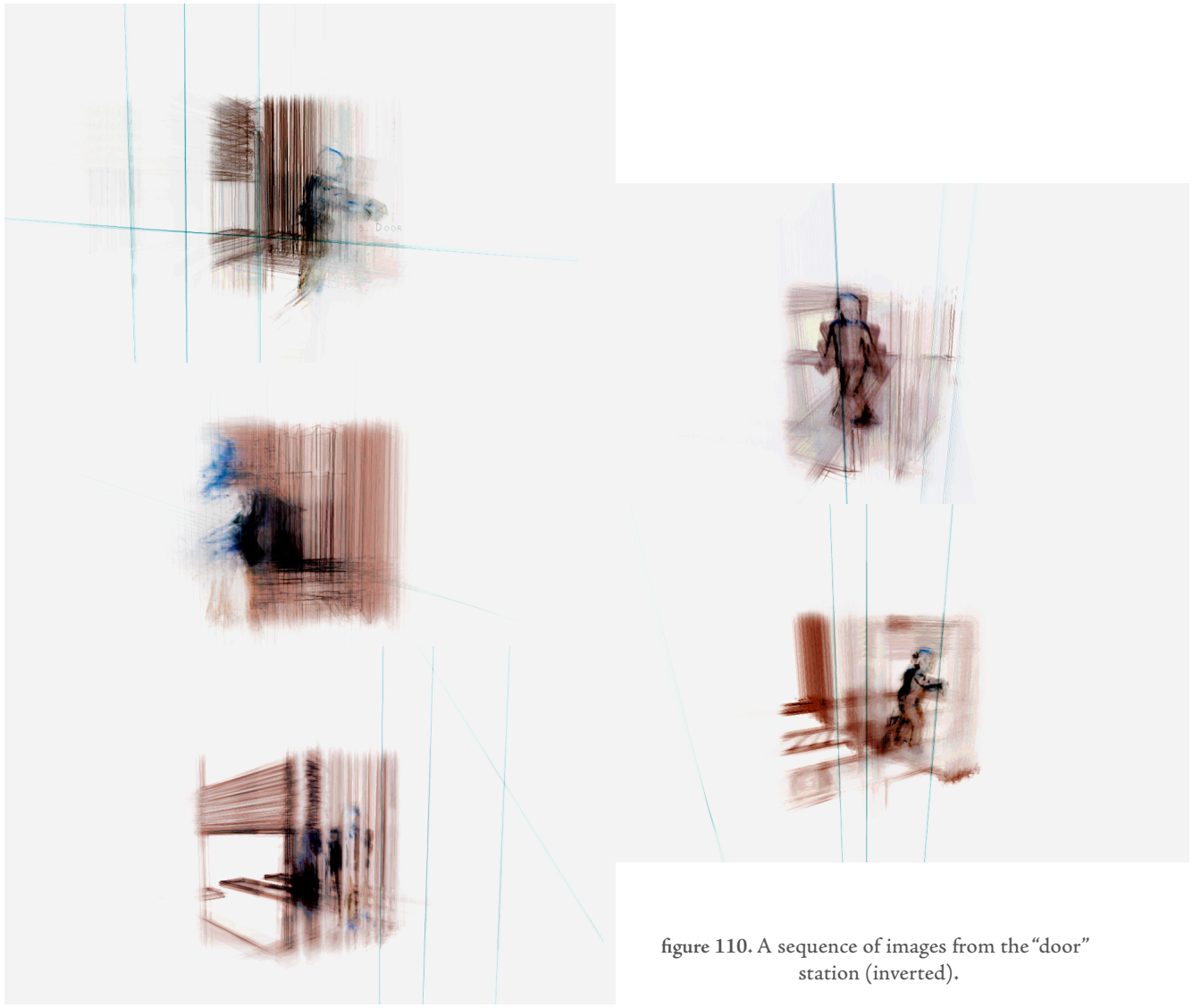


figure 110. A sequence of images from the "door" station (inverted).

figure 111. From "ladder" and "window" (inverted).

Distorted color spaces

One of the weaknesses of the re-projection rendering technique is its handling of color. Although it might be a highly controlled feedback technique it is still a feedback technique and as such, while we can control the feedback gain and the "neutral point" for each of the three color components — red, green and blue — the feedback structure has a tendency to diverge each component to either 0 or 1 resulting in an extreme quantization of color palette to only eight colors. Even when this feedback does not diverge, at the date of writing, general purpose frame-buffers on commodity hardware provide only 8 bits of precision for each color component. Normally 8-bits are sufficient for presentation purposes; however, it is insufficient for *computation* purposes — and as we move to drawing more transparent overlapping geometry with higher levels of motion blur we are more exposed to the accumulation of quantization errors in the frame-buffer. *22* is far away from the monochrome of much of my work (*Loops*, *Life-like*, *The Music Creatures* are all monochromatic work and *how long...* is a set of almost monochrome + red palettes).

Clearly the color interpretation of the frame-buffer must be remade while we patiently wait for commodity hardware to support more color depth. The full RGB component model is a good, generic color space, but perhaps there are alternative spaces that are better suited to rendering the results of the re-projection renderers. One conventional solution would be to interpose a 3x3 or a 4x4 matrix that could rotate, or rotate and translate, the RGB color-space into some other space. However, this solution is still limited to a linear or projective-linear transformation of color.

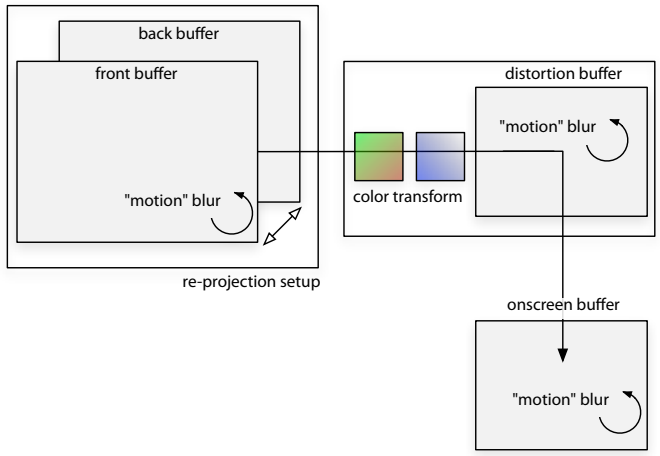


figure 112. The full re-projection setup adds a buffer used to perturb the drawing of the main buffer onto the screen and an additional two texture lookups.

In 22 we construct an alternative interpretation of all four rendered components R,G,B, and alpha before placing them into the final display buffer (for final accumulated motion blur and presentation). The most general technique would be to interpose a 3d texture lookup based on the (RGB) components before presentation. Unfortunately the size of this 3D-texture is prohibitively large, if one wants full resolution control over the texture. So there is a compromise solution, which suits the typically compressed color palettes of the work. Two stages — a 2D texture lookup based on the 'R' and 'G' components from a texture that we shall call the 'R/G texture', which is then multiplied by another 2d texture lookup based on the 'B' and 'alpha' components which we shall call the 'blue texture'. These 2D textures and the intervening multiply operation, are full floating-point range and precision — no quantization occurs here.

Clearly by reconfiguring these two textures, the fixed points of a divergent re-projection-feedback can be relocated anywhere in the color space; the edges of these fixed points can be softened by adding animated dithering noise to these texture buffers; and the journey to those fixed points can be reshaped in number of ways — providing we accept limited access to a portion of the complete RGB palette. The technique easily produces a continuous blend-space of monotones or duotones, for example. Further, otherwise hidden, almost transparent geometry can have unconventional effects on the material that it overlaps, effects far away from the generic rainbow of the RGB color model: "blue" geometry, can pick out the intensity of a piece of underlapping geometry, while, for example, transparent "green" geometry might alter the saturation of the material that it overlaps. In the final presentation buffer, subsequent overlapping renders are blended in conventional RGB space. In this way the problems of pushing computation into a display resolution buffer can be overcome, as properties of the color-space are set to be recomposed as needed by the artist during the work.

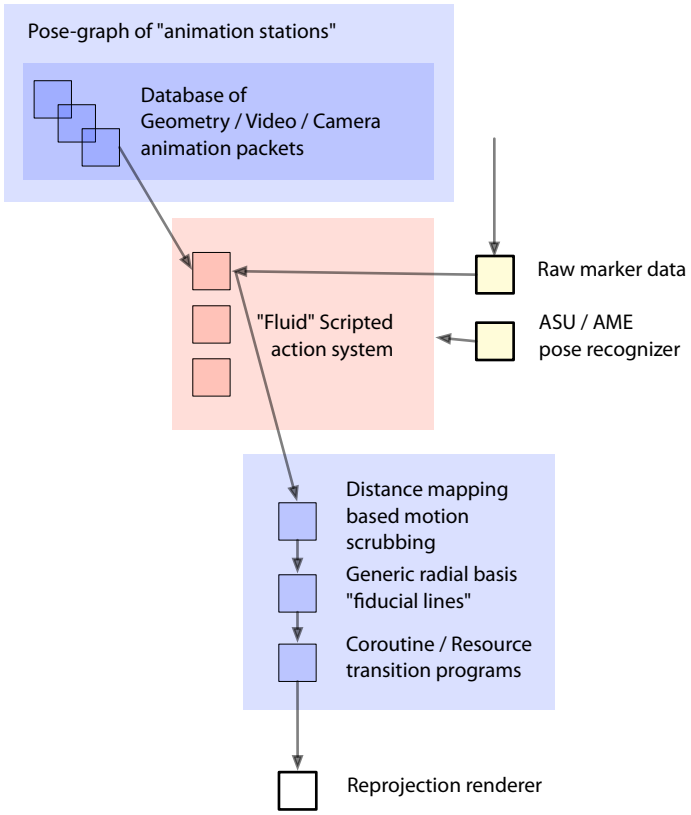


figure 113. The 22 agent system diagram.

The "motor system" of the agent creating this work live exists in a rather simple world and simply has to play out vertex, camera and video animations with various degrees of synchronization or deliberate de-synchronization. Its motor programs are dominated by the task of locking and holding rendering resources, *page 156*, that represent the constraints of real-time (video) rendering. For example, for extremely slow motion through a station it is preferable to blend successive frames of video, this requires the simultaneous use of two high resolution video textures. However, to cross-fade between multiple stations each station must make do with one one video texture. This degree-of-freedom locking life-cycle is expressible within the co-routine / continuation scripting model. The robustness of this framework here is critical: a single frame of misplaced video rendered across the wrong geometry is highly visible when we are playing this close to the photo-real.

Given the duration of the performance these stations are each performed twice. However, their moments of entry, the flow of time through them, their rendering parameters and camera movement, and their disappearance are coupled to Jones's movement and a set of stage-manager cues.

Jones's movement enters the work in two ways — firstly through a pose-recognizer, constructed by the AME motion analysis team at the Institute for Studies in the Arts, which recognizes a few of the 22 poses that Jones performs. These become cues that bend the flow of time through the imagery. The pieces of this malleable score, and techniques used to build it are the subject of the next chapter, *Fluid*. Secondly Jones enters the visuals through a distance-mapping-based analysis of his movement, irrespective of his pose. The mechanism used to manipulate the flow of the video/geometry playback is discussed in detail as part of the distance-mapping algorithm, *page 202*.

In addition to annotating and selecting the station material from the library of geometry and video to construct this sequence of "stations", a number of structurally important lines through the geometry were also hand labeled. These "infinite" lines, as they were rendered, are constantly on the transparent scrim in front of the audience and represented the fiducial possibilities of the space and the threat of the next station's coalescing. In the absence of information to track, these faint marks, sometimes propelled by motion from the stage, acquiesce and fade until they are barely perceptible. As a new station begins to form they prepare the way and guide the eye for the the underlying movement of the station. These transitions from being under the control of the geometry of a station, through traveling under their own momentum, to being quietly pushed around by the performer are negotiated through a simple generic radial-basis channel.

Concluding remarks

22, which like its predecessor *21* licenses the creation of dance works with multiple, simultaneous, streams of media that, unlike a classic Cunningham / Cage / Rauschenberg work, are deliberately crafted to create new figurative and narrative meaning. The imagery indicates, embodies and participates in the prevailing *threat* of unification in Jones's choreography and narrative: preempting, amplifying or illustrating the complex and unpredictable ways that Jones's stories, movement and vocabulary intersect. This challenge precipitated a different set of rendering techniques, and a different, less intricate control structure for them. As an investigation into the strategies required for creating autonomous systems that *coordinate* with movement, *22* holds a special place in the development of my thesis work. However, it is not until the next work that the promised dialogue between programmer and choreographer can be truly glimpsed.

2. *How long does the subject linger on the edge of the volume...*

The imagery for *how long does the subject linger on the edge of the volume...* was made over a period of a year and a half in collaboration with choreographer Trisha Brown for a new piece for dance theater for her company. It takes as its points of origin: the unprecedented technical ability to use a real-time motion-capture system to observe the dancers live; Brown's output as part of a "pre-history" of visual algorithmic art — her dance diagrams simultaneously visual programs for movement and the traces of movement; and my own impressions of observing, as an audience member, the unconquerable yet seductive complexity of Brown's choreography.

The imagery for this work is the action of digital agents creating their own "dance diagrams" live during the performance, displaying their own, inevitably incomplete attempts to slice and section Brown's choreography in the moment. The bodies of these agents are, as in *22*, projected over the dancers on a transparent scrim, allowing the images to share the same space as the performers. These agents offer their own choreographic hypothetical causes for the movement that they perceive and offer their own ways of notating the traces of the movement as it happens.

In many respects this work is the culmination of several of the threads of this thesis. As a work that is ultimately an overlapping parade of interactive agents, I was forced to reconsider the techniques I had been using to represent and manipulate the bodies of the agents — creating a generalized framework for doing so, the "blendable body" framework; the action-selection strategies and the motor-system organization exploit much of what was developed for our Diagram system; and the perceptual techniques used to build structures up from the live motion-capture material will, for the reader of this complete thesis, seem familiar — choreographic trackers based on the b-tracker framework,

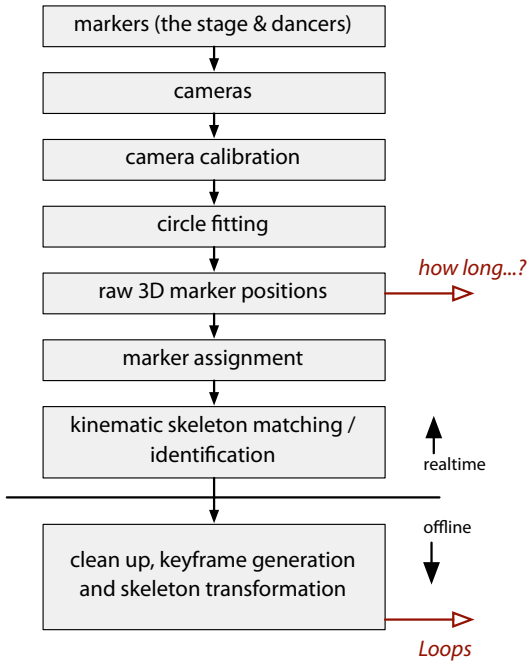


figure 114. Motion capture is a series of complex data-processing tasks. Real-time motion-capture systems must offer data at a variety of levels of processing if they are to be used for live digital artworks. *how long...* takes most of its data at a far lower level than other "real-time" applications. *Loops* required human cleanup of the data after capture, at a "rate" of around one hour of laborious clean-up per minute of motion capture.

distance mapping algorithms and persistent object structures. Finally, much of this work came after the creation of most of *Fluid* and of the glue systems, so it acts as a motivation and as a test of the principles embodied therein.

Raw motion-capture hardware provides points isolated in both space and time — these *untracked* points lack any skeleton or inter-frame correspondences. That our blendable body framework consists of a network of computational representations that start with these untracked points is no coincidence. Having this network act as a *body representation* allows for perceptual mechanisms to be reused as proprioceptual mechanisms and allows the agents to perceive other agents' bodies in the same manner as they perceive the live dancers. As we shall see below, it takes a some delicate work to turn these untracked points of motion into data suitable for coupled visual imagery.

Unlike *Loops*, *The Music Creatures* or even the other interactive work for dance that premiered on the same evening, *22, how long...* is a system that undergoes complicated and complete structural change during the 30-minute duration of the piece. The never-ending, never-repeating *Loops* or the carefully scored *22* represent one single system that is instantiated, and left to run in a world that, while it might push the system around, is always pushing the same set of parts. A few sections of the execution cycle might be turned on or off, but the actual palette never changes: in the case of *Loops* — 42 creatures, 30 actions, 20 rendering basis sets etc. — or in *22* — two channels of video, two channels of geometry, one or two agents using them.

Whole agents (and renderers and network resources) come and go during *how long...*, and perceive and stop perceiving each other and the dancers. This complex layering is harder to rehearse (it's not clear how one jumps into the middle of this world), more dangerous to program (the taking of problematic execution paths are deferred until late on in the piece and yet might be dependent on the

early parts of the work having taken place). It necessitates new tools — *Fluid* — and better programming methods — the "glue systems". And yet such a fluidity and a vocabulary of change is demanded by the heterogeneity of Brown's work and the exceptional intelligence by which changes of number, partnering and coherence take place on the stage. A *Loops*-like work with its constant, unchanging exploration of a change over a finite world would be out of place against any of Brown's recent choreographic output. The technical response to this challenge is the widespread use of the context-tree to ease construction and connection of objects and automatically handle their (sometimes partial) disassembly when they were no longer needed.

We will start with the lowest technical levels of *how long...* and work towards a narrative and technical description of each of the agents themselves, beginning with the treatment of the exciting, but troublesome, realities of the currently available real-time motion-capture data and progressing towards the general framework for constructing the bodies of this work's agents and then onto the agents themselves.

3. The problems of real-time motion-capture data

The leading industrial real-time motion-capture systems are designed to provide a very particular class of data — so called kinematic data. These data are the set of hierarchical joint angles and joint / bone positions that represent a human figure, and is the result of *raw*-marker data acting upon a pre-made kinematic simulation of a particular dancer. These data are typically clean, within plausible joint constraints, tracked and labeled with particular body parts. The goal of motion capture systems is to get to the underlying human figure, even in the presence of transient marker occlusions and measurement noise.

Unfortunately, during the complexities of modern dance, over the size of a proscenium stage and with a number of other, similar bodies also visible, these data are of surprisingly little use. In particular, since the typical motion-capture systems generally have found a role in military applications and medical biometrics, the systems are constructed to either provide very accurate tracking information or provide no information at all. A compromise is hard to find: due to the constraints of occlusion and the large capture volume, dancers wore a reduced marker set indicating their arms, head and back — making the kinematic fit harder. Both systems tested during the development of the work could take up to tens of seconds to register the entrance of a dancer into the motion-capture volume, and, when material became close and complex the kinematic data would simply disappear. There is no evidence to suggest that the kinematic modeling performed by these systems is anything less than state of the art, but, in the presence of uncertainty over an exact skeleton reconstruction and an exact dancer identification there should be something more useful than silence.

This piece, and this strategy, of course, owes its very existence to the generously provided hardware and engineering resources of the MotionAnalysis corporation who provided access to the raw, unlabeled marker data.

Instead of accepting this state of affairs we take the data at a lower level. We inject into the perceptual world of the agents for this piece, not the bones and labeled joints of the dancers but the raw marker data from the motion-capture hardware. This is before the kinematic fit is attempted, before the frame-to-frame tracking correspondences have been computed, just after the markers have been identified by the cameras and projected and intersected into three dimensions. It will be up to the agents that populate the world of the images to track and label these points, but as they do so, significantly, they can compute their own ideas of how reliable the tracking and labeling are. An entrance into the space becomes something that is immediately recognizable, exploiting the extremely low latency of motion capture cameras, while, of course, remaining a moment of imprecision and uncertainty as to the identity of the dancer or limb entering. That there would be some benefit to (re)building the data-processing path in the agent framework so that we can maintain the depth of the percep-

tual structures of the agents, rather than passively consuming the output of a proprietary black box, should come as no surprise.

One can extract a certain amount of information from unlabeled, untracked data, untreated. Indeed during the development of this piece, a motion-analysis team working in parallel created a number of metrics on the raw and labeled motion-capture data. The initial technical communication network of the work would have (unacceptably) put the visual imagery strictly "downstream" of this black-box analysis; the imagery was to be in a strict "visualization" relationship, or indeed a "mapping" relationship, with this "analysis".

Of course, this *ordered* "flow" of information is in reality an *order to* lose information, an order that the imagery was to be given no scope to negotiate. Such a position is fundamentally rejected by this work.

Unlabeled data

Let us first see how far one can get with untracked motion capture material, before augmenting it with our own tracking analysis. For example, there exist distance metrics that do not require inter-frame point correspondences — for example the (directed) Hausdorff metric from points $\{p_i\}$ to points $\{q_j\}$:

$$d_{h,p\leftarrow q} = \max_i \left(\min_j |p_i - q_j| \right)$$

This metric is useful for rapidly matching a subset of template points to a group of points, and it forms the basis for one of our "choreographic trackers" below. It is completely insensitive to the labeling of sets $\{p_i\}$ and $\{q_j\}$ (one can scramble the ordering of both these sets and their distance doesn't change). However, it is extremely sensitive to one of the kinds of corruptions that raw motion-capture

data face — ghost markers that flicker in and out, often a long way from the true position, due to errors in the reconstruction. A brief presence of such a distance marker can dominate the outer "max" operation above.

To construct more local (to a dancer or tight cluster of dancers) information one can always run a fuzzy k-means algorithm to partition the marker set into clusters. But this approach too suffers from a lack of robustness with respect to the flickering markers, albeit to a lesser extent — in particular in the case where the model-selection strategy dithers between different values of k.

J. C. Bezdek, Fuzzy Mathematics in Pattern Classification. Ithaca, NY: Cornell University; 1973.

Fulfilling my theoretical predictions of trouble, even after processing the various quantities (speeds, correlations) derived from this untracked information with filters with long time-constants or long median-filters, these numerical measurements of the stage were clearly inadequate — those that were smooth enough to appear reliable lagged so far behind the motion of a dancer as to destroy any possibility of a legible relationship forming; those that were fast enough to react in time were not stable enough to reliably use. This no-man's land of filtration seems to me a classic symptom of having incorporated insufficient information into one's signal set before treatment. Real-time motion capture systems can provide data with latencies on the order of 15 milliseconds — we should fight to preserve these startlingly correlated data.

Even had these low-level features turned out to relate to my perception of the movement of the dancers, *tracked* points and clusters of points are essential for any visual imagery that is tightly coupled to the dancers on the stage. To draw a line from a point on a dancer's body to another point in space that lasts longer than a single frame requires that we know the subsequent location of that particular point, unless the decision to draw a line is remarkably repeatable and based only on the position of the marker set. Therefore, constructing a point-level tracker, and arguably a dancer-level tracker is simply unavoidable.

Our solution consists of two parts: firstly we construct a point tracker. using the *b-tracker framework,* that can survive momentary presences of ghost markers and momentary absences of occluded markers. In doing so it can keep track simultaneously of the point locations and of how confident the agent should be in those locations. This confidence will then be used to assist and correct distance metrics and decompositions of the stage. These markers and clusters of markers enter into an hierarchical instantiation of the object-persistence framework developed for our agents and are now stable enough to be used as the impetus for visual imagery. Secondly, what subsequent layers actually do with these perceived markers dynamically controls the perception system's willingness to exchange stability (markers flickering) for accuracy (markers tracked with low latency). Indeed, having obtained a marker, or a "dancer", as a point of reference on the stage, subsequent systems are guaranteed that this position augmented with various quantities will be accessible for all future times. They are, of course, not guaranteed that this reference will be perfectly related to the present motion, but the existence of the point of reference itself is maintained and the computation of its associated qualities, including relevance, will take place as long as and action or motion command for an agent is interested. We might call this a "top-down" influence on our perception system and it seems to me that these techniques will be vital for the creation of visual imagery coupled to motion-capture data for some years to come.

The core of the point tracker is an implementation of a well-known linear-programming-based assignment solver known as the Hungarian algorithm. It solves the problem of assigning a set of n-objects to a set of m-objects in such a way as to minimize the sum of distances between the n-object pairs. Each object is paired with a unique object. This is the basis "incoming → ongoing match" part of the b-tracker assemblage. The input to this algorithm is simply the n by m distance matrix M with elements M_{ij} representing the distance from object i to object j. The strict metricality of the distance metric used to generate the M_{ij} is irrelevant; as long as M_{ij} are non-degenerate the Hungarian algorithm will find a solution.

The Hungarian algorithm is introduced in: **H. W. Kuhn**, *The Hungarian Method for the Assignment Problem*, Naval Research Logistics Quarterly, Vol. 2, 1955, pp. 83-97. It solves the assignment problem posed here in $O(N^3)$ time. For a more contemporary use in the field of shape recognition — **S. Belongie, J. Malik, J. Puzicha**, *Shape Matching and Object Recognition Using Shape Contexts*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (4), 2002.

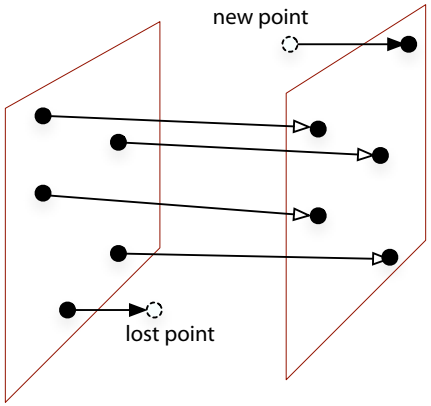


figure 115. The point-matching algorithm tries to assign this frame's set of points to the previous, predicted, ongoing model positions. Occasionally points are dropped and new points enter the scene.

We take $n \leq m$ for this discussion, without loss of generality, for we can reverse the sense of the set designations. Incoming points to be matched with the current population of points are in the I_i , $(i \leq n)$ and the current population of points in the set O_j , $(j \leq m)$. The Hungarian algorithm is executed for each new batch of incoming points to generate new assignments to the existing population. These new points are then "merged onto" the points that they were assigned to. This approach of merging new information with a dynamic population of old "tracked points" is of course similar to the core of many of the perception systems described in this thesis.

Obviously, the number of points that deserve to be in the current population changes constantly during the piece (with the presence and absence of dancers, or other agents). Further, the solver is constrained to map each object $i \leq n$ to a particular object $j \leq m$ no matter how far away this assignment ends up being. To allow a dynamic number of points in our current population, and to prevent the solver being forced into making particularly poor decisions for the sake of

making complete assignments, we should pad both of the object sets with virtual points that will act as sources and sinks for the assignment. If an incoming point is matched to a virtual point in the current set, it is because it has "just arrived", and if a current point is matched to a virtual point in the incoming set it is because it has been (perhaps temporarily) "lost". Matching a virtual point to a virtual point is an irrelevant and frequent event. As a first pass, we set the distance from marker to virtual point to be some value greater than the distance that a point could travel in a single frame. Virtual point to virtual point distances are set equal to half this value.

The input then to this algorithm is a set of distances, so we need a distance metric from an unlabeled marker to an ongoing point model. The **ongoing point model** is a simple 2nd order (modeling 3D-position, velocity, acceleration) Kalman filter, which treats being assigned to an incoming marker as a measurement event and predicts the place of the marker on subsequent frames. This predicted position is used as the basis for the distance between an incoming marker and an ongoing point model.

The **confidence model** consists of two components: the match history for ongoing points models (whether or not the point matched a real point or a sink), and the noise estimation on the position of the point from the Kalman filter.

The match history of a point looks like an low-pass filter on the pulse train of 0s (lost) and 1s (matched).

$$c_m \leftarrow \alpha_m c_m + (1 - \alpha_m) \cdot \begin{cases} 1, & \text{point matched} \\ 0, & \text{point unmatched} \end{cases}$$

Thus the complete confidence for a marker is given by:

$$c = c_m / p_\sigma$$

where $p\sigma$ is the noise covariance on position from the point's Kalman filter. Confidences are always normalized before use.

In the absence of top-down pressure to maintain a model, point models are culled when their confidence score reaches epsilon (10^{-5}).

Now that we have a dynamic set of ongoing points each with a confidence and associated with a position, velocity and acceleration from the tracker, we can revisit our low-level analyses of point motion. Speeds become computed not from the Hausdorff metric but from confidence-weighted sums of the absolute values of velocities from the ongoing point models. Since brief ghost points never achieve high confidence levels and short "marker-outages" have little impact on either the confidence of the points or their velocity models, these metrics are highly impervious to the kinds of noise we have been seeing. At the same time, since no filtering additional to the Kalman filtering of the points is taking place, the processing latency is that of the Kalman filters. Even the bulk confidence-weighted acceleration values appear relatively smooth during periods of high total confidence and, to the naked eye, related in the to the underlying movement. Critically, at periods of low total confidence, the agents might prefer not to make any "big moves".

The units of this confidence are highly *ad hoc* and we are free, by changing the time constants on the filter score or by changing the noise priors on our Kalman filter, to change the distributions of scores over the (0,1) interval. There is a limit to this flexibility; should the confidences themselves show appreciable noise then this noise is simply re-injected back into the sum. But in general this allows us to trade latency for smoothness in detail, prior to the bulk-summation over point-level analyses.

What about the backwards, "top-down" influence on this tracker? Should an ongoing point model continue to be matched against incoming data all is well. However, should this model be culled (due to a persistent lack of confidence in its existence) any reference to it becomes stale. Once the model has left the tracker it will never be updated again. The solution to this problem is not to simply forcibly re-inject markers of interest back into the tracker for, after all, the model has been dropped for a reason. Rather, we transition from updating the point model based on incoming marker data to updating the model based on the local velocity field of the current point set.

The **local velocity field** $v(p)$ at a point p looks like the following, given a set of velocities v_i at markers p_i:

$$v(p) = \frac{\sum_i \left[e^{-(p-p_i)/r^2} v_i \right]}{\sum_i e^{-(p-p_i)/r^2}}$$

Finally, it is at the very least more visually appealing to have a smooth transition from these two updating domains. We therefore add to our model-update equations a confidence-weighted influence (with confidences c_i) of the local velocity field:

$$v(p) = \frac{\sum_i \left[e^{-(p-p_i)/r^2} c_i v_i \right]}{\sum_i e^{-(p-p_i)/r^2} c_i}$$

While points of interest that are culled from the tracker no longer accept marker data as measurements, their positions and velocities are still updated according to the above equations, and these points that no longer reflect an exact marker position are swept along by the motion of nearby points (and have velocities similar to nearby points).

The dancer-level tracker

We use a very similar structure to construct more dancer-level (rather than marker-level) perceptions of the stage. Rather than constructing a trellis of "ongoing marker models" we construct a trellis of "ongoing dancer trackers". Here, each dancer-tracker is implemented as a k-means tracker (where, for the purposes of this piece $k = 1 \ldots 4$). The k-means clustering algorithm is a simple and popular unsupervised clustering algorithm that iteratively converges to a Voronoi-partitioning of space into k areas (clusters), each cluster being represented by a center, also a position in space. While this commonly used algorithm is simple to implement, and requires $O(nk)$ time per iteration, which is acceptable for our $n \sim 50$ domain, it suffers from a number of problems. Firstly, the algorithm only converges to local minima and while these are often quite good they may be arbitrarily bad, and once a tracker has found a poor solution subsequent iterations on related data will probably also be poor. The literature recommends restarting from fresh, variously heuristically described, center sets, perhaps multiple times on each data-set and choosing the "best" decomposition. Secondly, the naïve k -means implementation requires a fixed k . If we were to run many iterations of many freshly started trackers, each with $k = 1 \ldots 4$, on each time-slice of data then this clustering would begin to be rather computationally intensive.

Our less naïve implementation maintains a smaller population of (good) k-means trackers. At each time-step each tracker i "predicts" a future configuration of k_i - centers and, optionally, predicts a k_{i-1} or a k_{i+1} tracker formed by merging or splitting centers. By moving up and down in 'k' through splitting and merging we hope to leverage existing successful decompositions of the markers rather than randomly restarting them.

In order to fit into our perceptual framework, trackers need to be scored. We score in two parts:

The first part is a **running score**, and reflects how long this tracker has been part of the population. Two running scores are kept, with different initial conditions: for the purposes of "culling" this score is initially set to 1, for the purposes of confidence this score is initially set to 0.

The second part of this score is a measure of how well this tracker is clustering the data. Here we use the **Bayesian information criterion** (BIC) which has the advantage of penalizing the flexibility of high k models, the details of which are given earlier, *page 115* k -means trackers produce $k+1$ -means trackers by fissioning their largest cluster if this results in an increase of the BIC score. Similarly $k-1$ -means trackers are produced by fusing the smallest cluster with its nearest neighbor should this seem better from a BIC standpoint.

This perceptual structure proved more than adequate for the task of segmenting the stage into dancer-like clusters of markers during *how long...* Indeed, this framework offers flexibility that seems rather under-taxed during this piece. Less than a year before the premiere, it had been thought that a larger number of dancers (increasing the k- search space), each wearing a larger number of markers (increasing the computational cost of running a tracker) would be available. Given the constraints of camera placement and lighting in a proscenium the resulting marker and dancer counts were reduced.

"Tracking" higher level features

Given the Kalman-filtered tracked points, and their ongoing confidences, and the hypothesized dancer-centers, we are in a position to create slightly higher

level descriptions of what is occurring on the stage. And here our hypothetical mapping-based approach and this agent-based approach continue to diverge.

There are two kinds of ways of looking at the stage, in this work. Firstly, we can produce speeds, heights, and directionalities integrated and averaged over all of the tracked hypotheses; we might even weight these averages by the confidences of the models averaged, as above.

An advanced mapping strategy would typically take these measurements and begin to filter or manipulate them into something that would couple to something visually. For continuous processes this would literally be a filter network. For discrete events, triggers, thresholds and perhaps hysteresis mechanisms would be specified on these signals.

This translation from continuous to discrete is particularly poorly understood within a mapping realm. But an agent does not have to flatten, or integrate over, this information in order to make its decisions. Indeed it can defer loss of information until at least the action-selection stage and quite possibly beyond. This provides a second, alternative mode of reaction to the stage and interaction with the choreography: maintain these multiple hypotheses, these tracked models, and construct classifiers or recognizers over them. When it comes time to couple these hypotheses to discrete or continuous structures, do so through a competing, multiple action-selection technique.

The specific qualities of the hypotheses and perceptual sources will then influence, or parameterize, the specifics of the actions taken, so such "mapping"-like filter networks will have their place, but the use of these measurements adheres to a few general principles that make the working practice around these filter networks tolerable, or even tenable within a rehearsal or improvasatorial setting.

Firstly, we reuse the scaling and mapping techniques discussed in the development of *The Music Creatures* to provide a coarse level, purely data-driven treatment of these numbers without recourse to direct hand tuning. For example, any model of "fast" that is created in this piece is created on these rescaled data.

Secondly, we reuse the *beam-search mechanics* used for the b-tracker in order to form time-sequence *parsers*. These recognizers go beyond simple instantaneous thresholds of the data and again allow indirect and explicit specification of phenomena to match. For as soon as we begin to consider allowing the data to *trigger* an event visually — say, something to occur when dancers fall to the ground — we ought to approach this problem as a gesture-recognition task, no matter how simple, rather than a threshold- and hysteresis-tuning task.

Within this framework, constructing small "gesture recognizers" for the stage is simple:

a simple "recognizer" that recognizes when all of the dancers are performing floorwork, built using the beam-search matcher framework:

```
sequenceBSM = GaussianSequenceBSM();

sequenceBSM.new StateRange("high", 1, 5, 7.5f, 10);
sequenceBSM.new StateRange("low", 0, 0.5f, 0.5f, 3);

sequenceBSM.setSource(perDancer.getHeightChannel());
```

this creates a beam-search matcher that is looking to be able to parse sequences into two chunks, a state "high" for around 5-10 seconds followed by briefer state "low" for 0.5 to up to 3 seconds. Because of the automatically rescaled and remapped data provided by the height channel, we can specify high and low as simply 1 and 0. Because of the rich data provided by the height channel — both heights and confidences associated with all of the dancer hypotheses — we know that what the

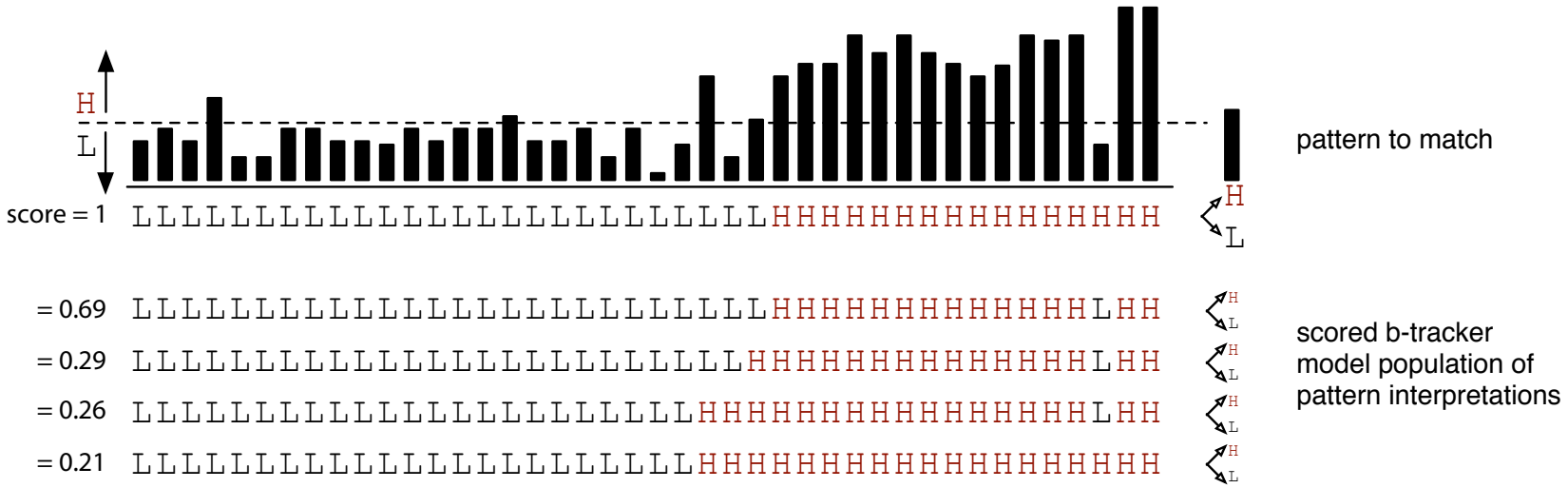


figure 116. A b-tracker population of models trying to parse a sequence of average heights into low (L) and high (H).

matcher is looking for hasn't been accidently filtered out from its data source.

Again, we use beam-search rather than a dynamic programming-based or Viterbi-parse-inspired approach — not for reasons of computational efficiency but to provide a little additional flexibility at a computational cost that we can easily afford for such simple (that is, low dimensional, low number of states, with simple state models) recognition tasks. The flexibility includes a more general choice of transition model, and access to the multiple-hypothesis structure itself.

This completes the overview of the techniques and principles used for the perceptual structures of the agents for *how long*... Before discussing the specifics of the agents themselves there is one more general framework that needs introduction. As this thesis has been presented there has been a general shift from pre-made agent bodies that are constructed (or loaded) once then manipulated — the wolf pups in *alphaWolf* and *Dobie*, for example, were modeled once and

then continually animated — to agents whose bodies are synthesized on the fly — the music creatures' *network* or *tile* for example. Each of these creature's bodies were created in a completely *ad hoc* fashion. For the purposes of *how long...* — a piece that was to be about the act of observing choreography with a whole range of agents — it was clear that a more general framework for creating such synthesizable bodies was needed.

3. Blendable body framework structures — point ⇄ line ⇄ plane ⇄ point

The urge to generalize before re-specializing is present throughout this thesis, and the same path has been taken with the body representations of the agents developed — the way that they are rendered on the screen, the way that they are controlled, and the way that they are internally represented.

Over the last decade the convergence of the asset pipelines of cinema special effects and computer games has resulted in a condensation of a dominant, core way of representing articulated figures in the broadening field of computer graphics. This paradigm can be briefly summarized as one that represents a figure's skeleton as a strict hierarchy (rooted, typically, somewhere in the base of the spine) of parent→child transforms covered with a deformable, "skinned" mesh. This viewpoint is re-expressed in the tools for modeling 3d characters , animating them, and rendering them, in the commonly available game engines, in the recent computer-game textbooks and in the commodity consumer-graphics accelerators. It is the responsibility of artists not to go so unquestioningly down a road that has been so neatly laid out for them. Perhaps there are "misuses" of 3D-modeling software, of key-frame animation packages, misreadings of the typical real-time rendering engine and more exploitive ways of exploiting commodity hardware, that lie off this rather well traversed path.

This standard, conventional body representation (in particular, a hierarchical structure of general transforms) is prevalent throughout computer graphics, video games and digital art. Papers where these issues are considered rather than simply assumed include:

W. Shao, and **V. Ng-how-Hing**, *A general joint component framework for realistic articulation in human characters*. In ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics, pp. 11-18.

N. Badler, C. Phillips, and **B. Webber**, *Simulating Humans: Computer Graphics, Animation, and Control*. Oxford University Press, 1993.

In computer games / pedagogy: **D.H. Eberly**, *3D Game Engine Design : A Practical Approach to Real-Time Computer Graphics*, Morgan Kaufman, 2000.

In digital art, artists often implicitly accept the body model assumed by the tools that they used. A high-water-mark in the aesthetics of the hierarchical body model is the installation of Paul Kaiser and Shelley Eshkar, *Ghostcatching*, 1997.

The re-projection renderer for *22* is in some respects one such misreading, playing deliberately with the artifice of the photo-real while exploiting the tools made for making it. But perhaps there are more fundamental destabilizations. Starting at the most technical level, the scene-"graph" library used for all of the work since *Loops* made the hierarchical description of its rendering figures optional — decoupling geometry from the usual tree of transforms; it has exposed only the parts of the underlying graphics hardware that has been called "contemporary" OpenGL — that which places the onus on the programmer to supply the logic that transforms geometry onto the screen and colors it; and it has taken control of the skinning algorithm rather than delegating it to an underlying graphics hardware. The resulting graphics "system" is smaller and more nimble that the real-time, computer-game-inspired graphics engines while lacking support only for large static worlds and complex photorealistic shadows. No particular contribution to the already well-populated world of graphics libraries is claimed here, but the work that follows would have been much harder to even conceive of if it had been constructed in relation to a large, conventional standard scene-graph library.

In the spirit of generalization we construct a small set of primitives which would be powerful enough to build the characters of a computer game, but which, more importantly, are general enough to support other kinds of agent bodies. We start with the primitives: points, lines and planes.

Points are stored movement — they are the markers of raw motion capture, they are the joints of a hierarchical transform creature, they are the positions of vertices of a mesh or the control hull of a smooth surface. Points can be asked for their position with respect to time, are organized into bundles that share a common expected time-base and, crucially, provide notification mechanisms for the appearance, disappearance and change-of-properties of points from the bundle. These notification mechanisms

allow filtered views of bundles to be created inexpensively, and allow points to be offered up by modules and then retracted. Some bundles offer connectivity information that indicates that points are connected to other points in a parent-child relationship (like a bone between joints); some bundles offer extra information about the points (like their speed, or how confident it is in the point's position or an additional rotation).

Lines are the drawn gestures — named, connected curve segments described *imperatively*. Lines are the curves of popular 2-dimensional drawing languages such as Postscript formed by a sequence of moveTo(...), lineTo(...), curveTo(...) instructions. They are externally transient: the primary interface concerning lines isn't one for storing lines, but for accepting instructions as to how to draw a line, although some line acceptors, of course, store data. These line acceptors can be arranged in complex, forking filtering structure and the name of a line often becomes a way of navigating this structure. And, of course, the name identifies this line as a line that exists over multiple execution cycles. Lines have more visual flexibility than points, so there is a context-tree-based stack language for constructing these filter-networks and handling the life-cycle of a line.

Our planar-element representation stores pure topology — vertices, edge, faces and bi-faces (quads). This topological structure is richer than most graphics libraries — holding much more information than is required to send the mesh to the graphics hardware. Sacrificing space efficiency for fast transformability, their interface looks more like the polygonal modeling tools available in a commercial modeling application than a close-to-the-hardware game engine — vertices, edges, faces and bi-faces can be added and deleted while maintaining a topologically sound representation at each level. Like point bundles, they are nexuses of notification about these additions and deletions, carefully batching these notifications for speed and sorting them for consistency.

These three representations are surrounded by a few auxiliary but important classes. **Points**, as sources, are generally façades covering generic radial-basis channels with 3-vector value representations, *page 152*; **line acceptors**, as sinks, generally store lines in channels with a more complex, multi-segment value representation and their stack language is an interesting domain-specific programming technique; and **triangular topologies**, as complex stores, require an additional class to manage the position of their vertices that would enjoy the flexibility of a channel per vertex but in most cases requires a more specialized store if it is to scale to thousands of vertices.

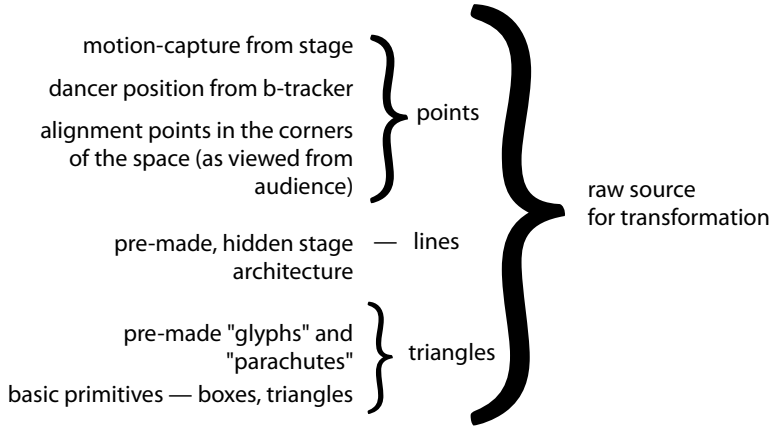


figure 117. The raw material in each of the three graphic languages. Of course, the motion capture from the stage is by far the most present.

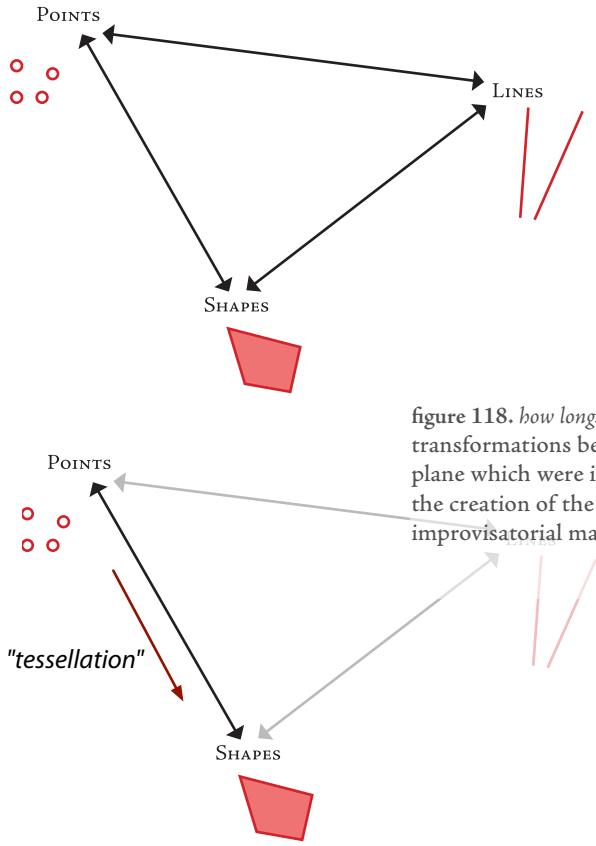


figure 118. *how long...* builds a vocabulary of transformations between point, line and plane which were in the initial stages of the creation of the work, deployed in an improvisatorial manner with live dancers.

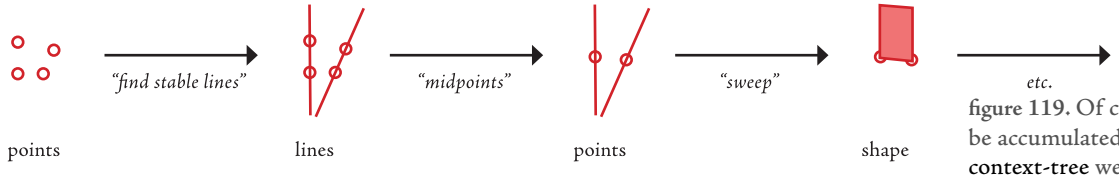
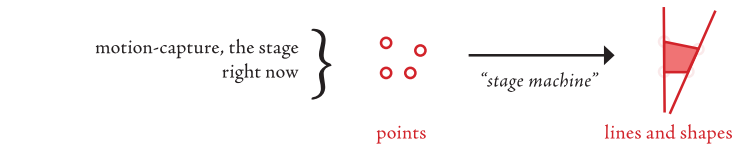
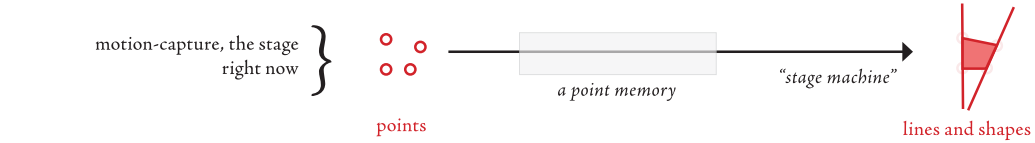


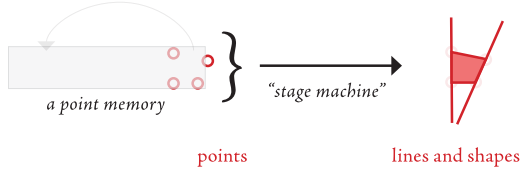
figure 119. Of course, these processes can be accumulated. By using the tricks of the **context-tree** we can stack processes on top of each other while loosely coupling them and the agents that instantiate them.



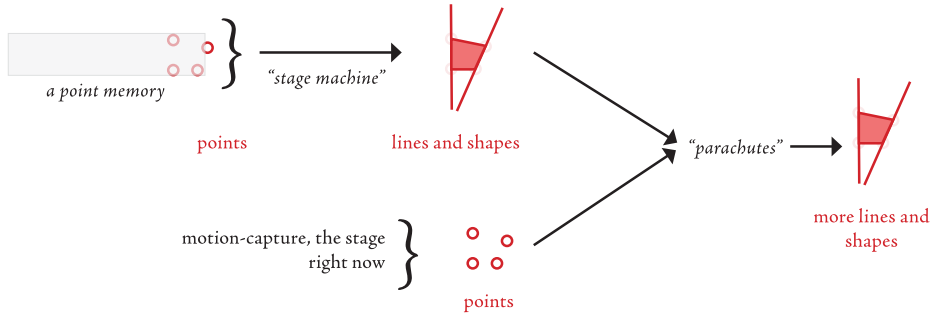
The points on the stage are illustrated by a particular agent, here, the motor system of "stage machine"



Stage machine, for example, records the point movements as it illustrates them.



Suddenly the points break free of the stage, and begin to cycle this captured animation



Another agent takes both the live motion-capture material and the body of the machine as its source material

figure 120. An example ordering of transfer processes — taken from documents shared between the collaborators.

The underlying metaphor for the line is the drawn gesture — it is exchanged between systems as instructions for drawing rather than a stored representation for what is drawn. This runs counter to the graphics-engine tendency (that wants to store something in a persistent place to facilitate its transmission to graphics hardware), but is in line with the successes of resolution-independent drawing interfaces such as postscript / pdf and the drawing models of advanced windowing systems like Cocoa.

As a "gesture", a system accepting a line often faces *life-cycle mismatch* with the code that it is accepting a line from. It might need the line to persist; it might need to update a line that it has previously drawn; it is unlikely that it wants to flash a line on the screen for a single frame — that's not particularly gestural, and it's certainly not the general case. At the same time lines as they are processed can accumulate all kinds of rendering parameters — colors and thicknesses for sure, but in the renderers typically used here also noise and blur parameters, projection parameters and many more. Further, lines are graphically transformable in more interesting ways than points — there's simply more to draw and more time to draw it. Dashed lines, mid-point perpendiculars, lines that connect the ends of lines — lines that trigger these transformations are the technical underpinnings of the "notational" strivings of the agents of *how long*...

The Postscript programming language for drawing — **Adobe Systems Inc**, *The Adobe Postscript Reference Manual, 3rd edition*. Available online at: http://partners.adobe.com/public/developer/ps/index_specs.html

The Portable Document Format — **Adobe Systems Inc**, *PDF Reference, 5th edition*. Available online at: http://partners.adobe.com/public/developer/pdf/index_reference.html

Apple's Cocoa application framework — developer.apple.com/cocoa
In particular the drawing model, shared by Apple's Quartz Compositor: http://developer.apple.com/referencelibrary/GettingStarted/GS_GraphicsImaging/

The line acceptor interface looks like the following:

```
interface LineAcceptor{

    void open();
    void close();

    void beginSpline(String identifier);
    void endSpline();

    void moveTo(Vec3 position);
    void lineTo(Vec3 position);
    void curveTo(Vec3 position, Vec3 control_1, Vec3 control_2);

}
```

Of course we are free to construct line-acceptor filter networks using conventional techniques — building a LineAcceptor that distributes to many LineAcceptors, and passing LineAcceptors into constructors and accessors throughout the codebase. In practice these become a source of considerable coupling between systems. One ends up having to specify many acceptors to act as outputs for a system if it is to draw in a variety of styles. The context-tree here doesn't quite fit, as an indirection method offers indirection on the level of a system not on the level of a line.

A solution is to have a line acceptor dispatch on the basis of the name of the line to a set of sub-contexts named by that line. While these contexts could exist in the main context-namespace, it is more appropriate and powerful to have them in their own local context-tree which *intersects* with the main context-tree, *page 211*. We are now in a position to form programs of filtering elements stored in context-tree-local lists, *pages 220*. What do these program elements accept? Because the begin(*name*), moveTo(...), lineTo(...) ... end() imperative style is convenient for suppliers of lines but inconvenient for accepting filters of lines, these program elements accept a line-storage package that contains all the control nodes (and extra drawing parameter nodes) of the line. This package also forms

the basis for a generic radial-basis value representation. And because filtering is so important for these small programs, the context will also contain the last line-storage package that passed through this context.

Now our elements' interface must support the following operations, with strong life-cycle contract: **open** — called before any other operations, may be nested; **close** — closes a previous open; **filter**(*name*, *inputPackage*, *outputPackage*) — an opportunity to copy information, add rendering parameters etc. to the output line; **shouldCull**(*cull*) — an opportunity to maintain this line even if it is no longer accepted by this line acceptor in this open / close cycle.

Line element programs are composed and altered using the following style in Java (*lc* — "line context", *filter* — the program being constructed);

```
lc.begin("square"); {
        filter.add(new FadeOutOver(10));
        lc.begin("edges"); {
                filter.add(new AddNoise(noiseParameters));
                filter.add(new MarkVertex("../corner", thickness);
        }
        lc.end();
        lc.begin("corner")  {
                filter.add(new Thicken(10))
                filter.add(new LowPass(amount))
                filter.add(new ForceCull(true));
        }
        lc.end()
        filter.output(new Momentum(0.9f))
        filter.output(new OutputTo(dynamicLine))
}
lc.end();
```

And then we can draw a square with:

```
output.begin("square/edges/allOfThem");
output.moveTo(0,0,0);
output.lineTo(10,0,0);
output.lineTo(10,10,0);
output.lineTo(0,10,0);
output.lineTo(0,0,0);
output.end();
```

Lines drawn beneath the above program in the context-tree will get a square, with some per-vertex noise, with the vertices of that line re-rendered in a thicker line which will lag behind the movement of the square. If the above output code stops executing, this square will still be drawn, and follow its previous motion with some momentum, fading out over 10 execution cycles. However, the corner markings will disappear instantaneously.

Because these stack programs are specified instantaneously, local (with respect to the main context-tree) overrides are still possible (*ct* — main context tree):

```
lc.begin("square/edges");
filter.addFirst(new InColor(0,0,1));
lc.end();
```

executed in the root-agent context changes the square edges made by the current creature to blue (but not the square edges of every creature).

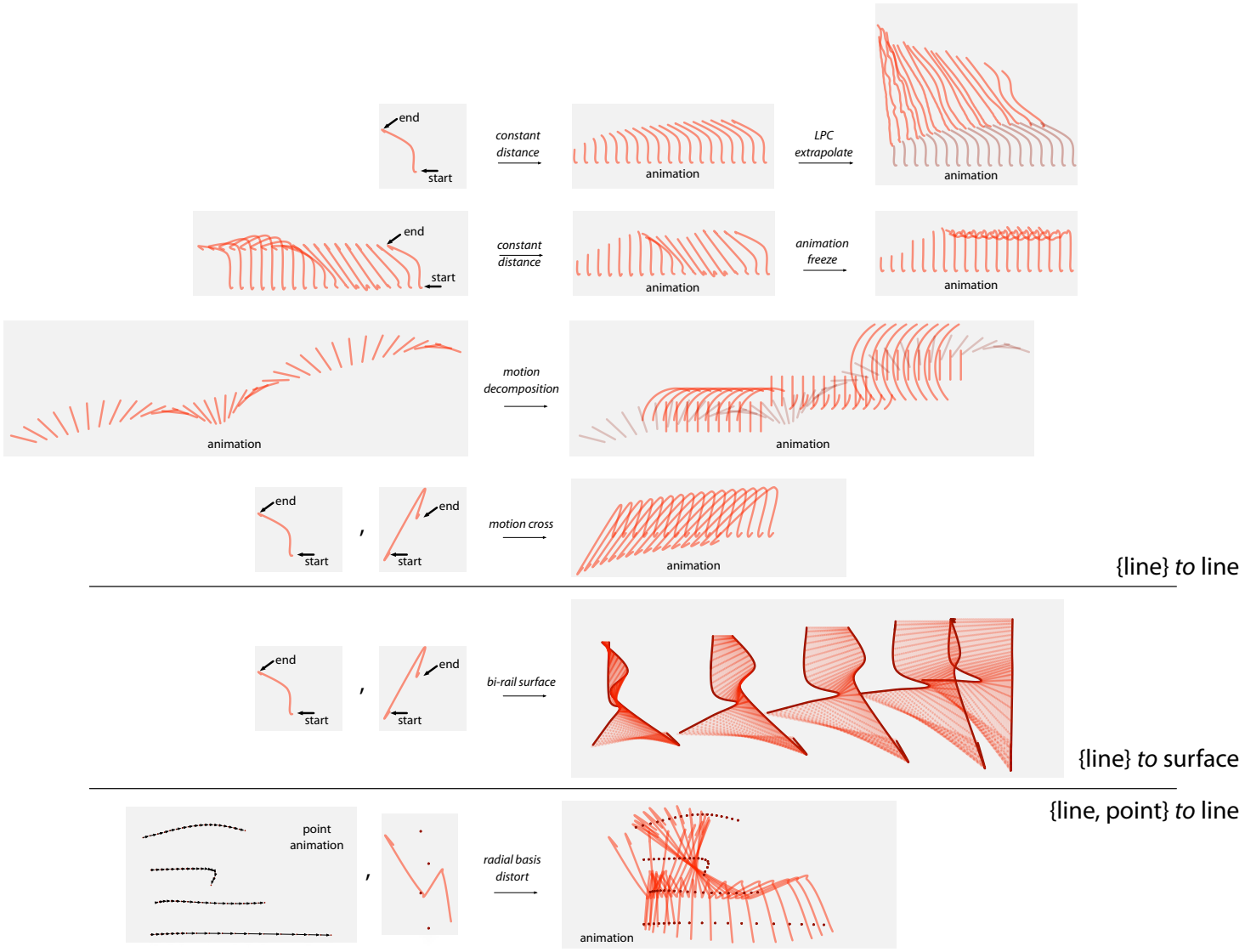


figure 121. Line acceptor primitives convert between (potentially animated) lines to (potentially animated) lines. Other conversions link points and surfaces to linear, gestural forms.

Triangular topology vertex-positioning system — fast "alpha blending" in time and space

Fast computer-graphics rendering systems store the positions of vertices in a large array, often in memory with special properties, and dispatch a list of indices into this array to describe the triangles to be drawn — nothing more than this is needed or considered by the engine. This vertex array is augmented by exactly one extra piece of information, the 3-tuples of indices into this array that define the triangles. Triangular topology stores store only topology, only the information in this index array, and do so in a much less space efficient way — storing, for the sake of fast computation, explicitly the edges, faces and quads of this structure and the network of relationships between them (a face has three edges, a quad, two faces etc.). This has the benefit that it is easy to grow and manipulate geometry live — we shall see uses for this representation in both the *triangle* agent, *page 343*, and the *parachute / accumulation* agent, *page 349*.

None of this stores a vertex *position* — what representation should we use? Computation prohibits the creation of ten-thousand generic radial-basis channels (although around a hundred can certainly make it onto the screen at the same time in both *how long...* and *Imagery for Jeux Deux*) so we need to pick a less general vertex-position representation if we are to work with intricate meshes.

In the common case, code for computer graphics writes and reads directly into the large array of vertex positions, all accesses are immediate and every write access completely overwrites the contents of a specific vertex position. We add two additional axes of storage to this: extending the temporal vocabulary of reading and writing to this array and allowing groups of operations to be blended into the array rather than overwriting it.

Firstly, rather than allowing raw access to this array we form a stack of $0 \ldots N$ auxiliary arrays "on top" of it and couple these arrays on an element-by-element

basis. At each update cycle we take the vertex V at level n, V_n and blend it into the level $n-1$ starting from the top and ending at the lowest array, the array that will be ultimately sent to the graphics hardware:

$$V_{n-1} \leftarrow \alpha_n V_{n-1} + (1-\alpha_n)V_n$$

$$V_{n-2} \leftarrow \alpha_{n-1} V_{n-2} + (1-\alpha_{n-1})V_{n-1}$$

$$\vdots$$

$$V_0 \leftarrow \alpha_1 V_0 + (1-\alpha_1)V_1$$

This cascaded low-pass filter structure is characterized by the number of levels N and the individual filtering constants $\alpha_i = 1 \ldots N$. By writing into this structure at various levels one can achieve various kinds of overlapping, blended animations of vertex positions. For example, writing into level N produces a blend to a new position over a time-scale governed by $\{\alpha_i\}_{i=0}^{i=N}$. This movement is smooth, long and, most importantly, permanent.

By writing into lower levels $\neq N$ one can effect faster, non-permanent changes. At levels near N the changes are soft and shallow, near 0 the changes are rapid (at 0, they are instantaneous) and decay quickly back to the level N state. By writing into multiple levels one can produce rapid permanent changes (for example writing into all levels) or rapid changes that take a long time to decay (writing into levels $0 \rightarrow (N-1)$). All "animations" created by writing by instantaneously writing into only middle levels have both "ease-ins" and "ease-outs" generated for them, without any maintenance. We call this technique temporal-alpha blending, after the alpha blending ubiquitous in computer graphics.

Alpha blending: **T. Porter** and **T. Duff.** *Compositing digital images*. Computer Graphics, 18(3), July 1984.

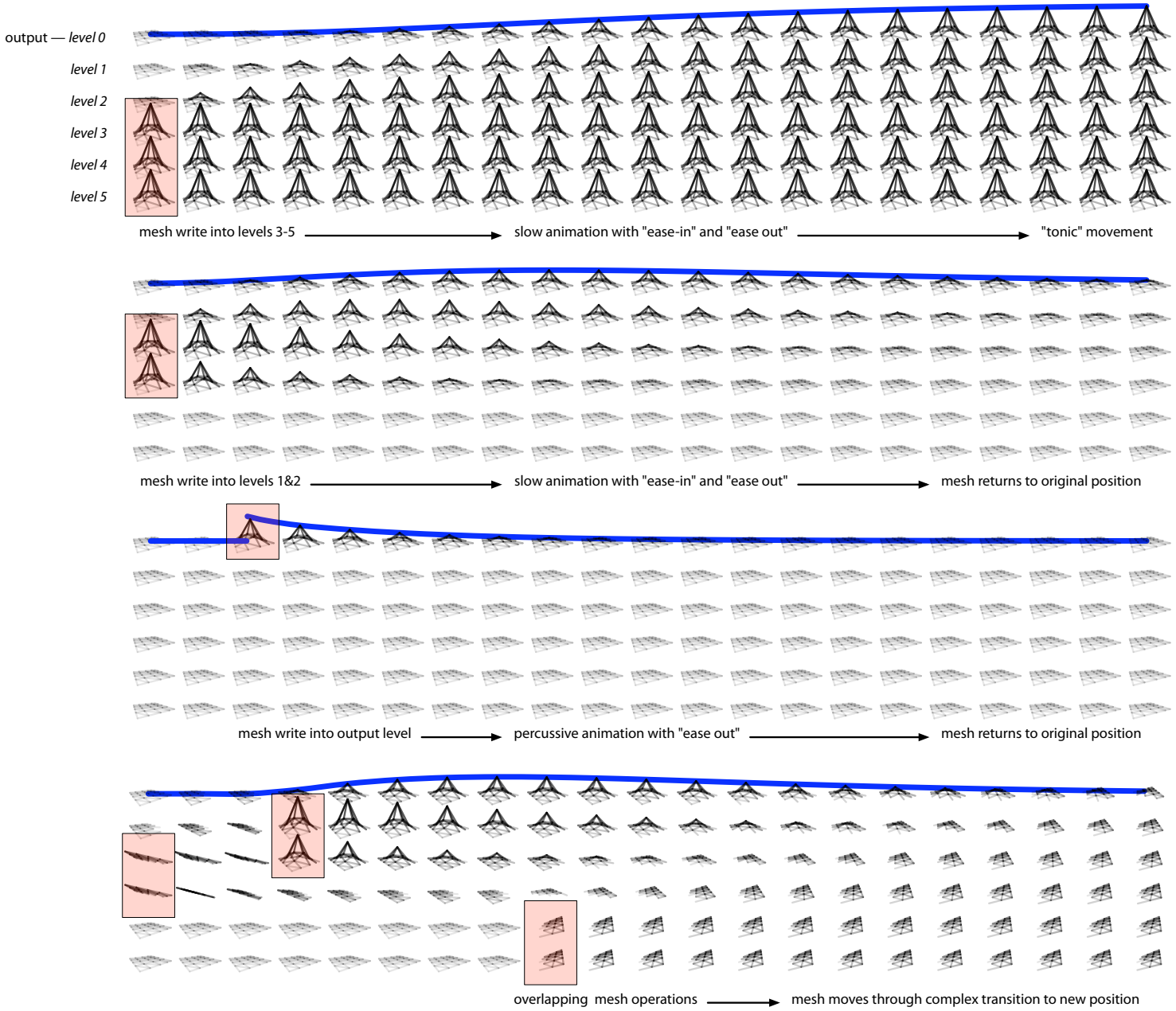


figure 122. By writing into the stack of vertex buffers at various places, smooth, complex or percussive transitions can be blended and overlapped without interaction between the processes that cause them.

Further optimizations can be made by storing the three-dimensional vectors as four vectors — which also happens to better suit the vector-processing units of most contemporary hardware — and using the extra, fourth, floating-point value to store a flag that prevents the update being performed in the case that each vertex in the stack is identical. This helps, but doesn't help as much as storing one flag per cache-line rather than one flag per vertex. Of course, this is rather architecture dependent. But with this level of optimization the structure is computationally inexpensive in the absence of animation, and scales extremely well in the presence of local changes or instantaneous changes.

Unlike the generic radial-basis channels, which use immutable value representations, implicate the context-tree and orchestrate a number of objects together to compute their values, this structure can be very efficiently implemented in bulk on modern processors.

The second extension to this fast-to-draw vertex array is a more traditional "alpha blending" of independent processes. That is, rather than writing over a vertex with a new value, this vertex is written with some weight or alpha and is blended with the old position of the vertex. This way we can execute simply written algorithms that modify vertex positions and easily adapt them to scale back their influence. Such blended influence is at the core of many iterative mesh manipulations in addition to the classical mesh skinning techniques used for graphical characters.

This blending is, of course, trivial to implement, even if we allow for the extra flexibility needed to state which of the N temporal buffers to write into. However, there is a specific implementation problem that is worth working through, for it sheds light on an approach that will be used in other places in this graphics system.

The following (python) code appears to iterate through all faces and move each vertex closer to the center of all the faces that it is part of — this code shrinks meshes to reduce surface area, and appears to mimic several biological and physical grown phenomena:

```
for f in faces:
    center = centerOf(f)
    for v in f.vertices:
        writePositionBlended(v, center, amount)
```

However, this code, tidy as it is, fails in our alpha blending for two reasons. Firstly, writePositionBlended(...) writes immediately to the vertex array that cen-

terOf(...) reads from — the results of this code are dependent on the order of "faces" whereas the intended algorithm isn't; therefore the code is wrong. Secondly, overlapping invocations of writePositionBlended(...), even with no subsequent reads, are order dependent — the last blend having more impact on the final value of the vertex than the first.

Of course, no-one writes code like that. But it ought to be that easy, especially if one is programming in a darkened theater. A better way is to add an auxiliary array to the main vertex array that we are manipulating, and write into this array and read from the main array. And in doing so, we store not 3d vectors but 4d *homogeneous* vectors, where the 4th element is the total weight written so far and the previous three are kept multiplied by this. An alpha-blended write of a vector (x, y, z, α) to an element in the array (x', y', z', α') becomes:

$$x' \leftarrow x' + \alpha x$$

$$y' \leftarrow y' + \alpha y$$

$$z' \leftarrow z' + \alpha z$$

$$\alpha' \leftarrow \alpha' + \alpha$$

to convert this (x', y', z', α') to an actual 4-vector we form $(x'/\alpha', y'/\alpha', z'/\alpha', 1)$

Of course, the code above need not care that this extra array exists and, in fact, inspection will prove that the above code executes correctly even if this structure exists. We call the act of making this copy an open and the act of writing this copy back onto the main array a close. Opening takes no parameters. Closing takes two — α_{mul} and α_{add} — that control how much the auxiliary A array gets to effect the main array V in the following equation:

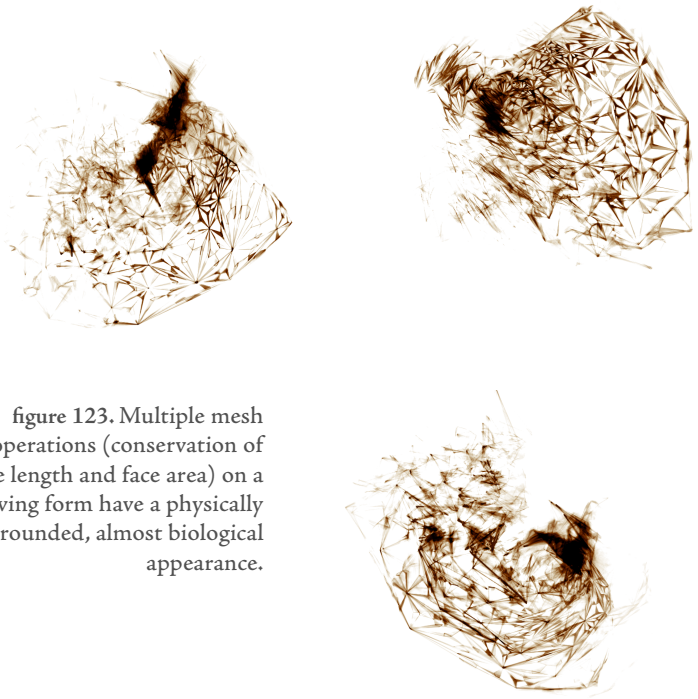


figure 123. Multiple mesh operations (conservation of edge length and face area) on a moving form have a physically grounded, almost biological appearance.

$$\alpha_e = \alpha' \alpha_{\text{mul}} + \alpha_{\text{add}}$$

α_e is clamped to $0 \ldots 1$, and:

$$V \leftarrow \alpha_e A + (1 - \alpha_e) V$$

The caller to the above code can wrap the invocation in an open...close bracket, and we can dispense with the "amount" parameter to this method altogether:

```
open()
moveToCenter()
close(0, amount)
```

is equivalent.

Of course, opens and closes can nest; we can form an auxiliary array A^2 to an auxiliary array A^1. In this case close has four parameters. The last two control what happens to the alpha component of the underlying array:

$$\alpha_e = \alpha' \alpha_{\text{mul}} + \alpha_{\text{add}}$$

$$A^1_{xyz} \leftarrow \alpha_e A^2_{xyz} + (1 - \alpha_e) A^1_{xyz}$$

$$\alpha_{\alpha,e} = \alpha' \alpha_{\alpha,\text{mul}} + \alpha_{\alpha,\text{add}}$$

$$A^1_\alpha \leftarrow \alpha_{\alpha,e} A^2_\alpha + (1 - \alpha_{\alpha,e}) A^1_\alpha$$

Finally, we note that since reading always occurs from previous openings, an opening does not necessarily imply a copying of the vertex data and in many cases a zeroing of the auxiliary array suffices. For completeness, we add parameter to the open(...) to control the amount written to each of the $1 \ldots N$ temporal buffers, should this be the first open in the chain.

These two mesh representations when combined, although lacking complete generality, allow a range of very compactly defined manipulations of the contents of the vertex array to be overlapped in time. In particular they allow the convenient recasting of mesh-*operations* into mesh-*processes*: as controllable, adaptive constraints or general composable animation generators. We are free to look to three-dimensional modeling packages that each have a great many mesh-operations (and this is well explored terrain, only a handful of the library of mesh operations of *how long...* are "new" in this sense), code them quickly and robustly, and then wrap and layer these operations inside this temporal alpha-blending framework to shape the animation of their operation.

The spaces on stage

We now have a strong general purpose framework for synthesizing and rendering controllable geometry from captured motion. One final ingredient is missing.

Unlike previous indirect methods of sensing dancers in live performance (for example gyroscopes, accelerometers or single video camera) real-time motion capture deals solely in terms of absolute, calibrated position. That is, the marker data that are provided by the hardware locate particular points, in real millimeters from a known origin. This provides a unique opportunity when it comes to projecting these positions back out onto the stage if we can work out how to transform "real pixels" back into "real millimeters".

Unfortunately, we lack a good way of projecting into a three-dimensional volume, so despite this accuracy and richness, we remain dependent on a computer-graphical trompe l'oeil to make the three-dimensional graphics, when projected on a very two-dimensional transparent surface (a "scrim") in front of the dancers, appear to occupy the same space as the dancers. Part of the "effect"

is purely stagecraft for sure — should any light fall on the scrim the effect is immediately absent and the imagery will read as flat no matter what is projected.

One graphical trick deployed in *how long...* and, to a lesser extent in *Imagery for Jeux Deux*, is a rediscovery of an old and traditional computer-graphical technique — depth cueing. In *how long...*, unlike with traditional distance "fog", the distance to the virtual camera shades not only the intensity of the linear material drawn but the hue, noise amplitude and transparency.

The noise amplitude is particularly important — left unscaled by distance, the typically sketchy quality of my randomly perturbed line actually works against the correct cueing of depth. As lines in the foreground, with the same amount of world-space noise added to them, appear to move more they overlap less and therefore, after the action of sequential frame motion blur, appear fainter. Scaling the noise amplitude and the opacity by a function of distance prevents this inversion or, alternatively, turns it into an apparently finite depth of field (by pushing more distant material into noisier territory as well).

Further, although the color palette of *how long...* appears to be an austere and diagrammatic monochrome + red, there is in fact almost no pure white in the work — rather each line is quietly hue-shifted towards blue (receding) or red (advancing). The difference between having these techniques and not is extremely apparent in the perceptual depth of the imagery as it hangs in the space of the stage even though the colors themselves are barely perceptible.

Such tricks work for any three-dimensional rendering projected into the space. However, in *how long...* we need an extra ingredient — a mapping from the three-dimensional space of the motion-capture data into a three-dimensional

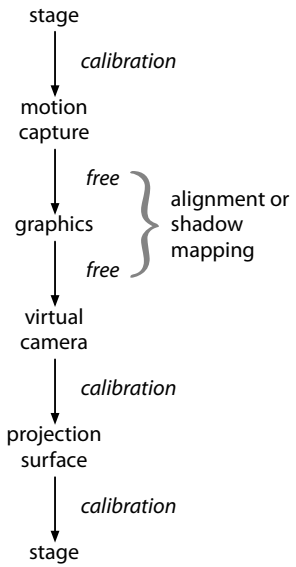


figure 124. The complete stack of coordinate system transformations that are possible when using real-time motion capture in theater. In order to complete this loop we need to know the corners of the projection screen, and the coordinate system of the motion-capture hardware in the same frame of reference as the theater.

"world-coordinate" space of the renderer which gets projected (mathematically) onto the image plane which in turn gets projected (physically) onto the scrim.

How Long... is built with a vocabulary of such mappings and they come in two species — the *audience-aligned mapping*, and the *shadow-projection mapping*.

The audience-aligned mapping is so called because, for a particular seat (at a particular height) in the auditorium the imagery and the dancers align exactly, for all positions of the dancer on the stage. As one moves away from this ideal seat the alignment grows less and less exact (though the imagery is still startlingly correlated). The position of the projections on the surface of the scrim is measured (by hand) prior to the performance. The mapping takes the points in real space and intersects the line between the marker and the auditorium seat with the scrim. This provides a two-dimensional representation of the marker data, in virtual camera coordinates. This two-dimensional point-set is given new depth, along the lines between the points and the virtual camera position, proportional to the distance of the real-space point and the real-space scrim. A "plan view" of the stage can be created using the same mathematics, only by locating the virtual audience member above the stage and the "scrim" at the floor plane of the theater.

The shadow projection uses the same mathematics, but a different interpretation: it places the audience position on stage. In this case, this position acts as a virtual light source casting a shadow of geometry onto the scrim. This two-dimensional image is then given depth again by offsetting it from the virtual camera plane by a distance proportional to the distance from the virtual light source. It is, in effect, a three-dimensional shadow — figures close to the virtual light source loom large and close to the front of the stage — that plays with the secret materiality of the hidden projection surface.

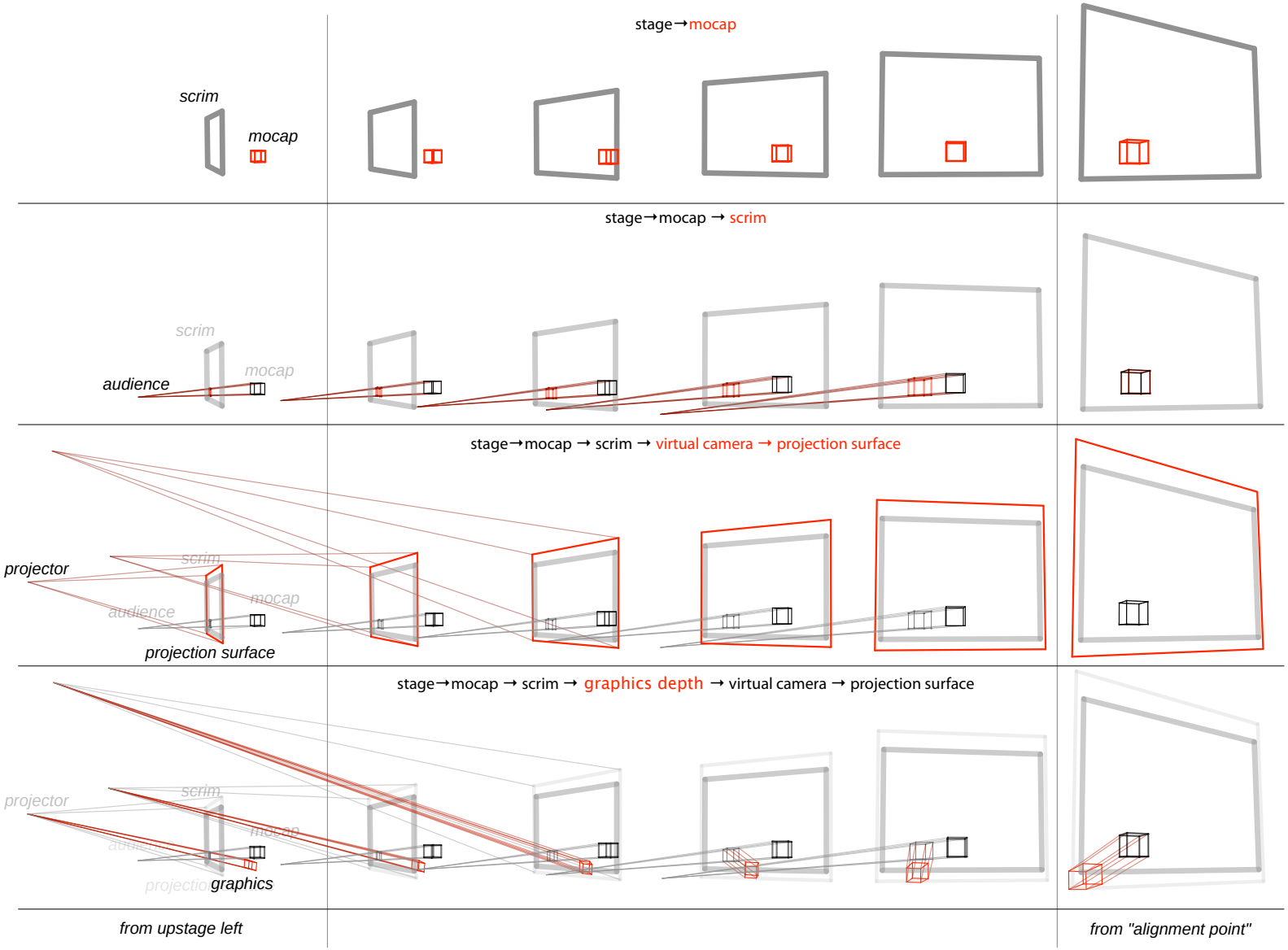


figure 125. A camera orbit of an object on stage with imagery projected onto a scrim in the front of the stage. From the alignment point the imagery point aligns exactly; from the point of view of the graphics world, the object has an oddly distorted perspective.

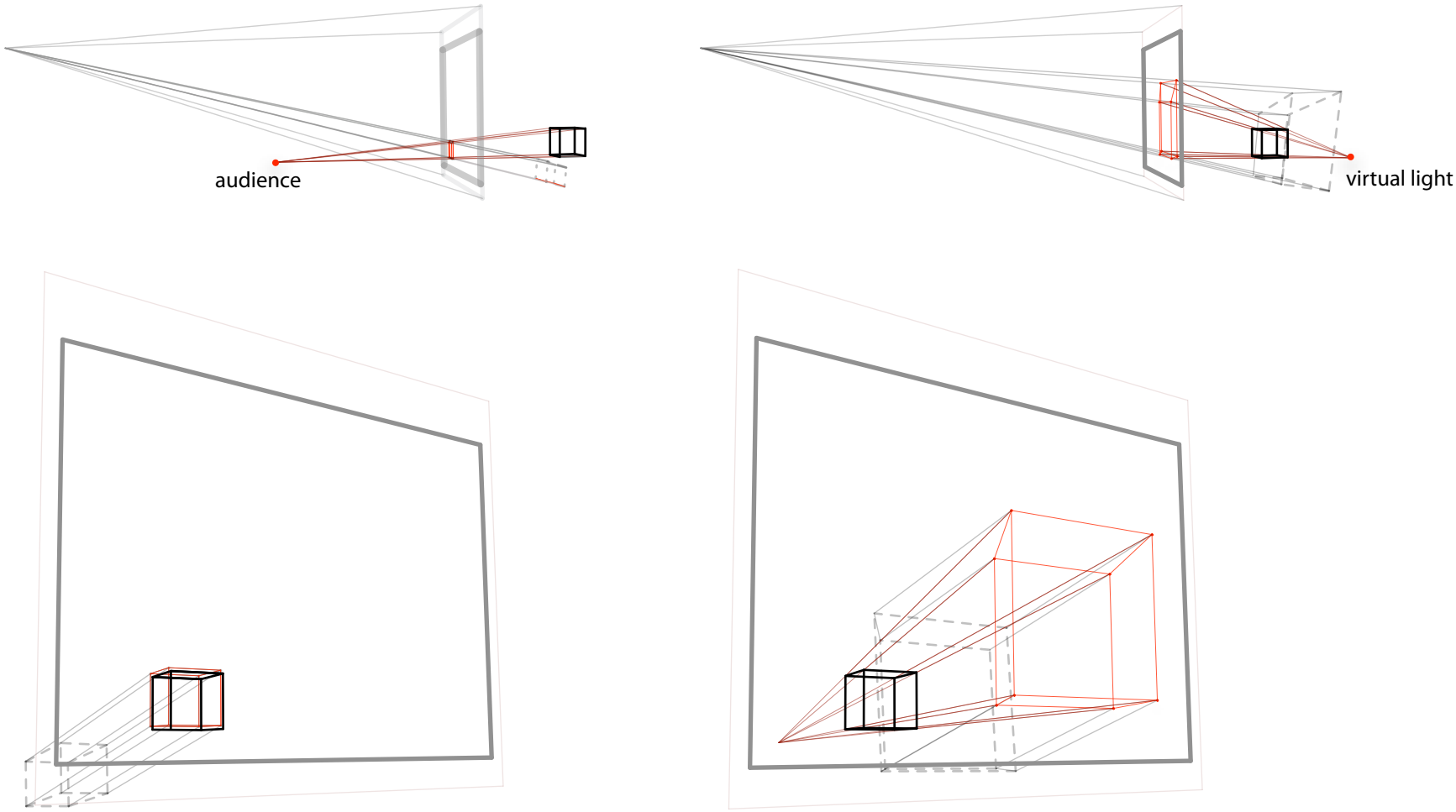


figure 126. The audience-aligned and shadow-projection mappings compared. An identical geometric construction yields dramatically different results. In the latter, objects and figures loom large and transiently on the scrim.

These two classes of mappings, or projected projections, find a balance between the readable and the dramatic — the audience can see the connection between both the movement of the images and dancers and, at times, the overlap between their positions. Deploying these coordinate systems throughout a piece becomes the problem of distributing clarity throughout the hall. It is a matter for considerable, but exciting, future work to adapt these techniques to the availability of multiple projection surfaces in richer (but, alas, less tourable) stage configurations.

3. The agents deployed in *how long...*

The power of this graphical framework comes not from the non-photorealistic shaders or the number of stock elements that can be composed in the line acceptor stacks to make complicated notations, but from the processes that move between the representations of points, lines and planes. *How Long...* necessarily starts with points — the points of motion from the dancers on the stage — but becomes a sustained trope about the recording of movement and the transformation into drawn line and the possibility of solid form. This chapter concludes with a description of the more complex agents, the details of their implementation, and a sketch of how they overlap.

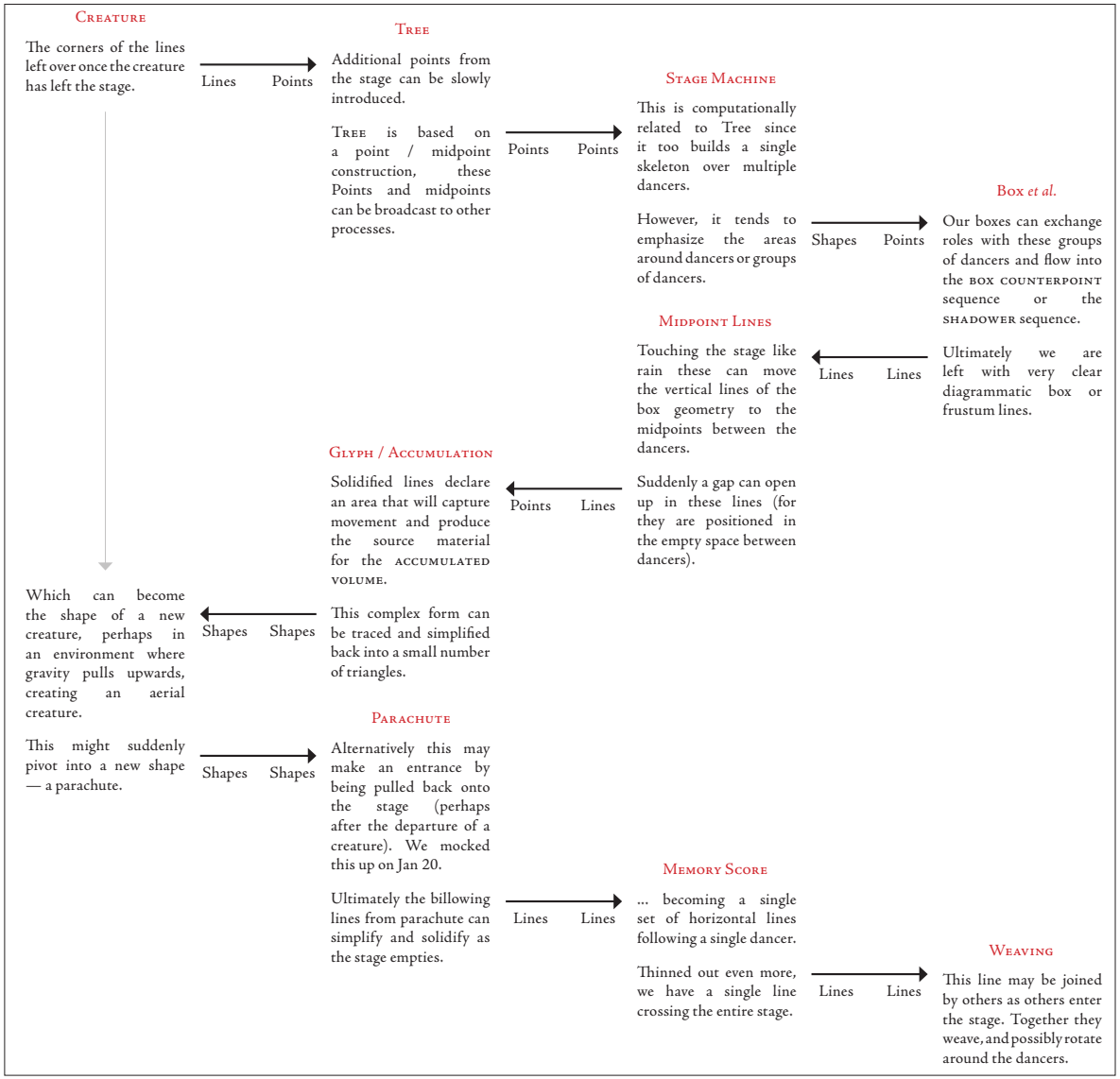


figure 127. An example "flow" through the *how long...* agent framework. This was the state of the piece for a series of workshops. The finished work adds a final return to the triangle here labeled "creature" — taken from documents shared between the collaborators.

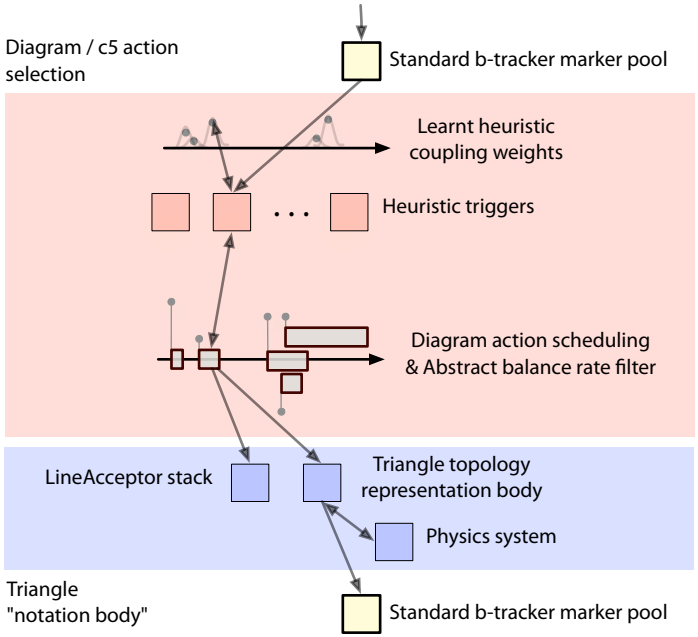


figure 128. The *triangle* agent diagram.

The piece opens with a creature stage-right playing a simple game with a simple goal — to make it over to stage-left. However, the only source of motion available to the creature comes from the movement of the dancers on the stage. In order to begin to make progress it needs to hitch a ride on the motion available, connecting and disconnecting from the markers it sees in order to pull and be pushed leftwards. The creature's body, initially a line supported by single triangle, accretes the diagrammatic traces of these connections, disconnections and movements as the creature continues.

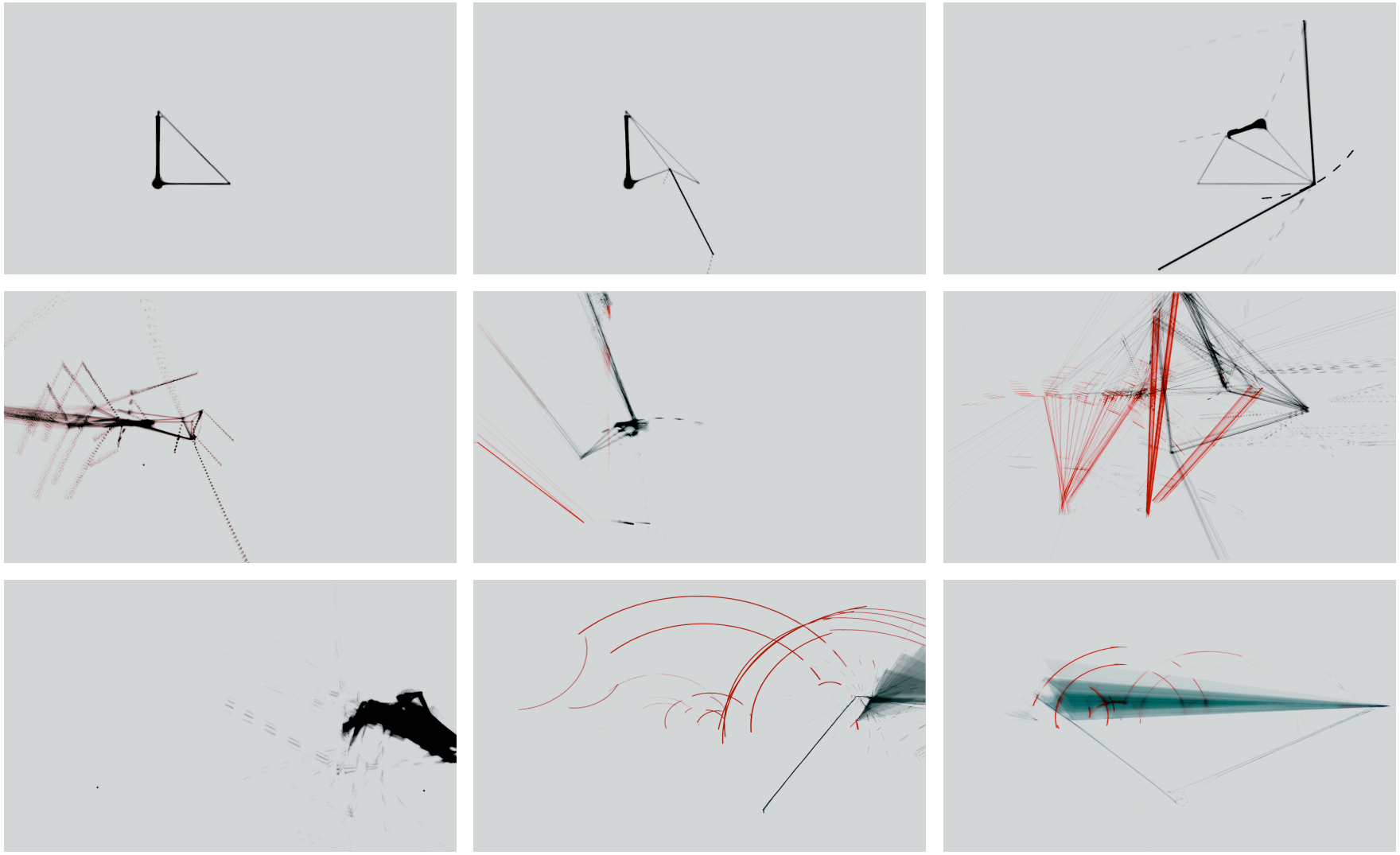


figure 129. The *triangle*. The first seven images are from various openings of the piece. A re-projection rendered line element picks out one path through the triangular framework. The last two images are from the "reprise" of triangle. (inverted).

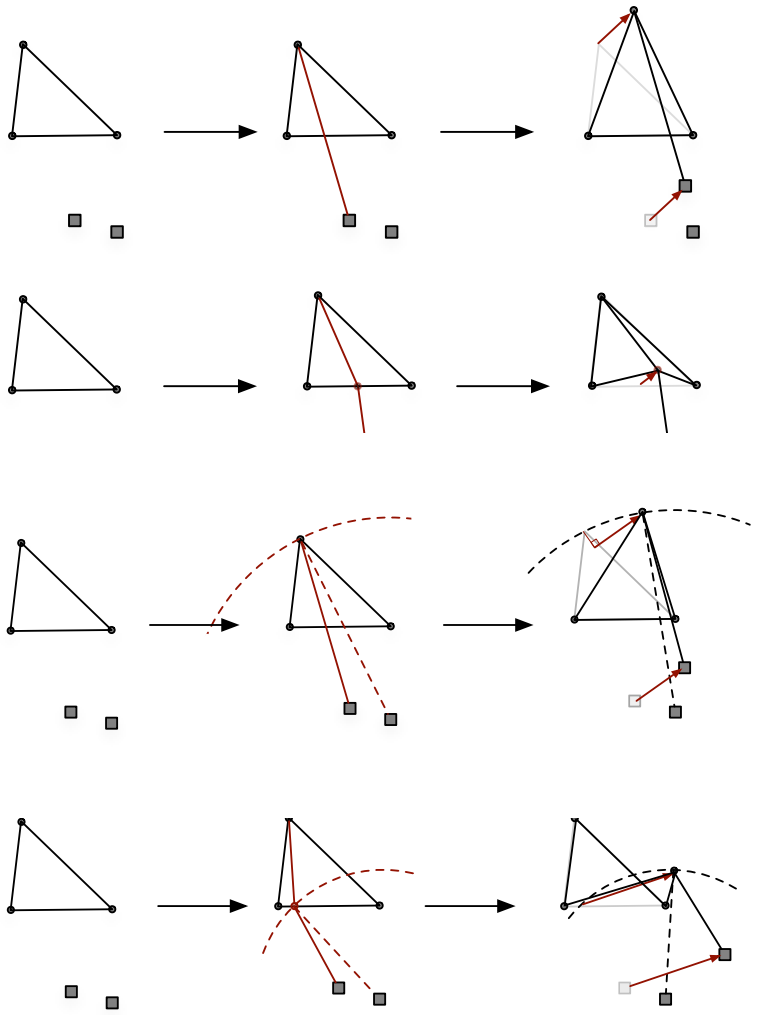


figure 130. The four classes of actions that *triangle* can take on one or two points: pull vertex, split-and-pull vertex, rotate vertex (constrained to be on a circle from another point) and split-and-rotate vertex.

There is a palette of four operations that the creature can ask of its body. Each operation can be applied to any vertex (or any pair) of the body — this multiplies the number of actions — and when disconnected each operation leaves, in addition to any new edges, vertices and faces that the operation creates, a trace that the operation took place which is respecified local to the coordinate system of that part of the creature.

The body exists in a simple physically simulated world — it falls to a ground plane, maintains angular momentum when it falls or is pulled over — and simultaneously is trying to conserve average edge length and face area (through our triangular operations framework described above). The simulation is distorted slightly to allow increased stability for this generally flat structure in the plane parallel to the screen (if the triangle were to fall over towards the audience, all they would see is a line). The creature succeeds (and during the performances, and dress-rehearsals it has never failed) because of a few simple heuristics, encoded into the structure of its action system, and "prior knowledge" of the choreography.

The action system of triangle is implemented within the Diagram framework, *page 251* — trigger factories produce the operations listed above in response to monitoring a b-tracker based marker tracker that scores markers based on whether they are going in the right direction or not. Actions are scheduled closely in the output channel and remain active — lengthening their markers — while the markers that they are attached to remain scored above a threshold.

What remains to be *learnt* — from exposure to the choreography — is rather simple: how strongly to couple these scores with the triggers that create action, and where to set the thresholds that ultimately cause the operations to disconnect from the markers. The representation that the destination for the learning is a channel of examples — a generic radial-basis channel with a time-base given

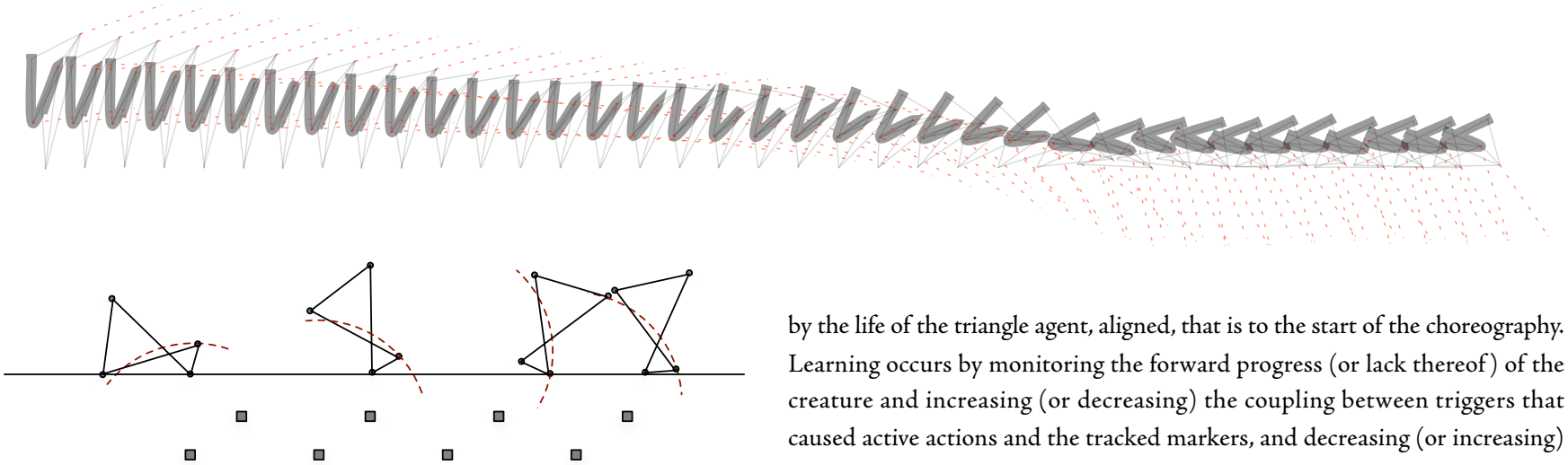


figure 131. Once actions have completed, traces of the operation remain connected to the body of *triangle*. This body is located in a simple physics simulation; thus, while it attempts to conserve edge length and face area it is simultaneously falling towards a hidden ground-plane.

by the life of the triangle agent, aligned, that is to the start of the choreography. Learning occurs by monitoring the forward progress (or lack thereof) of the creature and increasing (or decreasing) the coupling between triggers that caused active actions and the tracked markers, and decreasing (or increasing) the threshold for the active actions to withdraw. These marks occur in the channel itself, at the onset times of actions participating, so they are tied to the context of the choreography — assuming of course, that it does not change structurally. Ultimately this learning problem, as deployed in the opening choreography of *how long...* is not particularly difficult, as Brown responds to the agent's presence on the stage with material that appears to toy with the creature's intention but ultimately moves from stage right to stage left, the structure succeeds in capturing what it needs from the underlying motion.

Variations of the triangle agent appear, in different renderings and multiples during the piece — one alternative has its physics system modified, such that net rotational movement from the markers directly causes rotational momentum on the creature, forcing it to roll horizontally away from the dance which now it must attempt to cling onto. Similar, although much easier, learning occurred for this creature too.

While the design of this creature's learning strategies are much less sophisticated than that of say, *Dobie*, there are two points to note in any comparison. Firstly, the integration of a scored temporal element seems vital in "choreographing" rather than "demonstrating" learning. Secondly, the primary construction of a *triangle* action system took place in a single afternoon, the realization and reconfiguration for the alternative instantiations of the creature took place during a break in a rehearsal. Returning to the language of chapter 1, it might be tempting to compare this *experimental* learning, deployed in the moment, explored during the creative processes, with *Dobie's* carefully thought through, crafted and framed, dog-learning *avant-garde*. However, in all seriousness, this technical feat is simply the product of Diagram framework's insistence on an explicit articulation of time, *page 251*, the context-tree's power for allowing duplication-with-modification of agents, *page 222*, and *Fluid's* utility in allowing complex systems to be tested and tuned live, *page 387*.

figure 132. *Triangle* learns when to connect and disconnect from the passing dancers by rehearsing with the choreography in order to move from stage right to left. This figure shows the successive stages of learning (from top to bottom). Space is rotated for the purposes of presentation. Stage right becomes the top of each image (inverted)

It is a general principle that if there were no dancers, or they did not move, not much of the imagery would appear and even less of it would move. The next set of creatures — the *parachutes* and the *accumulations* — also capture motion from the dancers on the stage, similar to the triangles. However rather than deliberately connecting to a handful of points these creatures connect to the points en masse and we develop a new structure, a new kind of "animation" to control the relationship between these creatures' bodies and the bodies on stage. Just as *The Music Creatures* were a silent potential for music, until sound was heard in the gallery, this animation is an "open form", a movement-less animation that acts as a series of arranged receptacles for manipulated motion.

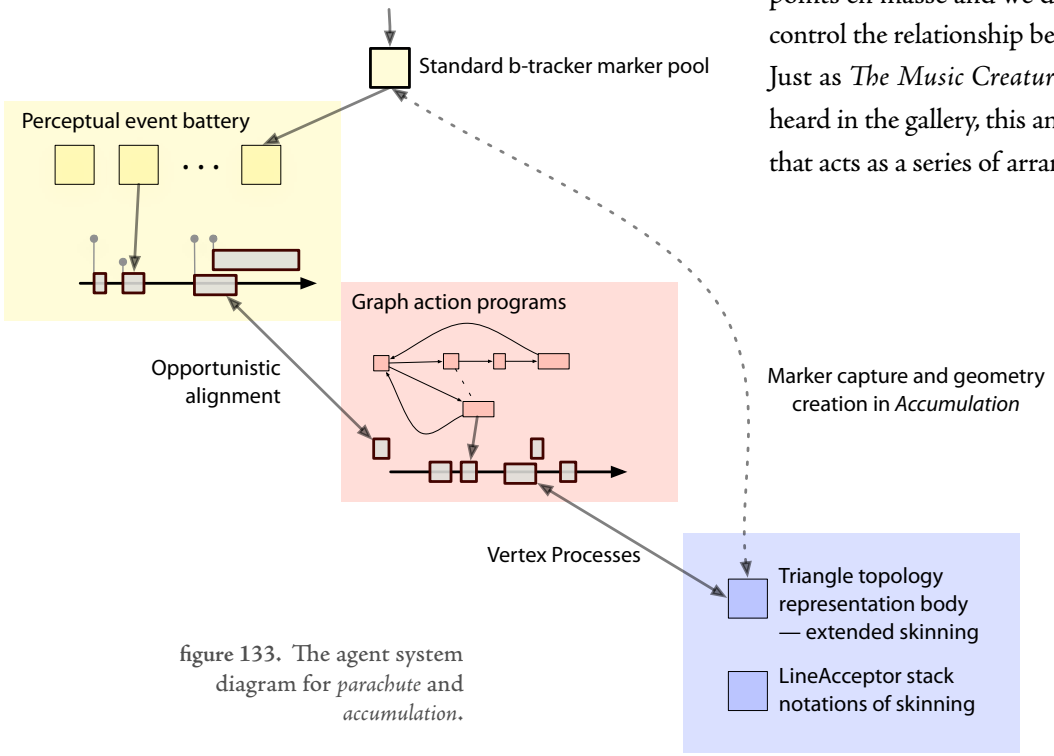


figure 133. The agent system diagram for *parachute* and *accumulation*.

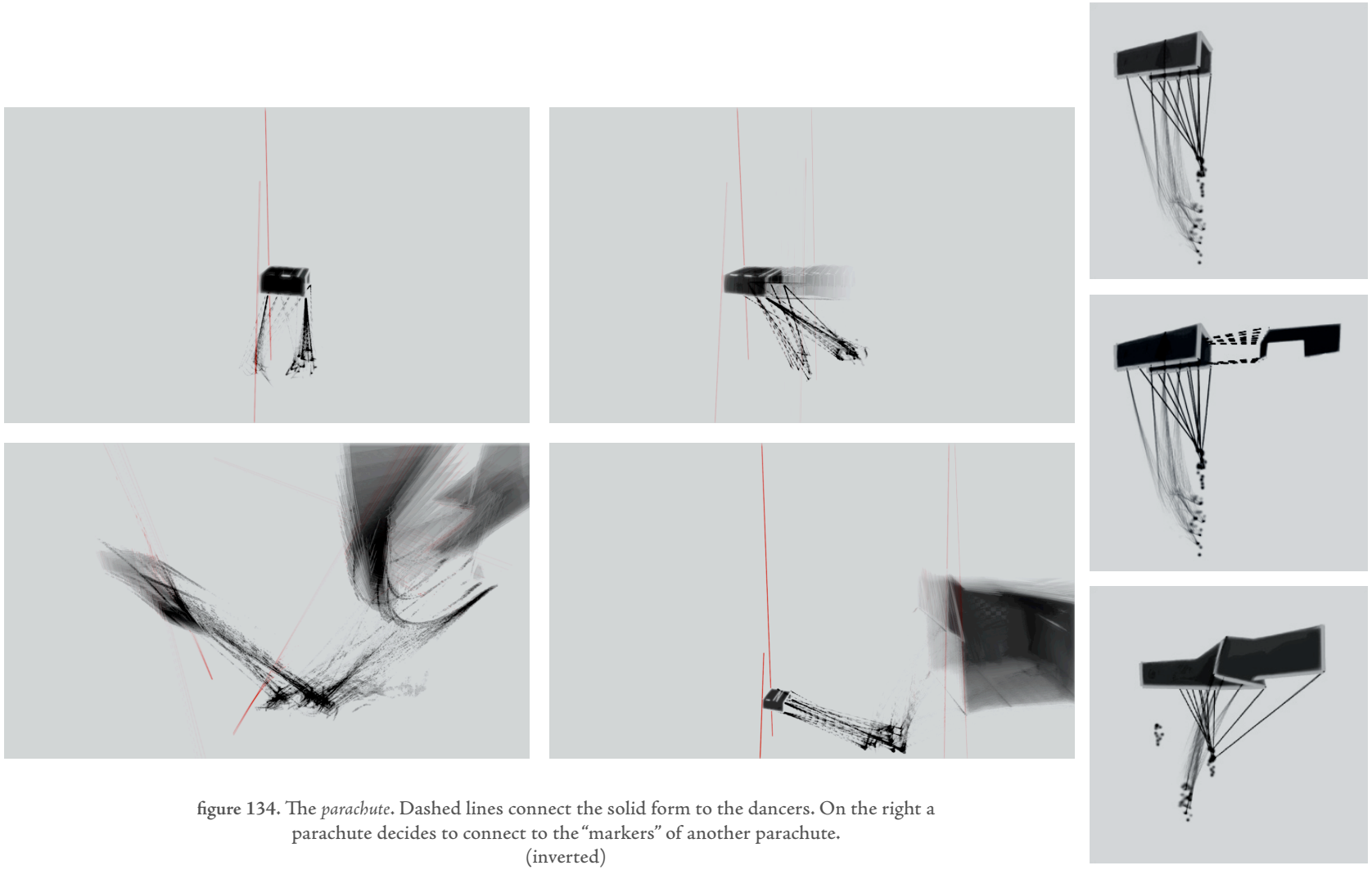


figure 134. The *parachute*. Dashed lines connect the solid form to the dancers. On the right a parachute decides to connect to the "markers" of another parachute.
(inverted)

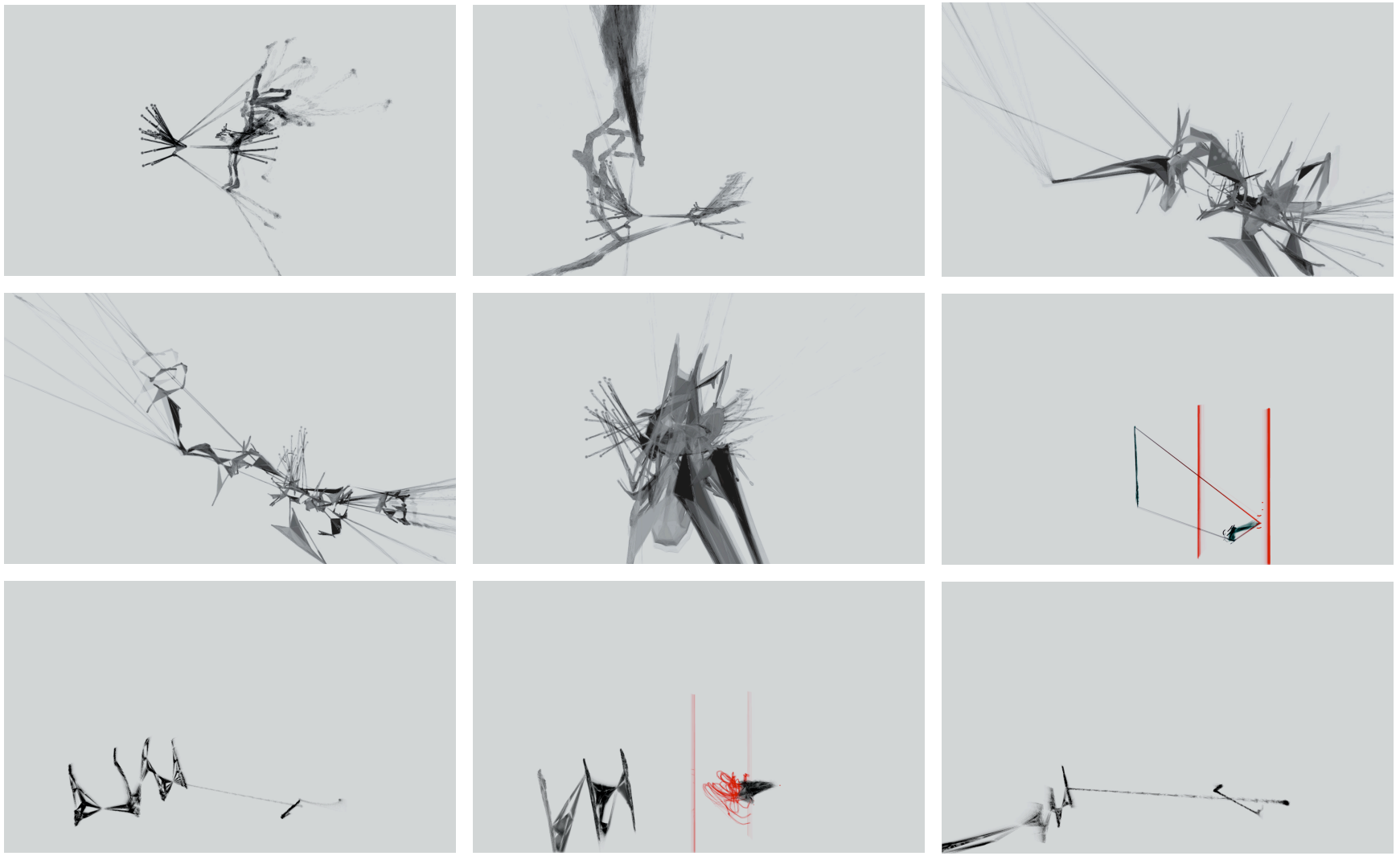


figure 135. The *accumulation*. The red lines indicate the sampling of movement material, and the creation of geometry, from the stage. (inverted)

Delaunay tetrahedralization is a three-dimensional version of the well known Delaunay triangulation algorithm —
E. W. Weisstein. *Delaunay Triangulation*.
From *MathWorld—A Wolfram Web Resource*.
http://mathworld.wolfram.com/DelaunayTriangulation.html

The implementation used here is to be found in **David Eberly's** *Wild Magic* graphics and algorithms library:
http://www.geometrictools.com/

A *parachute* is a simple, box-like form that appears in space. By translating and scaling itself to cover all of the dancers it forms a visible connection to each marker. The maintenance of these markers becomes a top-down influence on the parachute's perception system and these markers are continually updated and distributed across the actual current marker set. Subsequent deformations of the parachute shape are driven by a rhythmic exploration of a space of possible triangular topology vertex positioning operations. For *parachute*, these operations focus on generalizations of the standard "mesh-skinning" algorithm commonly used for digital characters. An *accumulation* is similar, but it takes its form from a rather dramatically captured motion — it is constructed as a distorted Delaunay tetrahedalization of successive slices through markers motion captured between two moving lines on the stage.

The standard skinning algorithm is the following. We capture a set of vertex positions (the skin) and set of transforms (the joints) at a particular moment — this is the "binding information". At any time there is a set of weights for each vertex that connect the vertex to these transformations.

for the bind positions of the mesh $\{v_i\}$, the bind-time transformations M_j , the current transformations M'_j and the weights $w_{i\to j}$ we can compute new positions $\{v'_i\}$ by weighted sum:

$$v'_i \leftarrow \sum_j \left(w_{i\to j} \cdot M'_j \cdot M_j^{-1} v_i \right) / \sum_j w_{i\to j}$$

For a more conventional overview of character skinning, the related work section of: **J. P. Lewis, M. Cordner, N. Fong.** *Pose Space Deformations: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation*. In Proceedings of ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series. 2000.

Further, since this idea has been canonized and hardware accelerated, both of the leading low level graphics libraries expose functionality for at least "matrix-palette" skinning. **DirectX** — http://msdn.microsoft.com/directx/, and **OpenGL** — http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_blend.txt

This mathematical expression is equivalent to a rather more interesting formulation. Conventional skinning is a set of blended, overlapping operations — one for each transform as above. However we can reinterpret this as a set of dynamic processes that are each trying to maintain a *constraint*. In the case of skinning the constraint is that vertices should be in the same local position as when they were bound, regardless of the new position and orientation of the transform.

This view allows us to integrate "skinning" into the blendable body framework, and create animations from the *process* of skinning, with the process of skinning attempting to maintain its constraints. It also suggests decompositions (generalizations) of the skinning algorithm.

For example: we call 2 sets of *positions* in space — a set of marker positions (from the source) and a set of vertex positions (from the parachute geometry) — a *binding*. (Without further work there are no rotations labeled in the markers, they are pure positions). We can restate the conventional (translation only) skinning algorithm easily within the vertex-positioning framework, and we can make temporally smooth movement by writing into higher levels of the temporal buffer stack. But there are other constraints — for one, we can keep the same distance :

for the bind positions of the mesh $\{v_i\}$, the bind-time positions P_j , the current positions P'_j and the weights $w_{i\to j}$ we can compute new positions $\{v'_i\}$ by weighted sum:

$$v'_i \leftarrow \sum_j w_{i\to j} \left[v_i + \left(v_i - P'_j\right) \left(\left|v_i - P_j\right| / \left|v_i - P'_j\right| - 1 \right) \right] / \sum_j w_{i\to j}$$

Further, we could let the distance change, but keep the same *orientation* with respect to the marker position:

for the bind positions of the mesh $\{v_i\}$, the bind-time positions P_j, the current positions P'_j and the weights $w_{i\to j}$ we can compute new positions $\{v'_i\}$ by weighted sum:

$$v'_i \leftarrow \sum_j w_{i\to j}\left[P'_j + (v_i - P'_j)\cdot Q'_{i\to j}\cdot(v_i - P'_j)\right] / \sum_j w_{i\to j}$$

where the quaternion $Q'_{i\to j}$ is given by:

$$Q'_{i\to j} = Q(v_i - P_j; v_i - P'_j)$$

that is the quaternion that rotates the vector $v_i - P'_j$ onto $v_i - P_j$.

There are local constraints too that we can apply, without reference to the marker bind positions, but simply to the vertex bind positions. Some of these have to do with the properties of the topology placed on top of these vertices — can try to maintain the edge lengths of a mesh and the face areas of a mesh, or, more conflictingly, we can try to reduce the face (surface) area of the mesh while trying to preserve edge lengths. More interestingly we can look at local relationships between the bind positions — similar to the angle representation used in *line* in *The music creatures*, *161*. Applying this operation allows *parachute* to coalesce its shape after many transformations, but possibly in a new position and orientation and in a rather indirect way. In any case: each constraint gets applied inside the temporal alpha-blending system to the body of the agent; these processes unfold over time or are stated and retracted by this system.

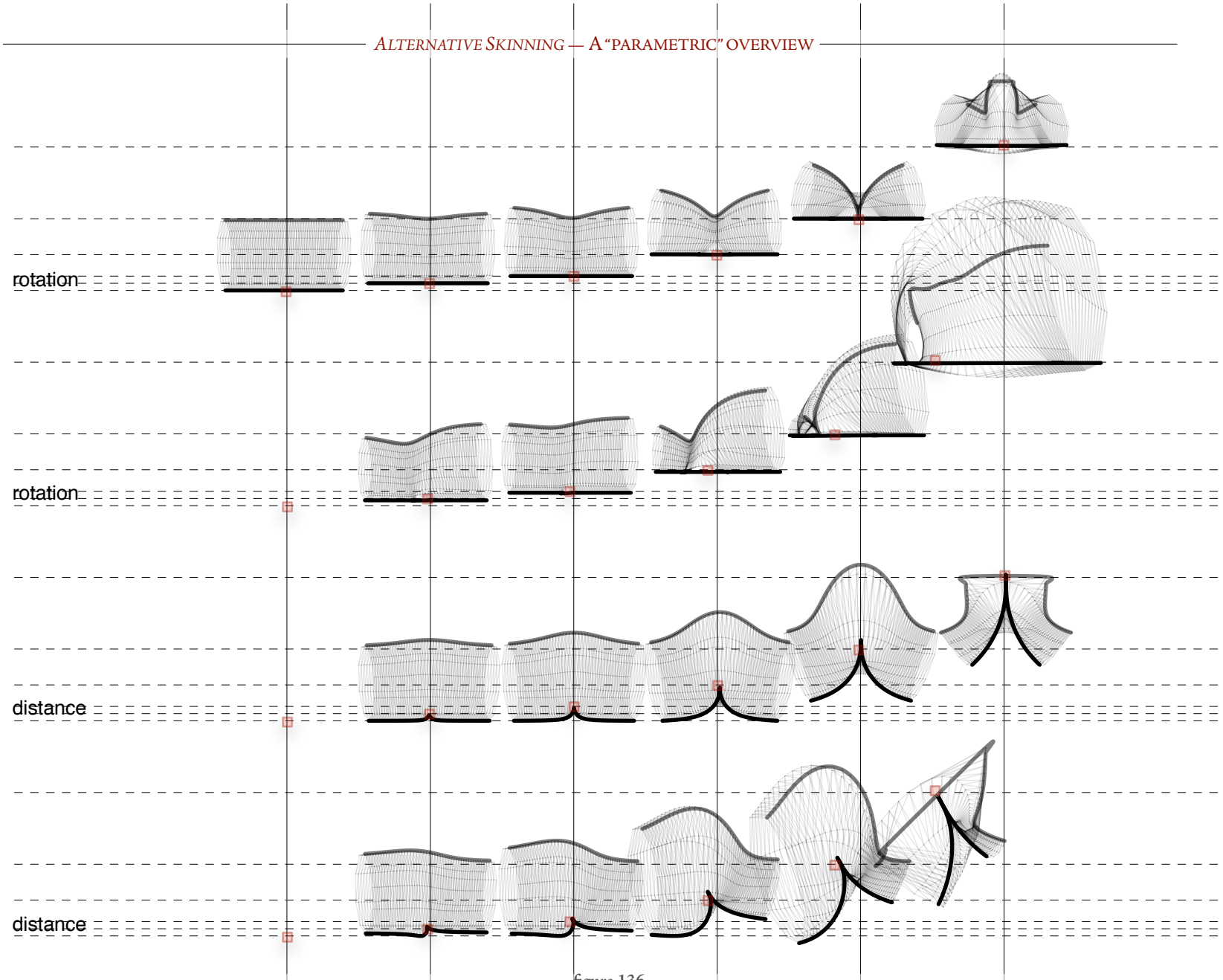


figure 136.
A single "skinning" control point applied to an initially undeformed cylinder — two classes of constraint are shown, in the upper level of each the control point —shown in red — is moved vertically upwards, in the lower, the control point is moved diagonally.

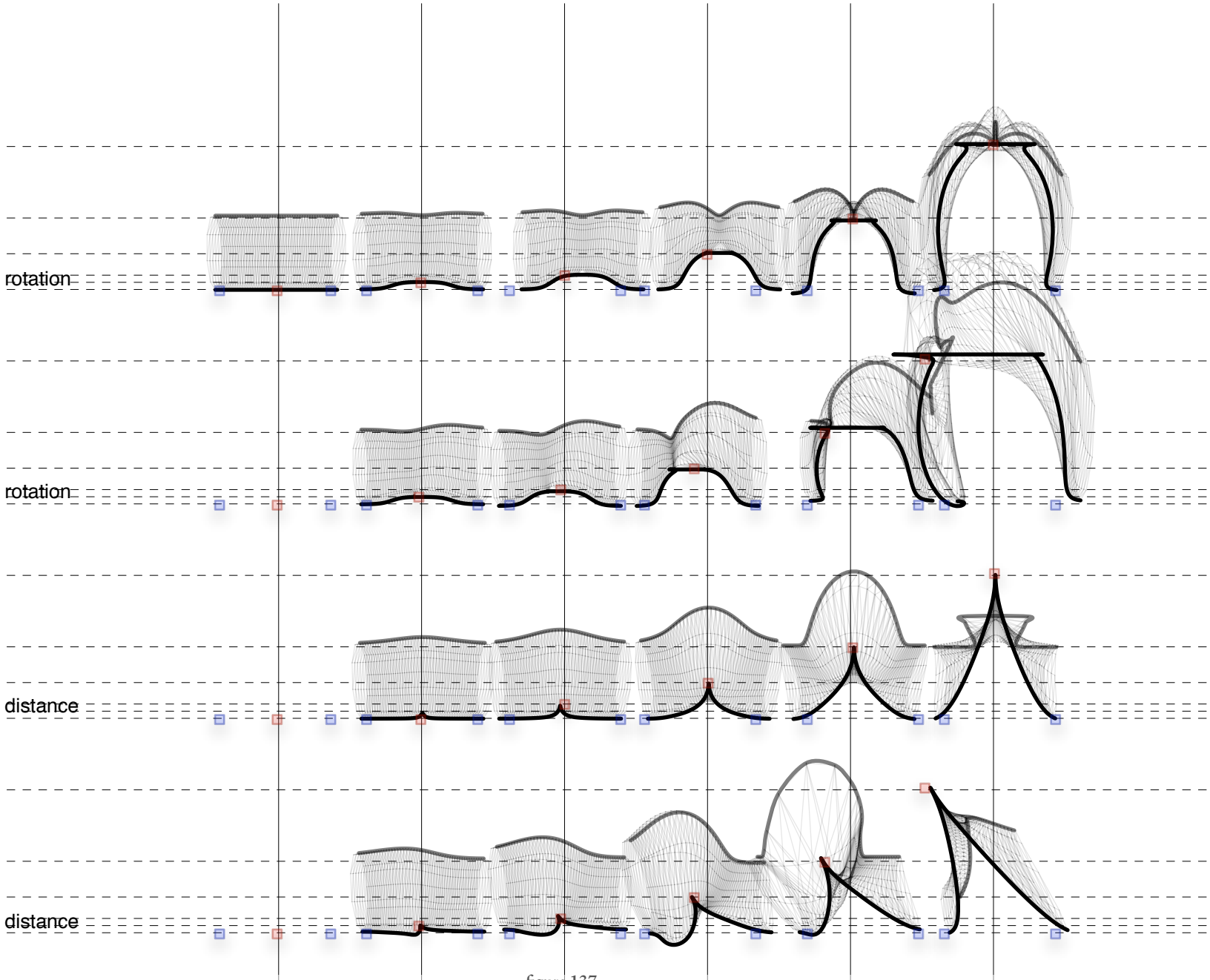


figure 137.
Similar to the previous figure, however here two additional "translation" constraints are added to each side of the original constraint. These act to keep nearby parts of the cylinder "undeformed". Of course, for more exotic constraints the implications of the "undeformed" can be quite complex.

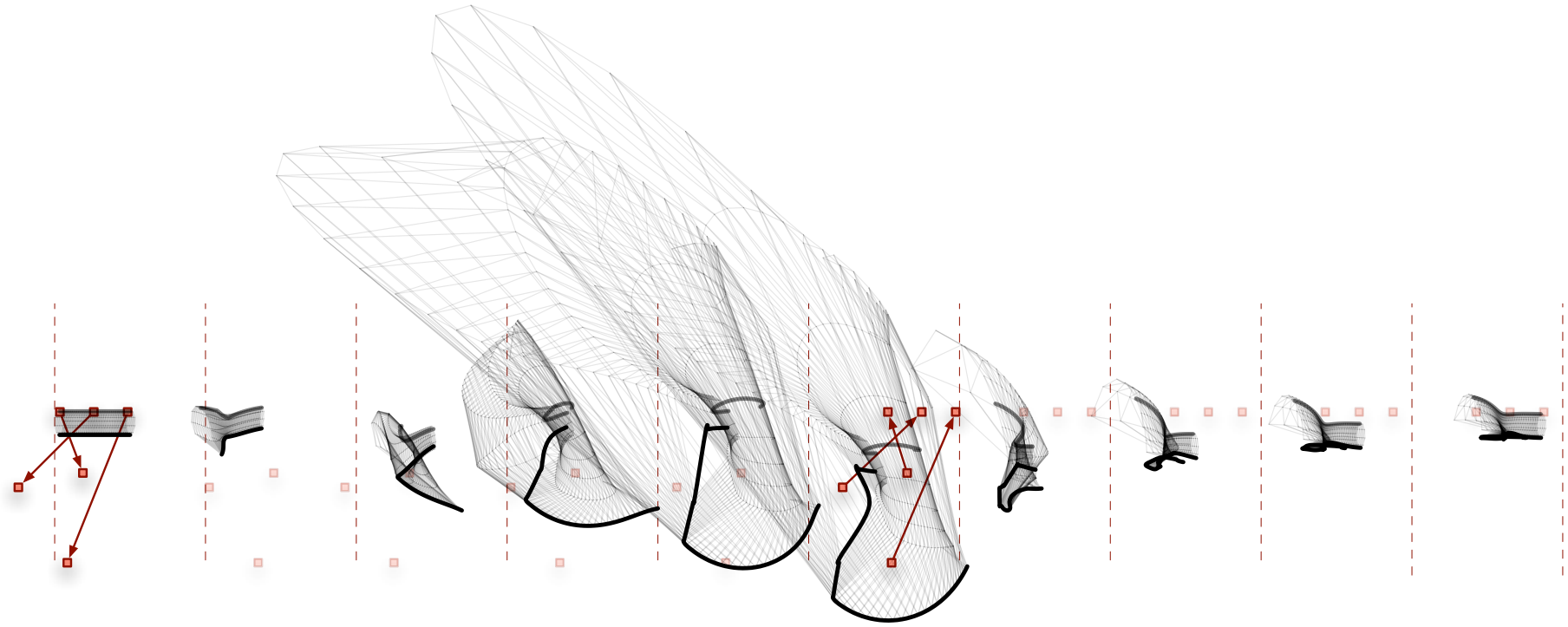


figure 138.
Because these skinning constraints are processes not operations, they necessarily produce animations not new static forms. The above figure shows an animation created by two successive, instantaneous movements of the control points. Because of the nested capture() and release() calls used to create this animation, the mesh does not return to its original configuration, but rather retains some of the trace of its animation.

To navigate the forest generated by this framework we clearly need to impose some structure on these operations. One container interface that has proven to be widely applicable is the "capture/release" interface :

```
interface CaptureRelease {

    void release(float amount);
    void capture(float amount);

    void merge(float amount, CaptureRelease from);
    CaptureRelease fork();

}
```

This interface does a good job of wrapping motor-system level constraints, including all of the processes described above. It works well for other constraints

such as inverse-kinematic (IK) constraints applied to a digital figure and, since this is a less exotic constraint, I'll use this as the illustrative example.

Consider making an IK-based "feet-sliding constrainer" — this was done for a small bipedal character called Max (unconnected with *how long...*, see figure 100, *page 289*) that was driven by an otherwise "conventional" pose-graph motor system but had purely procedural turning animations. This means that Max turned by playing a walk-forward animation, while being rotated clockwise or anti-clockwise around the vertical axis. One result of this is that Max's feet may well slide during his animation.

Indeed, it has become a typical strategy computer graphics to procedurally manipulate character animation (blending animations or otherwise transforming them, perhaps in response to user control or offline optimizations) and then use inverse kinematics on chains down each leg to prevent the feet from appearing to slide. To stop a foot slide the constraint tries to remove motion parallel to the ground plane but not perpendicular. In order to allow forward motion of the character at all, the constraint should be applied only when the foot is in contact with the ground. And because computer graphics is not as exact as real physics, there will be a little fuzziness in our idea of the ground, so the constraint should fade in and out as the foot nears the contact plane. This is equivalent to fading in (to 1) and out (to 0) the amount that we release() the results of computing the IK to the virtual skeleton and root node. However, the creature is being propelled by a force that is not under the control of the foot-sliding constrainer — the constrainer needs to update its idea of where the foot should be. This is what the amount associated with capture is for, it controls how much the virtual skeleton influences the constrainer's idea of its target. In this case this will decrease from 1, perhaps to 0 or to some low number, before increasing to 1 as the foot comes toward the floor and leaves. The remaining methods — fork and

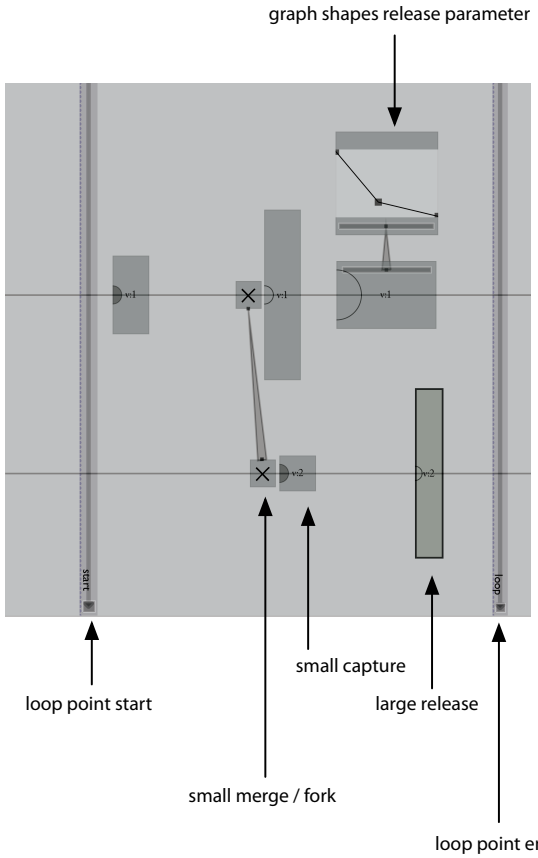


figure 139. The graphical environment *Fluid* (discussed in the next chapter) can be used to create these small, rhythmic pattern generators, scripting the application of the skinning and other mesh operations. These scores can be "unrolled" into diagram channels and opportunistically aligned with events gathered from the stage.

merge — allow the internal state of the constraining object to be duplicated and at some later point blended with another constraint.

We can wrap all of the above transformations in capture/release mechanisms. Clearly for our rather odd creature bodies the ordering, flow and amount parameters of these mechanisms will dominate their movement. We construct a purely meta-procedural idea of an animation by building an interactive notation for the chaining of capture/release structures. In these diagrams time runs left to right, horizontal lines show the same constraint, markings on the line indicate captures or releases. Time-synchronous graphs help provide *amount* parameters. Dashed lines mark forks and merges.

As we shall see, the creation of these domain-specific, executable notations is something that the Fluid framework has been specifically designed for.

Each of these diagrams specifies an open-animation, rhythmic cell that when repeated results in a complex expansion, tracking, dissolution and condensation of the *parachute* geometry. That the contents of the cell form connections to the movement enforces a relationship, however shifting, with the motion on the stage. The cells can be named, and called upon by a small action system that, in the case of the *parachute*, oscillates between three such cells, or motor programs each of around 10 seconds in duration; in the case of the shorter-lived accumulation there is only one cell.

However, although the cells are made by essentially opening and closing windows onto the motion of the dancers, their internal organization in this formulation simply remains unchanged regardless of the motion of the stage. As an agent metaphor, this is a motor system without an action system, an open, yet somehow "ballistic" form. Is there a way of coupling these overlapping cells without losing the notational fluidity, without coupling the notation to a par-

ticular choreography, or losing the sense of the cell's identity?

The first step to this perpendicular relationship is to look for ways of bending the repeated passage of time through the cell. Notations, such as those seen above (constructed in *Fluid*), can be "unrolled" into a Diagram channel or arbitrary length. This lets us bring the pattern matching and temporal manipulations to bear on the execution time and possibly ordering of these cells.

To complete this coupling we need something to couple these cells to. For this we construct a perceptual stream of "significant events" from the markers on the stage. A list of event recognizers are easy to synthesize from the agents' lower-level perceptions of the markers on the stage and from the dancer-like clusters and from a rough look at the choreography. They are: number of dancers changing; high acceleration maxima from a number of points; acceleration minima at low velocity for a number of points; sudden drop in the height of the points. Additionally, a couple of key marker configurations are identified.

These unspecified ideas can be sharpened up (in an unsupervised, automatic way) using the learning database techniques constructed for *The Music Creatures, page 143*, (specifically, they are a mapping from an unspecified input domain to a (0,1) range with the bottom 0.5 of the range cut off and ignored.

In order to identify marker configurations, the shape matching algorithm presented in:

D.P. Huttenlocher, W.J. Rucklidge, *A multi-resolution technique for comparing images using the Hausdorff distance*", Proceedings of Computer Vision and Pattern Recognition, 1993.

was implemented. This is easily applied over the entire set of marker hypotheses in the ongoing b-tracker, to create a more reliable identification of a moment of choreography. Such choreographic tracking, however, was never placed "in the critical path" of *how long...* (unlike the pose recognition of 22, for example).

Event generators that look for maxima and minima do so by parabola fitting and sometimes take a number of execution cycles to fit a clean parabola through the quantity, so events can be added to the fixed stream at temporal locations other than "now". This opportunistic matching for parachute runs with a latency of around half a second (of course, the actual geometric operations update at the much smaller tracking latency of the perception system). Such complicated layering of update rates is easily expressed within the Diagram system.

Now that we have a stream of mobile future events, from the unrolling of the rhythmic cells, and a stream of fixed near real-time perceptual events, we can incrementally search and shift the the mobile future events to maximize the coincidences between high "value" perceptual events and the elements of the cell. In doing so we maintain, of course, the ordering of the cell elements. *Fluid* allows for a labeling of other constraints on its diagrams and these too can be unrolled into the Diagram channel — the changes to the position of one cell ripple outward to all future cells. Although the prototype notation from *Fluid* could be copied out indefinitely, we choose instead to duplicate the previous cell from the Diagram channel itself. In this way, in the absence of structurally important events from the stage, the cell repeats its previous incarnation.

This animation technique conspires to align the changes of intention of the *parachute* creature — which is what the elements of these rhythmic cells are legible as — with notable events on the stage, if those events are present, and otherwise maintains a coherent but slow tempo. This allows coordination between two complex systems — the *parachute* / *accumulation* structure and the structure of the choreography as observed in very visible and simple ways — without specification; serendipity without chance.

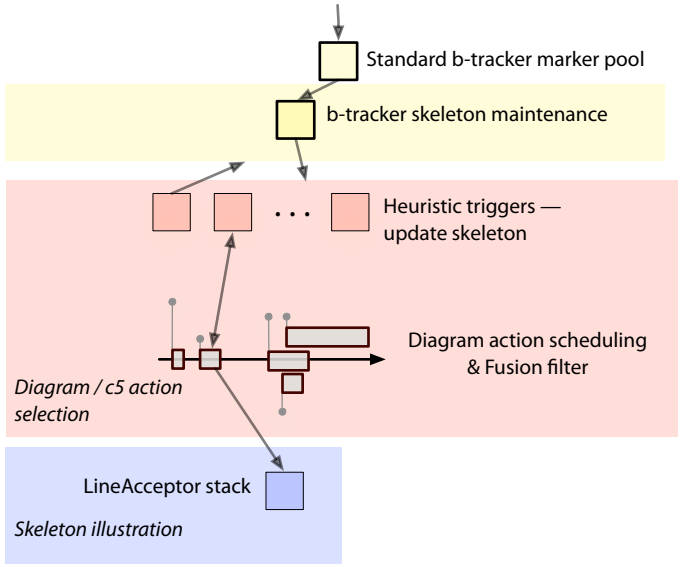


figure 140. The general agent system diagram for *tree* and *stage machine* and *forest fire*. Only the skeleton representation and the line acceptor stacks differ significantly — the agents share a common "base class".

Both parachute and accumulation are structures that react to and trace movement — they are "live memories" of the dance. The next class of agent enters into the choreography in a different way, offering alternative motivations and partial explanations for the movement as it unfolds.

Three agents are alternative ways of drawing a *skeleton*: for a figure or for a stage. They are three relatively straightforward studies exploiting the notational potential of the blendable body framework. ***Tree*** constructs a skeleton for an offline captured solo and re-injects it into the piece (partially overlapping with the performance of the solo itself); ***Stage machine*** hypothesizes a skeleton for all of the markers on the stage; ***Forest fire*** flows edges over an invisible lattice built from an instantaneous snapshot of the markers, revisiting the percolative action-selection dynamics of *Loops*.

Each agent fails to grasp its goal repeatedly — there is no such thing as a skeleton for a whole stage of dancers, but rather a series of plausible constraints that the dancers are obeying at any particular moment that the agent can fleetingly illustrate.

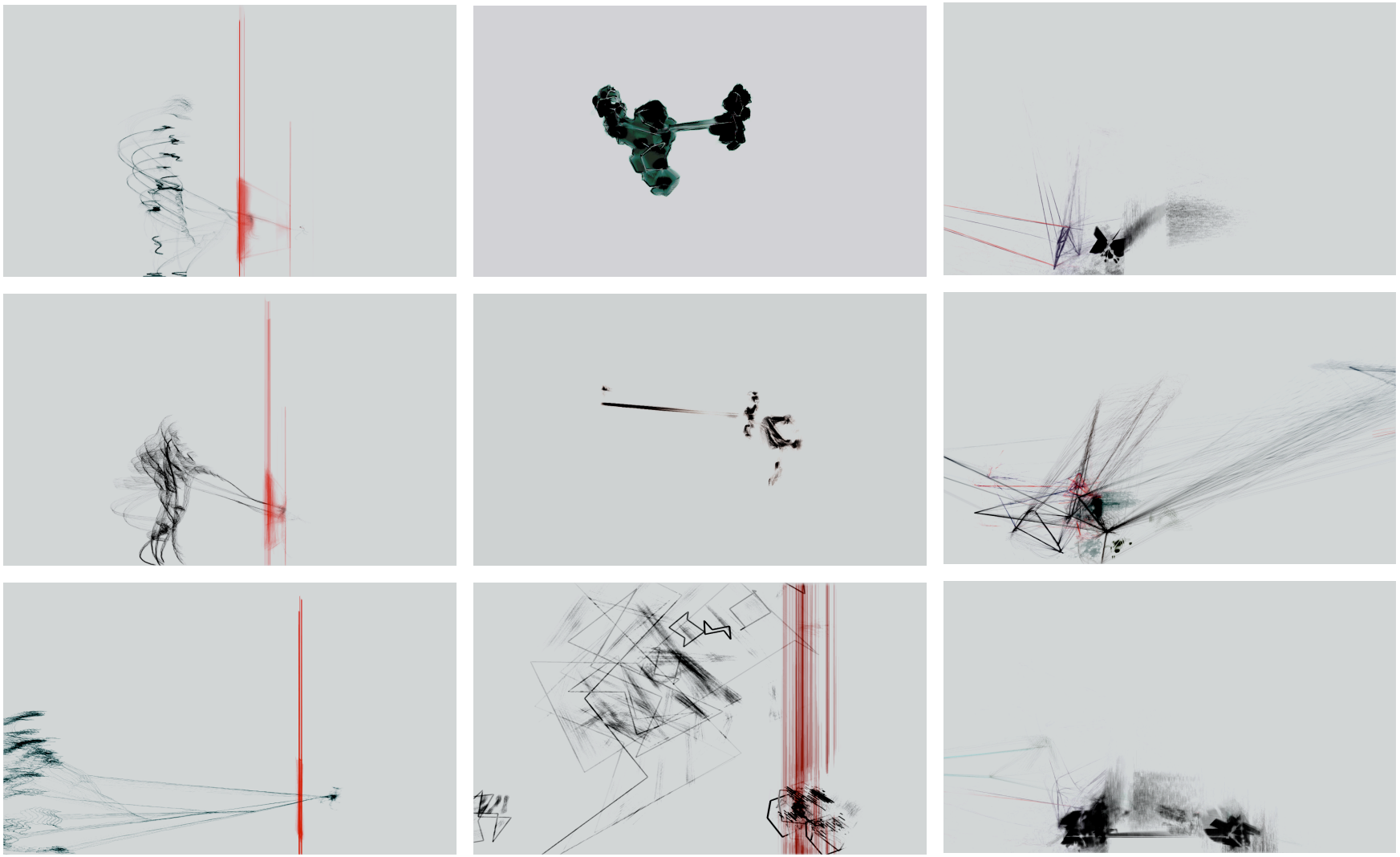


figure 141.
Left: *tree* reconnecting with material from the stage; middle: *stage machine* illustrated two different ways; right: *forest fire* and *stage machine* overlapping. (inverted).

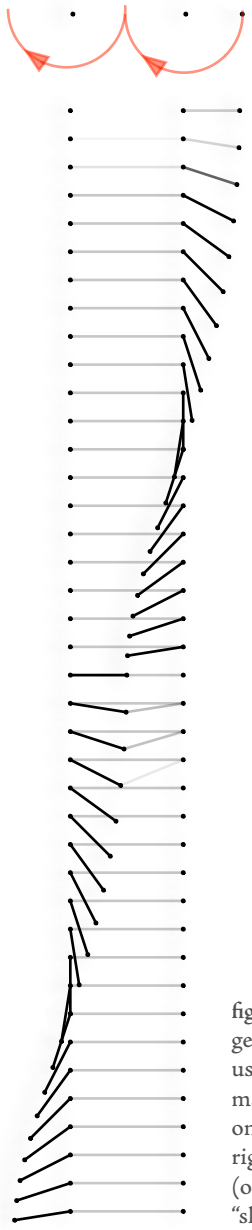


figure 142. A simple automatically generated skeleton process — as used in stage machine. As the markers transition from obeying one constraint (orbiting the rightmost marker) to another (orbiting the leftmost) the "skeleton" follows.

One well-known algorithm for computing the minimum spanning tree of a set of vertices is : **J. B. Kruskal.** *On the shortest spanning subtree of a graph and the traveling salesman problem*. Proceedings of the American Mathematical Society 7(1) 1956.

Traditionally, motion-capture systems, when they are actually asked to label markers, label them against a known skeleton — a set of fixed length bones with fixed markers and fixed classes of joints connecting them. The stage machine agent performs a much more difficult and less grounded task — to try to infer the skeleton from observing the marker data. At the core of the stage machine's algorithm is a matrix of low-pass filtered bone "affinities". That is, a connection between marker m_i and marker m_j :

$$a_{ij,t_0} = \frac{\left|m_{i,t_0} - m_{j,t_o}\right| \cdot \left(\left|m_{i,t_o} - m_{i,t_1}\right| - \left|m_{j,t_o} - m_{j,t_1}\right|\right)}{\left(\left|m_{i,t_o} - m_{i,t_1}\right| + \left|m_{j,t_o} - m_{j,t_1}\right|\right)^2}$$

is considered a good candidate for being a bone if they are both moving and, while moving, they are maintaining their distance relationship. At any given moment we can strike through this matrix a minimum-spanning-tree, a tree that, while connecting every marker, minimizes the (instantaneous) total of the distances of its edges. Within the language of the blendable body framework, this is a process that casts points (markers) and their history of movement into lines (the spanning tree). Inevitably, parts of this tree will reveal themselves as false — points of motion that happened to move in similar directions — two dancers in unison — or points that orbited around another as a center — one dancer around another — (both plausible "bones" by the above metric) will shift direction and break apart. How and when should this skeleton be reconsidered?

Stage machine uses the b-tracker framework to maintain both the underlying marker movement and its hypothesized skeletons. The minimum spanning tree is periodically injected into the bone-level b-tracker as a set of plausible bones. Each tracked hypothesis inside this tracker is a bone — a marker-marker edge — and as these edges evolve we can trace a "skeleton" by drawing the highest scoring edge for each marker. Slowly we drift away from a clean spanning tree of the markers and eventually individual markers themselves might lose all rea-

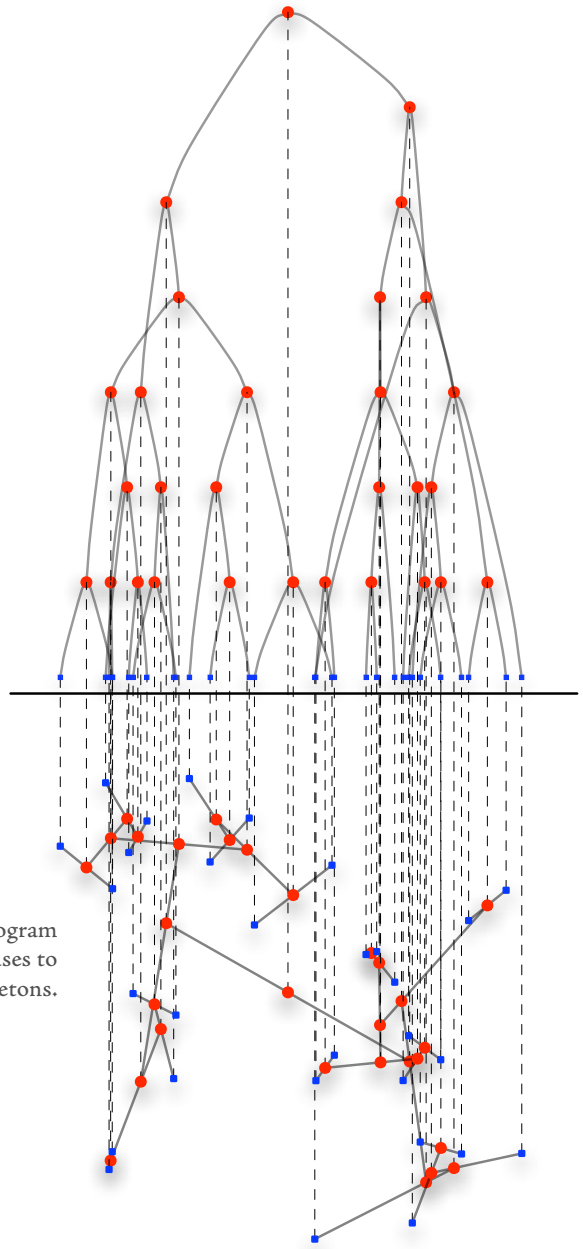


figure 143. The dendogram generator that *Tree* uses to construct its hypothetical skeletons.

sonably scoring bone hypotheses. At this point it is time for the agent to act upon its perception system: to re-inject a new minimum spanning tree into the tracker.

Two twists complete this agent as deployed in *how long...* During the life-cycle of this agent we can modify the above bone metric as it feeds into the minimum spanning-tree algorithm to favor or disfavor bones that span different dancers.

We may go so far as to render cross-dancer skeletal lines differently — indeed at the second appearance of this technique, bones that bridge dancers are used to form a perpendicular, running fence. Controlling the above preference thus controls the density and the detail of this dynamic partitioning of space.

Secondly, we perform the re-injection of the minimum spanning tree, which is often a rather dramatic reconfiguration of the current illustration, using the scheduling techniques of the Diagram system, similar to the intersecting rhythm generators of *parachute / accumulation*.

Tree and *forest fire* are similar to the stage machine in that they are Diagram scheduled reorganizations of a instantaneously hypothetical skeleton; they differ in their definition and rendering of the topologies that they find. *Tree*, rather than finding a minimum spanning tree through the bone matrix, constructs a hierarchical clustering of the markers and their bone matrix. Specifically, *Tree* is a dendogram of the markers with the euclidean distance metric between points.

As a dendogram of N points introduces on the order of another N mid points *tree* thickens the cloud of markers around the dancer. Unlike the use of stage-machine in *how long...* which tracks only the markers of the dancers, *tree* links an offline motion-capture take (with many more points) with what is currently being performed — thus in addition to choosing to reschedule the rehierarchicalization of its perception system, *tree* also interprets its current skeleton as a

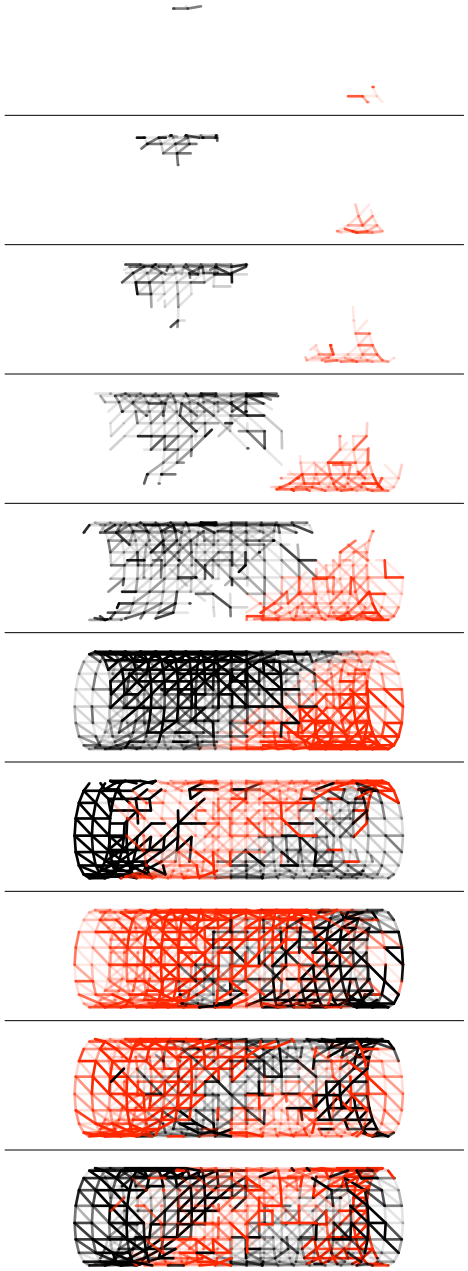


figure 144. The forest fire process,as applied to a static mesh. Two processes (fires), red and black compete for vertices (trees). By altering the propagation probability, the burn time and the regeneration rate a range of dynamics and interchanges can be created.

For an introduction to percolation phenomena: **D. Stauffer , A. Arahony,** *Introduction to Percolation Theory*, Taylor & Francis. 2001.

However, I have particularly enjoyed the presentation in: **H. Peitgen, H. Jurgens, D. Saupe,** *Chaos And Fractals: New Frontiers of Science,* Springer-Verlag, 1992.

filtering network on the underlying motion-capture data.

Each node (bar the leaf nodes) of the dendogram consists of a parent node and exactly two children markers. Typically the position of the parent node corresponds to the center of the two children. Unfiltered, motion of the children markers freely moves the parent; however we can separate the rotational and linear components of the motion of the children with respect to the motion of the parent. By increasingly strictly conserving the distance from each of the children to the parent we can allow the inferred skeleton structure to "push back" onto the data. Rather than fading out as irrelevant and being replaced by a new attempt, the tree can spin out into absurdity and a new tree structure can capture these new markers and bring them back toward the original underlying motion.

Finally, *forest fire* constructs its "skeleton" by a periodic, Diagram-scheduled, Delauny tetrahedralization of the marker set. Rather than displaying this volume of triangles over the dancers, this structure becomes a hidden lattice for a percolation simulation. Specifically, a number of propagable forces or "fires" compete for space on the lattice. At any given moment, for any given occupied node, there is a small probability that this node will succeed in capturing a nearby node. Nodes remain occupied (burning) for a certain time, and nodes remain unoccupiable (burnt, "re-growing") for a similar duration. The trace of propagation for a particular point of origin becomes a cascade of curves over the lattice. The agent acts to replenish the lattice illustration by adding original fires, or, less frequently, to reconstruct the entire lattice.

In different ways, each of these agents construct speculative frameworks for the movement from the stage that they perceive — these frameworks are live, maintained and reconsidered by the agents. It is these agents the provide the most direct slices through the movement and hints of the choreographic proc-

esses behind it — the forging and reforging of similarities (*stage machine*), the propagation of movement impulses from dancer to dancer (*forest fire*), and the making and breaking of clusters on the stage (*tree*). Sometimes, these agents are overlapped with each other, each conflating the present of other agents with the markers of the dancers themselves, providing a dense, accumulative network of computational representation made visual.

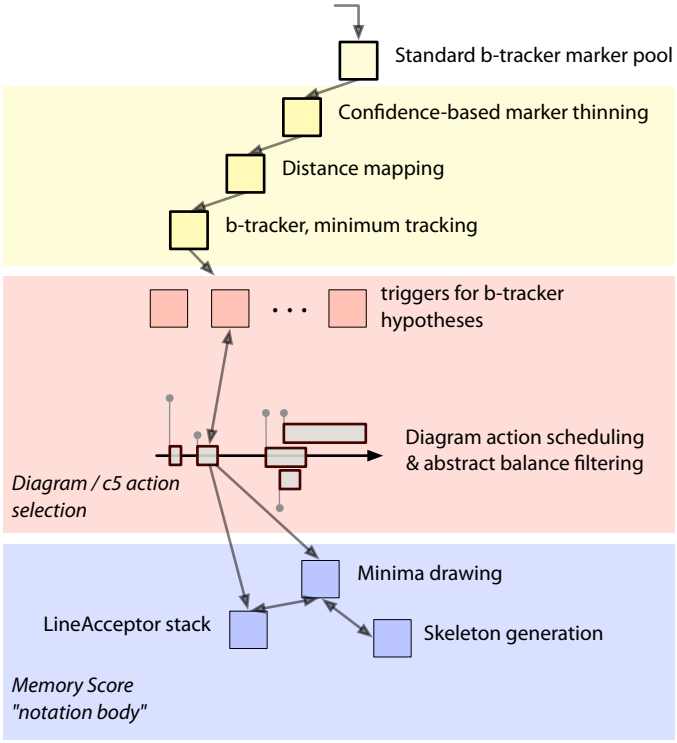


figure 145. The agent system diagram for *memory score.*

The memory score agent was constructed to accompany a tangled, "triangular" solo in the piece. The agent is principally governed by the attempt to visualize a distance-mapping-based decomposition of movement — a comparison and a projection of the current configuration of the dancer's body with the agent's memory of the solo.

The central algorithm is an attempt to decompose the movement into a series of key poses — these will be poses that represent the boundaries of phrases — and then illustrate the connection between the dancer and these poses.

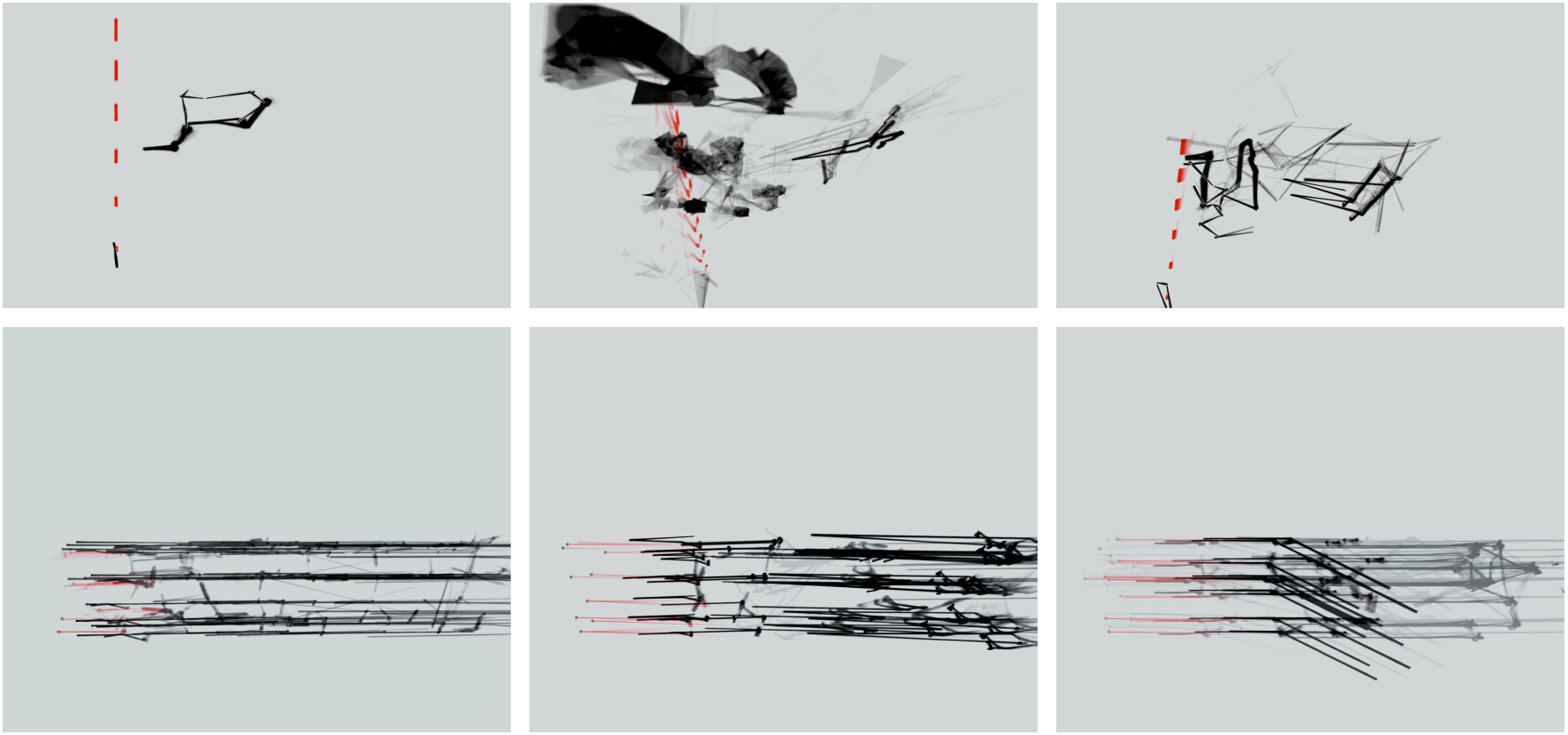


figure 146.
Memory Score, above: taken over the "tangle solo"; below, a more complete marker set. (inverted).

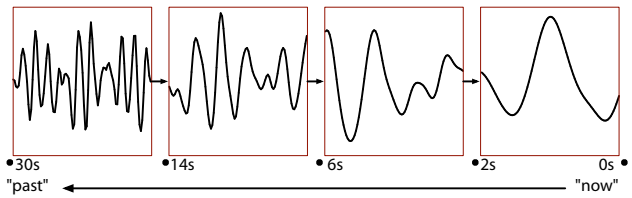


Figure 148. A cascaded, variable resolution signal for the distance mapping algorithm.

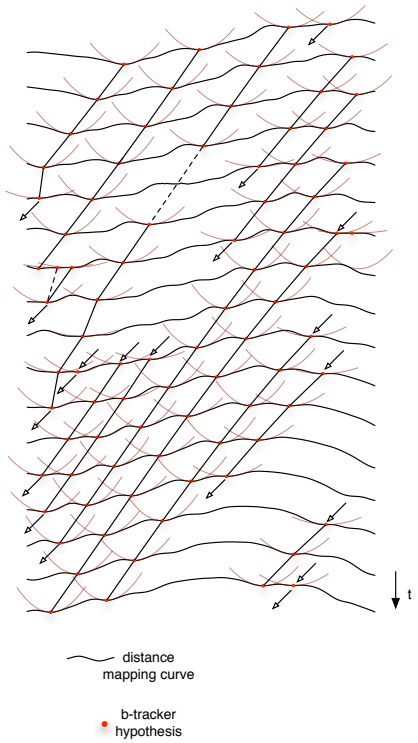


Figure 147. Tracked minima roll backwards through the output of the distance-mapping algrorithm.

First we execute the distance-mapping algorithm to reduce the motion of the dancer down into a single scalar trace. Rather than compose this algorithm over a very long, high-resolution memory of movement, we use a telescoping buffer of time; with 4 time-scales each with half the temporal resolution of the previous one this allows for the representation of 30 seconds of movement at initially 40 frames a second with 200 "frames". Next we look for and track points of local maxima and minima on this trace. For this, of course, we use a simple b-tracker framework: hypotheses are parabolically fitted extrema.

These tracked moments of time, which roll slowly backwards, sometimes splitting apart, sometimes coalescing, correspond to poses that have a special relationship with the structure of the memory — and they have an appealing, intuitive interpretation. They are the points that are locally, maximally or minimally distant from other material. They are the furthest points reached, and they are the points that are unexpectedly returned to after some journey.

The memory score agent acts in response to the creation and deletion of new b-tracker hypotheses, fitting new poses into an illustration of the timeline, projected above the dancer, connecting poses from tracked point to tracked point to form horizontals of time, connecting the structures of poses using minimum-spanning trees to create a vertical. It has a limited amount of information that it can send to its motor system — specified in total connection length and deletions per update cycle — so it is a task that is not without effort.

Visually, the material from the dancer — which at times is quite startlingly clear, ricochets back across the stage as the agent tries and ultimately fails to tether it to the developing score. Although it proceeds and lingers after the solo it ultimately finds a point of correspondence with the, literally, tangled and folded movement of the dancer on the stage.

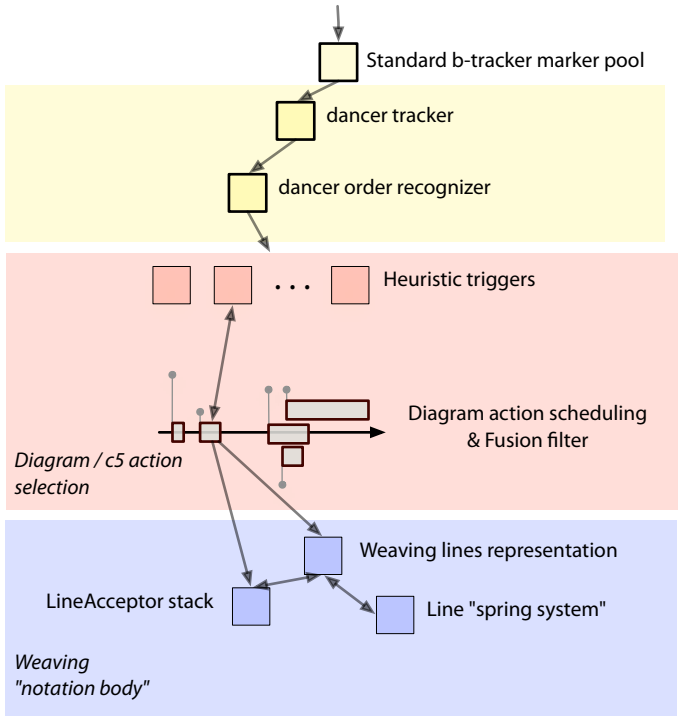


Figure 149. The agent system diagram for *Weaving*.

While *parachute / accumulation* and *memory score* formed and moved their bodies from the traces of dance, and the *tree / forest fire / stage machine* agents grasped at the improbable mechanics behind the dance, the last agent that will be described here hides its body completely.

Weaving, the very simple agent which nearly closes the piece, is constructed from a hidden creature that looks only at the ordering of the dancers, from front to back, and tries to retrace this ordering by weaving a set of lines in space. Specifically, it tries to notate these re-orderings that occur from front to back on these lines by weaving the dancers' respective lines. The agent's motor system is a single Diagram fusion filter that fuses together weaves that occur in quick succession that would ultimately cancel each other out, yielding a simpler weaving pattern. The notation is completed by a perpendicular line that links the event to the dancers in the space, and by a distortion of the lines to overlap with the location of the midpoint between the involved dancers.

As the agent progresses, it moves from fixing its coordinate frame from that of the weaving material to using that of the dancers — the horizontal lines are pulled around by the dancers' increasingly repetitive winding and unwinding, yet at the same time the material is pulled back onto the stage by the crossings and uncrossings of the perceived movement. The initial clarity of the strong horizontal lines when disrupted by this rotational force captured from the stage powerfully disturbs the space occupied by the dancers. As the agent catches up with its perception of the dance, the very stability of the woven lines that it manipulates begins to slip away.

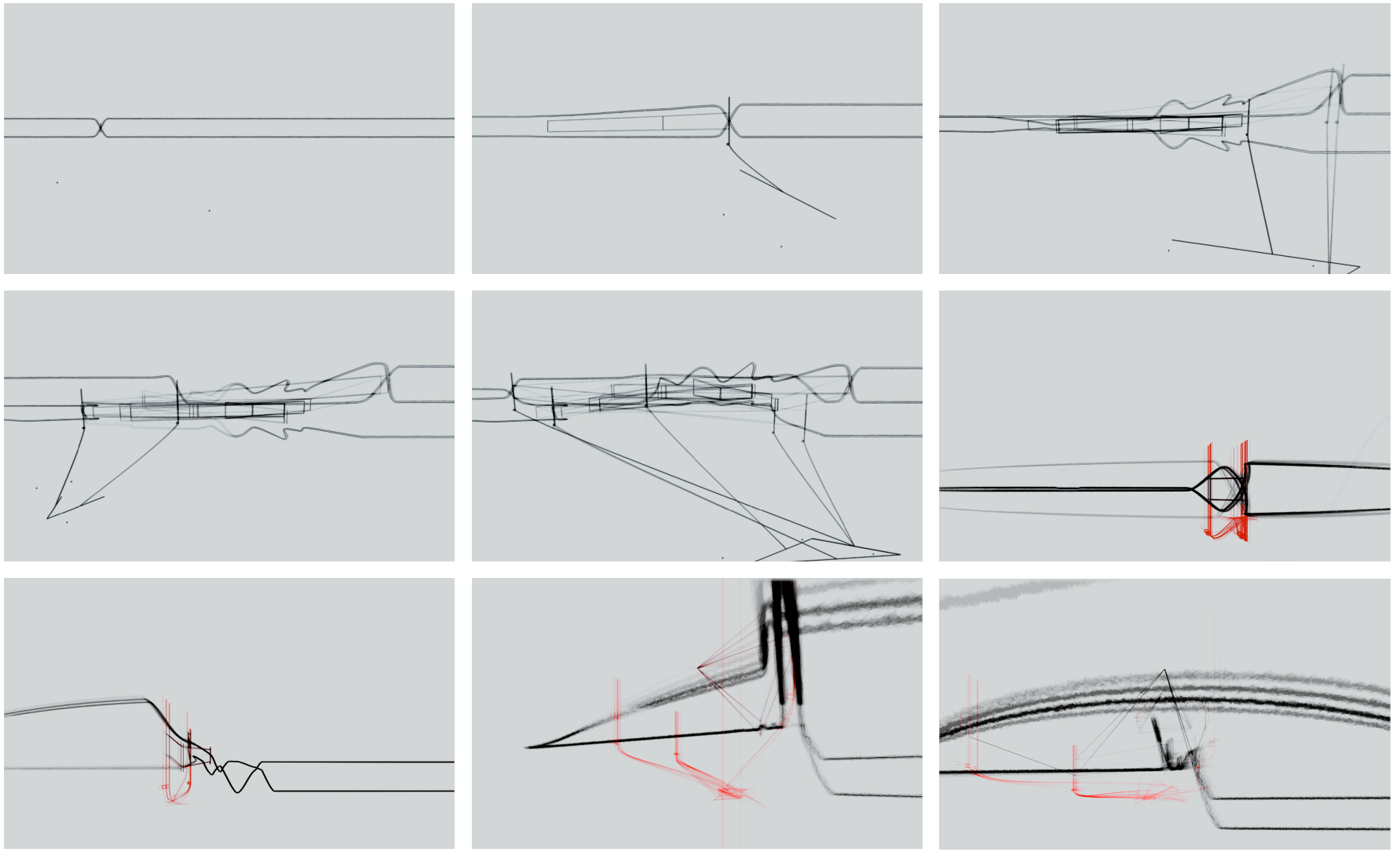


figure 150. *Weaving.* (inverted).

4. Concluding remarks

Both *22* and *how long...* are, in very different ways, a deployment of the agent metaphor to create interactive imagery for a dance theater work. Despite their use of a level of hardware sophistication that offers unprecedented fidelity, they resist the simple manipulation and transformation of this data into images that duplicate the movement of the dancers. This indirect approach might provoke a justifiable fear that by beginning so far away from the source — the movement and the choreography of my collaborators — far away from "visualization", "sonification" or even just careful measurement, I begin on a path that leads only to the disrespectfully cryptic and obscure.

But let us return to how Forsythe, Brown or even Cunningham choreograph. Rather than simply displaying the results a transformation of movement or instruction to move, their working practices accrete transformation, layer and bury the traces of layers on top of one another. This experimentation, this working by working out, is *compositional*, not *instrumental*. It is already at the "opposite end" of the spectrum from mapping, visualization, or rendering-visible.

This distinction, between composition and instrumental echos Robert Rowe's axes of "partner" and "instrument" in interactive computer music performance.

In a traditional "mapping" approach, the search for a few good visualizations of movement data, a fine sonification of motion-capture material, a handful of good-looking points in the space of mappings, is the quest for an instrument — one for Brown's dancers or for Cunningham's movement to play. However, the motion of these choreographers is so dense, and the story of how this motion came to pass so rich, that I understand the urge to take a fragment of motion and study it, pinned under the microscope before it disappears — and this is where the urge to *visualize*, to faithfully *map*, even to *explain* comes from. While *Loops* — an installation work with "offline" motion capture of Cunningham's hands — does not go in this direction, it perhaps might have done.

how long... and *22*, for live, rather than pre-captured, dance, do not do this either. In theater, rather than in installation, one rarely has the opportunity to freeze time, to save motion from disappearing. And to do so, to seek to save the movement from evaporating, is radically against the very nature of the choreography that we are collaborating with. This would not be a collaboration nor even, to my eye, a respectful response to the choreographers' work. Rather we should find architectures for our own algorithms that can twist, fold and intersect in parallel to those of the choreographer, and tools to let us keep pace with the choreographer — open to chance, open to improvisation, open to rehearsal, open to collaboration. That is what I believe my work has achieved.

The aesthetics of the projections for this piece draw directly from generalizing, transforming, representing and computing what is happening on the stage and indicating to the audience that this is already occurring in the theater and has already occurred in the creation of the work. One of my goals is to enable a visual and interpretive mobility for the audience in their reading of the dance, and in their writing of the dance's mechanisms over and above what they normally have in a staging of a dance piece. The space shared by the agents of *how long...* is common not only to the dancers, but to the audience as well; and the imagery in this work here, I believe, rather than mediating the dance *for* the audience, unfolds a *simultaneous* staging of the experience of watching the dance.

One danger is that the projections become authoritative, flattening into a single reading of the play of relationships before they even unfold. The other pole is that the obscurity of the projections erase the connection between the dancers and any dance. These dangers are faced by every piece labeled "interactive", but here the stakes could hardly be higher — one of the most evocative aspects of Brown's recent work is the simultaneous choreography of appearance and oc-

cultation of movement — the unexpected and alarming clarity of what ought to be complex, and the disorienting disappearance of what should be visible.

To avoid these dangers in *how long...* and *22* I sought to build systems that, like the audience, seemed to chase after fragments of movement, fragments of relationships, fragments of non-narrative meaning. And I sought to accomplish this apparent intentionality and this continually deferred presence of choreographic intent by actually constructing systems that truly *would* chase after fragments of movement, while sharing a common space with the stage and possessing their own, related, choreographic formal structures. This work is a sincere and considered response to Forsythe's "architecture of disappearance", to Cunningham's "ephemeral dance", to Brown's perpetually "unstable molecular structures".

This section introduces the graphical environment developed to create, manipulate, script, debug, and visualize my most recent artworks. Chronologically, it comes between The Music Creatures *and the dance works. It is presented in contrast to the predominant tools that digital artists use today.*

Chapter 8 — *Fluid*, an environment for digital art-making

In earlier sections of this document we have described an "authorship stance" with respect to the critique and construction of artificial intelligent agents — a stance where the ease of construction of the agent is critical, a stance where the ability to conceptualize the creation of an artwork, while creating it, inside the agent framework, is vital. It is not enough to demonstrate the academic place and power of learning algorithms, action-selection techniques, and motor-system representations — these innovations must be framed in a relationship with a creative practice.

I have indicated that the authorability afforded by agent systems or AI systems in general receives little direct attention, but we have proceeded to construct a series of technologies that enable the assembly of the kinds of complex structures that AI agents seem to demand. And yet at the same time as the context-tree, the Diagram framework, the generic radial-basis channel and the use of historical databases of various kinds enrich the vocabulary for expressing agents, they in turn make their own complex potentials. In this section the authorship perspective on agent systems meets a more sustained critique of the authorship perspective afforded by "mapping" and its kin as we construct the toolset used for navigating our agent-based practices and the agent toolkit.

Further, in the introduction, *page 28*, we have set up the concepts surrounding the blanket term "mapping" as an opposing model against which to pitch an agent perspective, as both metaphor and implementation. I have already noted that this term is a pervasive one, but nowhere is its pervasiveness more manifest than in the tools that digital artists use.

But there is perhaps more at stake here than the critique of existing environments for multi-media interactive art. The construction of tools for creating, treating and manipulating the complexities of an agent toolkit — tools that transform this technology into one that can be rehearsed with, speculated with and collaborated with — is a problem that broadens out to more general issues in digital design; issues that are independent in many ways of the particulars and the eccentricities of the agent-based . While I shall focus on the comparisons between my tools and those dominant in interactive art, I hope more general contributions will become apparent.

1. A critique of existing environments

Some of my works, in particular the earlier agents — *Music Creatures* and *Loops* — exploit and motivate these AI techniques and metaphors, but they remain artworks that are created through the writing and testing and tuning of tens of thousands of lines of code. Some lines of course are part of the toolkit and they have been written and tested before; some of the new lines are refactored to become part of the next toolkit. However, a great many of them were specific to the particular installation. and the making of the art-work is the making and remaking of those lines of code until they are right.

Not many digital artworks are made like this today. Rather there is a burgeoning community forming around a growing set of software tools for making digital art, in particular interactive digital art without "recourse" to text-level

programming. And, since these tools are well entrenched in the community and form the basis of the courses and programs in schools, it is likely that very few digital artworks will be made like this tomorrow either.

With few exceptions, these popular graphical environments (they are controversially sometimes referred to as visual programming languages) are based on a common small reservoir of ideas: a few visual metaphors, a few structuring concepts. They each possess a surprisingly similar flavor and set of capabilities. So similar, in fact, that one might suspect that we are suffering from a digital art tools monoculture.

My recent works — *how long...* and *22* — were both created over a long, sustained period, their premieres scheduled two and a half years in advance, each given five, week-long workshops spread evenly out over that time, with a relatively stable set of hardware. Further, these works were made in collaboration with other artists, both visual and in other media, some of whom programmed while others did not. This rare time-frame of sustained collaboration allowed a long and profitable look at the tools needed to *survive* these intensive workshop scenarios, effectively both allowing and necessitating a move away from a technique based completely on writing, testing and tuning those thousands of lines of code. Under ideal conditions one might argue that such a sincere look at not just "what should be done" but the inseparable "how it should be achieved" is part of the responsibility that artists have to their collaborators upon agreeing to work together. To construct and own one's tools as far as possible, marks nothing less than an openness to the potential of the collaboration. The resulting goal was to find a different reservoir of ideas that could be drawn upon for the creation of a fresh programming environment for the making of interactive digital artworks. This would be an environment for which I would be responsible — not (just) for the maintenance of the engineering, but for the supply and ultimate flux of embodied concepts.

This section will begin with a brief survey of the common principles behind the graphical environments. This will not be a critique of their implementation details, their stability, or their processing power, but rather of what it is they set out to do and the affordances they offer to artists who come to them.

I shall articulate three main weaknesses of these environments, before moving onto a more sustained, contrasting description of the graphical environment that emerged out of the needs and pressures of authoring the agents for *how long...* and *22* and the end of *The Music Creatures*. My discussion will summarily ignore the recent interest in pure programming environments such as the notable Processing, Drawing By Numbers and Alice applications. The former compromises slightly Java's support for maintainably complex projects in exchange for a significant and admirable gain in pedagogical impact, but remains thoroughly and deliberately eclipsed by more fully fledged programming environments. From a community perspective Processing is extremely vibrant and interesting, but as a development platform it is only half-way toward something else. The other text-based programming environments aim, in different ways, for even greater pedagogical impact and an even greater simplicity constraint.

The mainstream tools have been criticized before, however my purpose here is a little more focused. Nor is this discussion the place for a fruitless competition between the agent-based and extant tools. Rather, we are looking for environments that allow us not just to create complexity but to interact, navigate, manage and collaborate around the kind of complexities that the agent-based approach tends to create. While they may be sold (and even taught) on the basis of how rapidly they create potential, what I will ask of these tools here is how they interact with, navigate and manage the potential of interactive media.

Begun by **Ben Fry** and **Casey Reas**: http://www.processing.org . Its tactical simplicity when compared to other Java environments was declared in a personal discussion between myself and Reas.

Alice: http://www.alice.org/

Drawing by Numbers is by John Maeda, **J. Maeda**, *Drawing By Numbers*, MIT Press, 2001.

The most productive critical dialogue around Max— conducted by a number of major figures in computer must — happened a considerable time ago (to apparently little effect): It is collected in

P. Desain and **H. Honing**. *Letter to the editor: the mins of Max*. Computer Music Journal, 17(2). 1993.

The graphical suite with longest pedigree is Max/Msp/Jitter — with Max being the name of the core and Msp and Jitter being progressively more recent extensions that allow the manipulation of sound and video respectively. More than anything else Max is the canonical data-flow programming environment for interactive digital art. The central metaphor is that the flow of data between processing modules will be represented as a visual circuit — this is a digital implementation of the wires of an analogue synthesizer. The computational strength of the environment is then measured solely in the number of available modules and perhaps the number of data-types that these wires can carry.

Circuits can be hidden inside custom modules and while a few modules present custom views and interface elements onto their inner workings, most, including embedded circuits, retain a rather generic appearance — a label and input and output terminals. This itself is not a particularly problematic design decision — an attempt perhaps to maintain a rather clean and minimal visual appearance to a complex circuit.

But visual programming is an idea that seems always to be sliced in two, and Max partitions the visual and the programming at a very particular place. What is visual is precisely that which is not programming and what is programming is, I argue, not made especially graphic. The actual layout, appearance, size and visual relationships between these modules are meaningless. This has the stated benefit that users are relatively free to reorganize the visual appearance of the circuit to create their own "interface" to the patch. In practice this flexibility is greatly curtained by the circuit's metaphorical use of wires which do act to constrain layout and worse: the primitive interface possibilities of the completely static layout of a circuit. Is any other complex software product content to display an interface that does not change structurally? Rather, the static panel of knobs and switches is again borrowed from the analogue synthesizer. But if one argues that Max is neither a interface language nor a programming language, it

Information about Max/MSP/Jitter can be found at http://www.cycling74.com

Other environments in this tradition:

Meso's *vvvv* —
http://vvvv.meso.net/

Infomus Lab's *eyes-web* —
http://www.infomus.dist.unige.it/eywindex.html

Trokia Ranch Dance Company's *Isadora* —
http://www.troikatronix.com/isadora.html

Miller Pukette's *pd* —
http://www-crca.ucsd.edu/~msp/software.html

IRCAM's *j-max* —
http://freesoftware.ircam.fr/

There is one exception to the meaninglessness of the visual layout of a Max "patch" or circuit — that the top-to-bottom, left-to-right ordering of elements breaks ties in deciding the execution ordering of modules. But it is generally believed that if a patch depends on this subtle execution ordering the patch ought to be redesigned.

remains to be seen if there is a better way of slicing the problems presented by "visual programming".

Equally ambiguous but more important is Max's very assumption that a depiction of the flow of data through modules that process the data is a particularly good way of capturing what a program does, that the manipulation of the flow of data through modules is a particularly good way to change what a program does and that thinking about the flow of data is a good way to think about what programs do and should do.

It is hard to find any persuasive science either way on these questions — few care about the speed with which artists can assemble their programs and even fewer would try to measure the quality of the decision making under such constraints — although there are some researchers who measure the behavior of programmers in similar environments (the popular LabVIEW environment which is targeted at engineers, but has been used for interactive artworks). In any case, the literature is utterly inconclusive about the merits of data-flow versus opposite paradigms, most notably "control-flow", where visual elements represent the looping and gating constructs of imperative programming languages rather than the inputs and outputs of procedure calls. There are shades of this alternative presentation buried inside Macromedia's Director and applications that date from its era. It is telling that environments based on this depiction have been more popular in two areas: scripting languages and programming pedagogy than they have in interactive art per se. There seems to be something of relevance to the "temporal arts" that the pure data-flow path misses.

A survey of our incomplete knowledge concerning the efficacy of visual programming environments can be found in: **A. F. Blackwell, K. N. Whitley, J. Good, M. Petre,** *Cognitive Factors in Programming with Diagrams.* Artificial Intelligence Review 15: 95-114, 2001.

For a use of LabVIEW in interactive music: **T. Marrin-Nakra,** *Inside the Conductors Jacket: Analysis, Intepretation, and Musical Synthesis of Expressive Gesture.* PhD Thesis. MIT, 2000.

Macromedia — http://www.macromedia.com/

For control-flow-based visual "programming" take, for example, Apple's wrapper around the text based AppleScript — Automator: http://www.apple.com/automator.

The visual interface for the Alice environment is also control / object first: **S. Cooper, W. Dann, R. Pausch** *Teaching Objects-first in Introductory Computer Science*, Proceedings of SIGCSE 2003.

It is perhaps for this reason that PD, the next generation of Max-like environments, possesses a nascent "data-type" system.

Data-*flow* also trumps data-*type* in these environments. Max's wires can move numbers, sound and video around in addition to nested lists of numbers and strings — in theory, a circuit can talk about any data "structure" that, say, LISP can. Yet at the same time the use and inspection of these non-uniform data-structures are utterly un-visual and un-composable. Nowhere is this more apparent than in the handling of geometric scene data, which necessarily are complex hierarchical linked systems. For all of my earlier discussion of the controllability of geometry versus the blendablility of video texture, geometry in these applications — with its messy, heterogeneous, hierarchical, typed, non-flowing data-structures — is less controllable (and, of course much less blendable) than video. Geometry in these applications is fixed and solid, a container for texture; it is something that is imported and displayed rather than synthesized. This lack of interest either in variable data-structures or variable control-structures is clearly antithetical to the needs of my work, since much of the technical contributions of this thesis has been given over to the task of making complex systems that change structurally while running. I believe that a toolset and a methodology that draws one towards such "static" complexity actually draws one away from the potentials of interactivity — be this between artist and tool, dancer and stage, or audience and screen.

That Max, and its progeny (including *PD*, a re-implementation by Max's main original author Miller Pluckett with different license restrictions, operating systems and, of course, modules; and *vvvv*, a re-implemention of the same ideas with, at the time, a different operating system and a greater emphasis on video), should focus on illustrating data-flow rather than control should come as no surprise. And we can use this as circumstantial evidence in the absence of any applicable visual programming language science. The Max module is the most succinct "visualization" of the mapping metaphor that one could imagine, short of our earlier "function" image. As far as it is visually concerned, in the language of computer science, the module is no more than a function call. Indeed if there

is a common computer-science reference to data-flow programming environments it is not "object orientation" as has been claimed but rather a simple side-effect-free functional programming language. Of course, Max cannot extend this principle too far, and ultimately compromises its functional "purity" — hidden, un-visualized side effects abound.

It need not be this way. Max's modules nest but never intersect, they are not views onto a complex system, but the complex system itself. My point of departure is a slightly different place, I require tools for manipulating the agent-toolkit, offering, that is, windows into systems rather than the material of the systems themselves.

In the language of user-interface design, however, this is not a matter subject to taste. Rather Max conflates the *model* (the data, here both the modules and the wires) with the *view* (the way of manipulating the data, here both the box and the lines) and with the *controller* (the glue that binds the model to the view). Such a conflation is considered unforgivable by many a human user interface programmer. It couples the model so tightly to the view such that no other view can be offered onto the model. This visual monoculture is our first main criticism of Max and can be levied regardless of what one thinks of the power of the contents of its boxes and wires.

For a detailed definition of the Model-View-Controller pattern: **F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal,** *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns.*John Wiley & Sons, 1996.

For the visual-programming literature does have consensus on one topic — the vast number of different visual metaphors available to choose from. There are hundreds of visual programming languages. Max offers one metaphor, but more critically, enforces this single view onto the "program". Indeed, its view onto the program is, as far as it is concerned, the program itself.

See, for a range, the two-volume review of the field up to 1990: **E. P. Glinert,** *Visual Programming Environments: Applications and Issues & Paradigms and Systems*. IEEE Computer Society Press, 1990.

One can extend Max by adding a module type — either through a external, compiled textual programming language, or though the nesting mechanism —

but one cannot add a new way of looking at the "program" that Max has helped assemble either from outside or, critically, inside Max. This lack of self-reflexivity is the second of our main criticisms.

Of course, the exact same charge can be levied against a textual programming language — few source codes are open or reflexive in this sense. Although we'll note in passing a recent interest in doing just this — XML based programming languages such as *Water*, the Inversion of Control XML configuration files of several container systems and of course this less-than-recent aspect of LISP, seek to blur program and data by programming with a structure designed for data. The goal here is to allow programs to view and remake programs, in much the same way as we asked previously if Max should not allow modules to make, move and delete modules, even if just for the sake of having dynamic interfaces. Some have gone so far as to predict the slow death of "single-text" programming languages as they become inherently multi-perspective. If textural programming is becoming self-reflective and in an odd way "multi-media" shouldn't there be a panoply of domain-specific, multi-media programming environments leading the charge?

Water, C.Fry and M. Plusch — http://www.waterlanguage.org/

This trend comes from plotting a line through the interest in extensible syntaxes for example, the Fortress languge — http://research.sun.com/projects/plrg/fortress0618.pdf

and **G. L. Steele, Jr.**, *Growing a Language*, Journal of Higher-Order and Symbolic Computation 12(3) 1999.

and trends towards direct manipulation of abstract syntax trees, including James Gosling's Jackpot project: http://today.java.net/jag/page15.html

Isadora —
http://www.troikatronix.com

If Max has its technical roots in the analogue synthesizer and its conceptual roots in mapping, few environments can be seen to widen these bases a little. *Isadora* is an interesting environment for the purposes of this thesis because of its development in an interactive dance context — it is the work of the Troika Ranch Dance company, artist and engineer Mark Coniglio. Its more accessible revisitation of the visual design of Max is noble, but not an issue for this discussion any more than its processing speed or range of modules.

More interesting are the two concepts that Isadora, depending on one's view, either adds to Max or pulls out of Max and names: that of the specific recoverable graph configuration or "scene", and that of a separate control surface for a circuit or "control panel". The "scenes" are the most interesting innovation — they offer specific support for a clumsily created control structure implemented in a master "graph" in Max that switches between activating various subgraphs.

An Isadora document can have any number of Scenes, each of which is a collection of actors (modules) that manipulates one or more streams of digital media. Isadora scenes are

Isadora (v1.1, pre-release)
manual, p. 68 —
http://www.troikatronix.com

> like scenes in a play: each one may have a different set, different lighting, etc. [...] Because you can jump almost instantly from one scene to another [...] it is possible to move from one interactive setup to another as you move through sections of a performance.

In a sense they are environment support for episodic pieces. The metaphor given is one of lighting or stage cues but while this is useful for understanding what they are, it is just as useful for discovering what they are not. For in lighting and stage cues there are a well-defined set of resources for a scene change to act on. This enables the idea of a transition to be defined as resources — light levels, ropes etc — are moved from one state to another. Not so in digital media progressing networks. A scene change, as given by Isadora or by Max's limited control flow, a necessarily dramatic event — it involves the initialization and

configuration of one circuit and the termination of processing in another. No amount of support, which Isadora has, external to the patch, for fading in or out the video output of a circuit really meets the challenge of a live multi-media "transition". A cross-fade of end-products does not allow the outgoing modules that control that video output to negotiate their relationship with the modules that will soon replace them. Even the simplest L-cut of film-editing — where a cut in one medium precedes a cut in another — violates the constraints of this clean cross-fade.

This technique depends on the superimposability, and we might rather say, texturality, of the predominantly video-like media that flows through those circuits during this switch over or "cross-fade". This is in contrast to a tool that would acknowledge the geometricality of the processing graphs that are being juggled by this scene switch.

The ultimate inadequacy of Isadora's "scene" leads us to our third main criticism of the whole data-flow paradigm: while Max and others may organize the flow of data around efficiently and somewhat visually, and their control structures, while questionable, are clearly serviceable, their relationship with *time* is particularly weak. To my taste, to be able to create and manipulate complex temporal flows is a more central and harder problem than the creation of complex flows of data. Superiority in the domain of creating complex data-manipulations are something that visual environments such as Max can battle over with programming languages like Java, but the layering and negotiation of temporal structures is something that both Java and Max do unquestionably poorly.

The perception / action / motor system decomposition of an AI agent is as much about the layering and negotiation of time than it is about anything else, and we have already seen (*page 242* and *page156*) a number of additions to our

core programming languages that extend their vocabulary in this regard over traditional imperative languages. Perhaps these victories for a text-based environment can be secured if they are placed in a visual framework that starts by depicting the flow of time rather than the flow of data.

2. *Fluid*, an overview

Fluid, while it might compete in the same arena, has to solve a different problem. While it shares with Max and Isadora the goal of being a working environment for artists — the tool that they use live *in* rehearsal, the sketchpad that they use on the plane *to* the rehearsal and the environment that they develop ideas and materials in long *before* the rehearsal — unlike Max it doesn't have to take responsibility for creating all of the complexity of the piece. Rather, Fluid's job is to make the agent toolkit approachable, improvisatory, extensible and developable, to cull from its potential the pieces of an artwork. Thus, it is necessarily hybrid, sharing the development space with the environment used to make the toolkit itself. This can be both a feature and a drawback: it is a feature because the more conventional texual programming environments have had far more development time spent on them than any tool in the arts probably ever will, and a drawback because the Fluid system will never have a complete view of the development of artwork.

A summary of some design principles behind Fluid will help locate Fluid and define its relationship to both traditional pure "code practice" and the traditional use of environments like Max.

Visible, editable code is a ubiquitous glue in every visual element — thus, every visual *element* on a Fluid *sheet* contains code — be it a simple box, a time marker, a graph — and this code is inspectable and changeable and, of course, executable. This means that while Fluid is most certainly a vis-

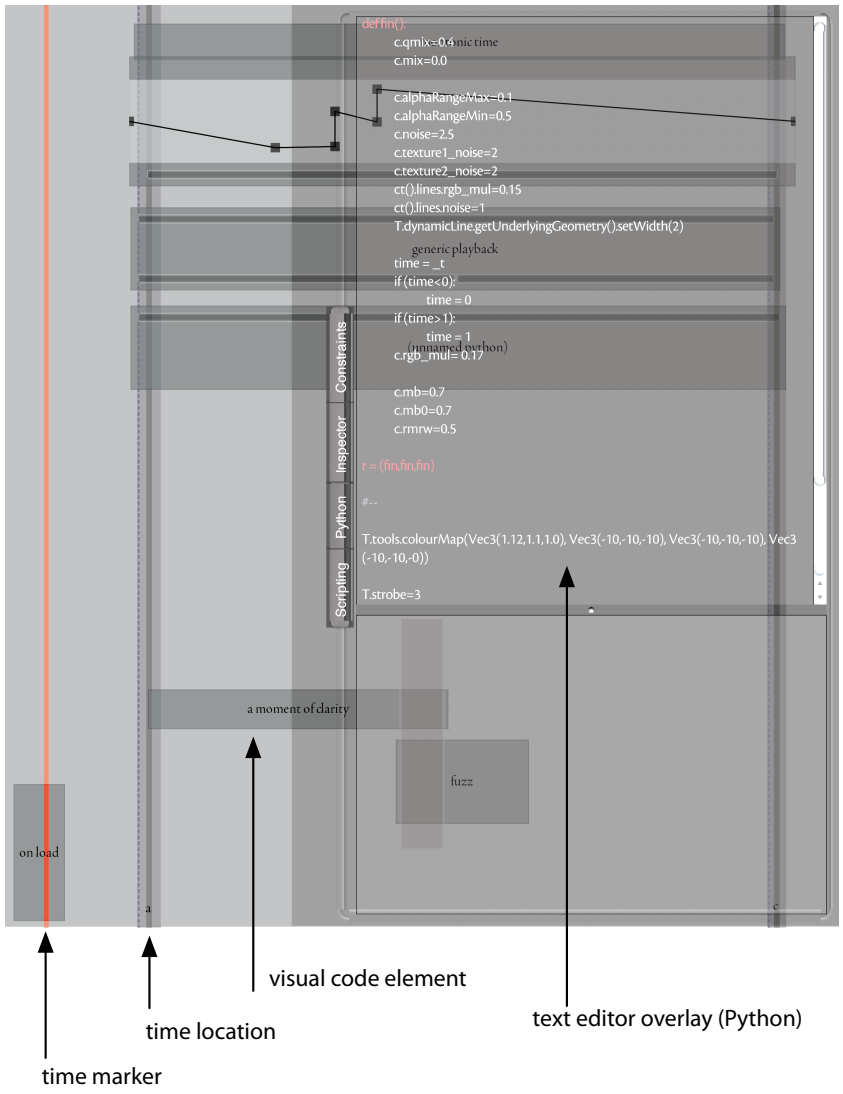


figure 151. The main window for a sheet — visible here are some code elements of various kinds, and the text editor and the output panel.

ual programming *environment*, it is most certainly not a visual programming *language*. The atoms of programming are readily available and editable but they are not necessarily graphic.

Every visualization is "executable" — a Fluid sheet is a place where one makes fragments of code that call upon the whole agent toolkit, however, it's also a place where the whole agent toolkit can deposit visualizations of its state. But, here, the same principles apply — editable code surrounds and connects the visualized elements to the sheet and to each other.

Visual presentation matters (somehow) — the visual arrangement of sheet elements is typically interpreted by other sheet elements and is always meaningful for some process. Therefore Fluid invests a considerable amount of code towards making general-purpose spatial manipulations available: multiple groupings of objects are possible, two constraint-based layout systems are available, code can talk about spatial filtering. But, the visual presentation's meaning is open to definition and redefinition — it might be that there is a flow of scripting type from left to right, or it might be that child elements above other elements are responsible for the children's life-cycles. The visual presentation's interpretation is not set, but there are enough tools for controlling the layout of elements that it can be made usefully important.

One visual element may be in a number of "places" — since visual position is meaningful in a variety of ways, visual position is no longer completely available to the programmer to act as a "secondary notation" for organizing thought and storing memory. To restore some of this flexibility we allow and expect objects to be able to exist on a number of sheets simultaneously, including sheets that may not be currently loaded. Often multiple sheets are stacked in layers showing visualizations of lower layers' contents and workings or showing the relationships between sheets.

The history of using the tool is in the tool itself — programmers have typically surrounded their editors with extra versioning systems that keep track of how textual code is changing from day to day. These tools are almost always domain-agnostic, handling text files with no knowledge of their contents. However, in collaborative art-making the history of the collaboration is part of the collaboration, and environments should make the history of their use directly part of the environment. We have seen this need in our analysis of *alphaWolf*, *page 100*, and I have seen it in my own work, *pages 114*, *143* and *225*. This necessitates the creation of domain-specific versioning systems and domain-specific loggings and analyses of how the tool is being used.

The environment can refer to itself — Code in Fluid is open to manipulation by code (in Fluid). Following on from the previous principle, the code inside each sheet element can talk about, manipulate and script the appearance of itself and connect to an interface that allows the manipulation of other components in the sheet. Additionally, the format for storing Fluid sheets is a both human- and machine-readable XML. This principle is also manifest in a certain self-reflexivity of interface. What would be a fixed set of menus, slider or other user-interface "widgets" in a traditional environment become code "boxes" that happen to have particular appearances. The boundary between a finished "user interface" design for the tool and the tool itself is removed; part of the action of working within Fluid is to reorganize the interface itself.

Fluid sheets can be instantiated with or without visual display — A sheet can be instantiated without creating any visual component, in the complete absence of the underlying windowing system. This is an apparent feature of almost any graphical programming environment; however it is harder to maintain in the presence of embedded code that might end up manipulating the "appearance" of a non-graphically instantiated sheet ele-

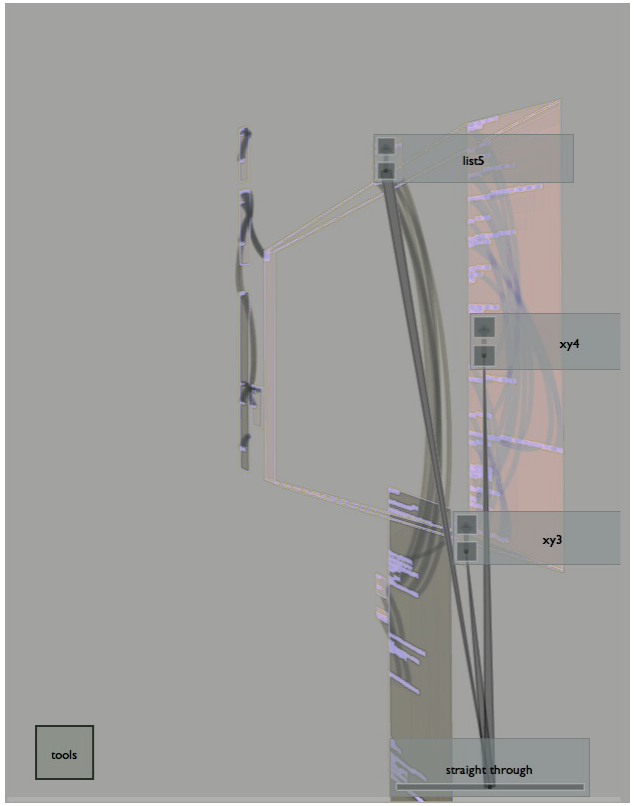


figure 152. The boundary between finished artwork and development tool is blurred. This image is of the Diagram visualizer, which can underlie the *Fluid* framework. The visualizer itself showed along side *Loops* & *Loops Score*.

ment. To fully exploit the power of *ad hoc* visual layouts for all kinds of tasks both big and small, we should expect hundreds or thousands of sheets to be instantiated during the life cycle of the work. Any hint of the underlying windowing system in such a process prevents this use-case from being either practical or stable.

The boundary between finished artwork and environment is blurred — no clean separation between what is "Fluid", what is "toolkit", and what is finished artwork has been maintained in my practice. The movement of ideas has not always gone from toolkit to environment and the flow of control isn't always from environment to the toolkit. Some examples: the layout constraint system become the motivation for some of the advanced generic radial-basis channel combinations; Fluid can overlay the main graphics canvas and track objects on the screen; there is a pose-graph motor system representation for fluid visual element positioning — one can think of Fluid "agents"; parts of Fluid have even been exhibited alongside *Loops*. The visual element structure can quickly become just the visualization of a Diagram marker channel and vice versa.

Together, these principles imply a visual programming environment that is a radical break from the Max tradition, or indeed from any art-tool relationship widespread today.

This environment was tailored for a specific domain and a specific working style: the creation of collaborative interactive artworks through intensive workshops, with generous but expensive time in theaters, through improvisation and through the condensation of improvisations. It dodges completely several problems that others confront and, I believe, fail to convincingly solve. Unlike the Max / Isadora / vvvv / EyesWeb tradition it does not attempt to be a visual programming language — it limits its use of graphical display and makes extensive use of editable code. This places in the Processing / Drawing by Numbers /

Alice camp. Yet at the same time, this allows it to be in some cases more visual — the visual layout of elements can be made meaningful to and visible to the program itself. However, unlike these extant "art languages" Fluid sheds any pedagogical claims in favor of offering an environment that retains the power of "full" programming languages to scale to large, multi-hundred-thousand-line code-bases. Yet it retains and exploits a commonly available programming language that, while it is extended, does not break its connection to all of the literature for the language, or the research behind the language.

Without extensive and difficult-to-control user studies, Fluid can make few claims for ease or power of use. This makes it no worse off than the dominant "products" in the marketplace today. which operate without a firm predictive or explanatory theory of their use. At the very least Fluid stands as a unique set of wholly implemented, and demonstrated hypotheses about what features art-making environments need to possess to survive long collaborations around open-ended and complex systems.

The principles above form the backbone of my description of Fluid.

The canonical definition of the Java language is **J. Gosling, B. Joy, G. L. Steele Jr., G. Bracha,** *The Java Language Specification, 3rd edition*, Addison-Wesley, 2005.

for Python, see http://www.python.org

for Jython, see http://www.jython.org

The code that is ubiquitous in Fluid is a "dynamic language" called Python, specifically the Python implementation written in Java known as "Jython". In contrast to the main language of the agent toolkit (Java) Jython is an interpreted, late binding language, the semantics of which are extremely malleable, containing a full and rather usable meta-class programming framework. It has been designed with the purpose of being hosted from within a larger system in mind, and Jython has been designed to be hosted within a Java runtime. Finally, as a language that has a compact and open source implementation, the interpreter is not a closed black box, but rather an ideal site for further introspection and augmentation. The basic strategy is as follows: by using language such as Java for the large and intersecting agent toolkit, and a dynamic language such as Python for the assembly, glue and interface to the toolkit, the strengths of both programming languages can be combined.

Because of the modifiable semantics of the language we can build carefully prepared classes and environments — for small amounts of Python code to use — and glue visual elements that contain these codes together, giving them purpose on the sheet. Obtaining an editor for these pieces of code hidden inside elements is easy — Fluid presents a typical, and rather complete, code editor that happens to support rich-text-format files and runtime automatic line completion. Objects in Python can be inspected live, and code executed per-line, per-selection or per-element with a single key-press. Quite a bit of time, especially in the early stages of development or testing, is spent purely inside the editor window executing code and inspecting objects. Useful code is then propagated outside the editor into separate visual elements for execution or even just documentation of ideas, examples and tests. At this point we have a system that can support simple spatial mnemonics — the equivalent of a "Desktop" metaphor for small snippets of code.

This is one route into a visually "extended" programming, but to go further, the positions of these elements must begin to mean something. The simplest example of this is a Fluid timeline. This is a commonly used way of configuring a sheet and is a good starting place from which to build an improvisation environment.

Inside a time-line there is a series of executable elements and one or more time markers. Broadly put, a time marker executes elements that it crosses. Of course, this means that there is a *life-cycle* for the executable elements — at some point in time they transition from lying dormant to being "executed" each cycle (at various times) and then later from executing to being "stopped". We know from the rest of this work that it is important to state the contract between the executor and the executee in such life-cycles; the life-cycle for these elements is detailed, open and strongly enforced.

Runners and execution

Helping the time-marker is a Runner class, that maintains this contract, actually executes the code inside each visual element, and interprets that code's "return values" or effects on the code's local environment. Although time-lines are ubiquitous, especially inside computer music systems (one could think of any sequencing package or audio editor in the last 20 years), the presence of executable code rather than musical notes or sounds inside the elements adds a complicating dimension.

The possible state diagram for visual elements executed by a moving line is more complex than it first appears. Since the timeline runner executes code by intersecting visual elements' bounding boxes with the rectangle formed by the sweeping time marker over one execution cycle, we should look at how these rectangles can intersect.

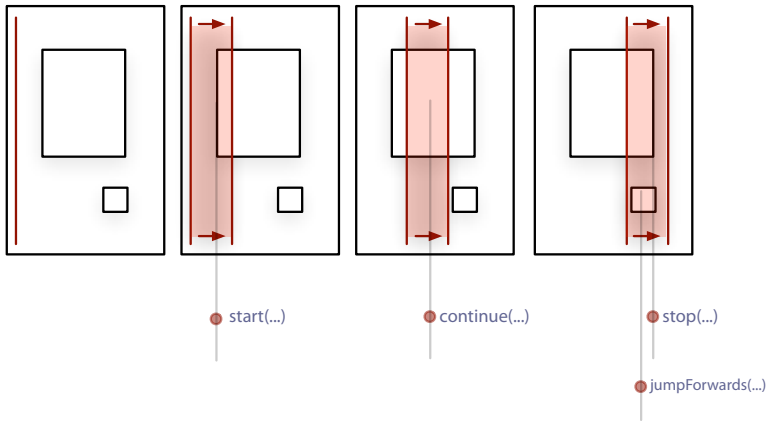


figure 153. The intersection of a moving time-marker and visual elements causes a number of messages to be sent to the elements.

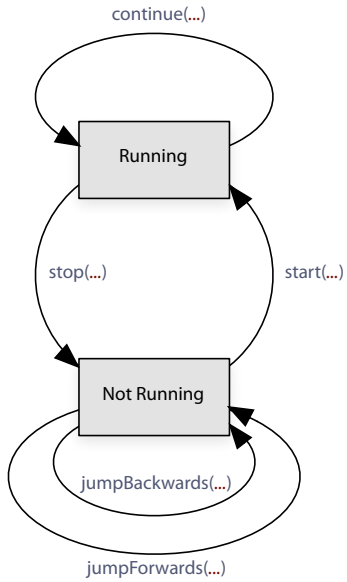


figure 154. The full state and transition diagram for a visual element.

The traveling time-marker can cut across the start or the end of a visual element but it might also wholly consume the visual element (effectively starting it and stopping it in one cycle); additionally it may do this while moving backwards. It is important to allow the code inside the element to respond to each of these events differently if needs be. Many visual elements run only once (on startup), some do the same thing at start, continuation and ending, some are "unmissable" (the proper execution of subsequent elements depends on this element having started) and execute at least their start and end on being jumped over, others are not worth starting unless they are going to continue for a while.

It is the Runner's responsibility to take the text of the code that the visual element contains and map it onto this finite state structure. In Fluid this is done by executing the entire code box once and looking for the value of a "return variable", ***r***. By interpreting the value of *r* — which may be any one of a number of Python objects — the runner interprets what part of this transition diagram gets populated. The following summarizes the return values that have been found useful inside the current Fluid system (over *how long..., 22, Imagery for Jeux Deux*):

nothing, *r* remains unset after execution. In this case, the visual element wants no further execution. This doesn't mean that it never executes, in fact it has already been executed once to see what *r* was going to be. Such visual elements, therefore, are executed only on "start", "jumpOverForwards" and "jumpOverBackwards".

***r* = an-executable** (any of a class containing a Python function, method or generator, or a Java or Python instance implementing Updateable). In this case the visual element has offered up an object that should be evaluated or executed for each execution cycle that this visual element is "running". Generators are called only until they no longer return values. In this case, nothing additional is executed in the case of "jumpOverForwards" and

"jumpOverBackwards".

r = **a 3-tuple**; *r*=(start-executable, continue-executable, stop-executable). In this case the visual element has offered up something for each of the "start", "running" and "stop" stages of the visual element life-cycle. This case is by far the most commonly used case throughout Fluid.

r = **a dictionary**; *r* = {*start:*start-executable, *go:*continue-executable, *stop:* stop-executable, *jumpF:* jump-executable, *jumpB:*jump-b-executable} — the completely, and rather more verbosely, supplied dictionary of things that might be executed. Any of these can be omitted without error.

In addition to reading this "return value" the runner ensures that certain variables are configured before execution (and before the execution of the returned components of r in the future). These are purely for convenience and readability; all the information is available from the Fluid interface with a little indirection.

_t — the normalized position of the time marker through the visual element. This can, in the case of start and stop parts of the life-cycle, be greater than one or less than zero.

_dt — the normalized instantaneous velocity of the time marker.

_attributes — the persistent attributes dictionary for the visual element. This is a window onto the visual element from the outside world, and a place for the visual element to store things that will survive across executions and even across application launches. This is also how visual elements customize some of their user interface — Python functions that are stored in this dictionary become menu items for the visual element, numbers become interactive sliders, and strings become editable text boxes.

In the case of multiple time markers the **_t** in the execution environment is modified to become an instance which masquerades as a number, but contains all the **_t** information from each of the time-markers should the visual element require access. At present the scalar version of this is the **_t** corresponding to the most recently created time marker.

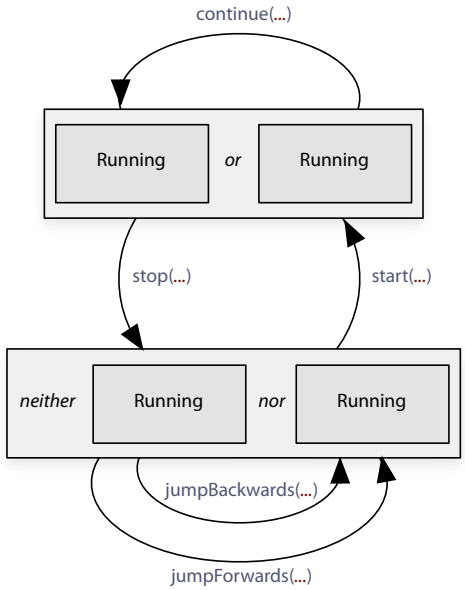


figure 155. The full state and transition diagram for an element on a sheet with two time markers.

Finally, we note that there can be multiple time-markers at work on one sheet — we'll see below how this can become increasingly useful in more complex sheets. Indeed, there is, in addition to any time-marker on a sheet, another Runner, one corresponding to explicit mouse-clicks on the visual elements. Fluid elements can be executed by option-clicking on them, which spawns a time marker local only to that element for the duration of the mouse movement.

This means that runners must organize themselves such that the life-cycle transition diagrams for individual markers can be effectively shared. This is achieved through a context-based parenting mechanism — specifically all the children of a runner share the activations of the parent. Although there are a great many ways of taking the product of two of these state-diagrams, in practice only one based on the logical "or" of whether a diagram is executing has been of use. Specifically, if any runner claims a visual element as running then it remains or becomes running. One could imagine forming "and" and even "exclusive-or" intersections between runners and their time-lines, but so far no project has needed them. This multiple, distributed-access state diagram is supported using the deferred dispatch and channel rewriting capabilities of Diagram, *page 273*.

Clearly, even with our time-line example there is quite a lot going on — we should begin to look at how these elements are coupled together, and how the sheet assemblage is designed and perhaps even more importantly stored over time. We have spent some time analyzing the conditions under which tight coupling between systems occurs in the agent framework and building techniques such as the context tree that prevent relationships between apparently independent code fusing solid.

Firstly we'd like to be able to reuse visual elements inside different contexts, different sheets, and specifically, according to our design principles above we'd like to be able to use them non-visually. This means that they must communicate with the visual presentation system but not couple to it. Fluid makes extensive use of two techniques — a tree delegation system and an external extension mechanism. The first technique is similar to the delegation chains used for event handling in many windowing frameworks. But I extend this idea to allow arbitrary method calls to be propagated up a branching container chain in a breadth-first fashion: from element, to containing elements, to the sheet and eventually to an interface to the containing agent. All of the event handling, execution and visual presentation is handled in this open, over-ridable way.

The Cocoa application framework that is used to implement Fluid also uses delegation chains in part to deliver events from input devices to visual elements, and to delegate method calls; Fluid extends this technique to include the storage of attributes.

Data storage acts in a similar way to the delegation chain, external to the visual element itself. The component of "plug-in" architecture of Fluid is fundamental, rather than just an extra layer of extensibility. Python code execution is a plug-in, time-markers, constraint systems are plug-ins, the very visual position and size of the "visual" element is maintained by a plug-in. The actual information maintained by the visual element itself consists of nothing more than a unique ID. Plug-ins are added to the sheet and as a result offer the ability to set, get, store and delete properties with respect to this tree of containers. This allows

plug-ins to overlay services into the sheet: by manipulating the container chain, plug-ins can affect the default behavior of some or all of the visual elements. This allows extensions to the Fluid system that are "multiplicative" rather than "additive", extensions that alter the ways that sets of visual elements can execute, combine and can be manipulated, what the visual elements actually present visually, and how they act visually. This is in stark contrast to systems such as Max where "externals" simply add to the numerical quantity of the modules available.

Secondly, although the quantity of code stored by any one sheet is much smaller than that of the framework supporting it, the situation is just as important from a storage perspective. For in order to commit to an environment one has to trust that it will always be able to recover one's work even after a several month hiatus, during which time the agent framework might have changed, but more importantly Fluid itself might have undergone revision. The file structures of Fluid were explicitly designed to allow the environment to grow without losing the ability to load previously saved files. This, in itself, isn't a particularly hard problem, and can be achieved by storing versioning information in the files (for similar techniques, see the long-term learning database, *page 143*). However, it's also important that the environment can shrink or that sheets can be loaded into and saved from completely different environments. So, for the purposes of long-term storage, data travels with both the plug-in and with the visual element. Unknown data is both carefully ignored, carefully propagated, and in most cases still accessible even in the absence of an actually executing plug-in. Plug-ins are defensively coded to verify the relationship between their internal structures and what remains in the individual visual elements upon load. In practice, two-year-old sheets are still loadable today.

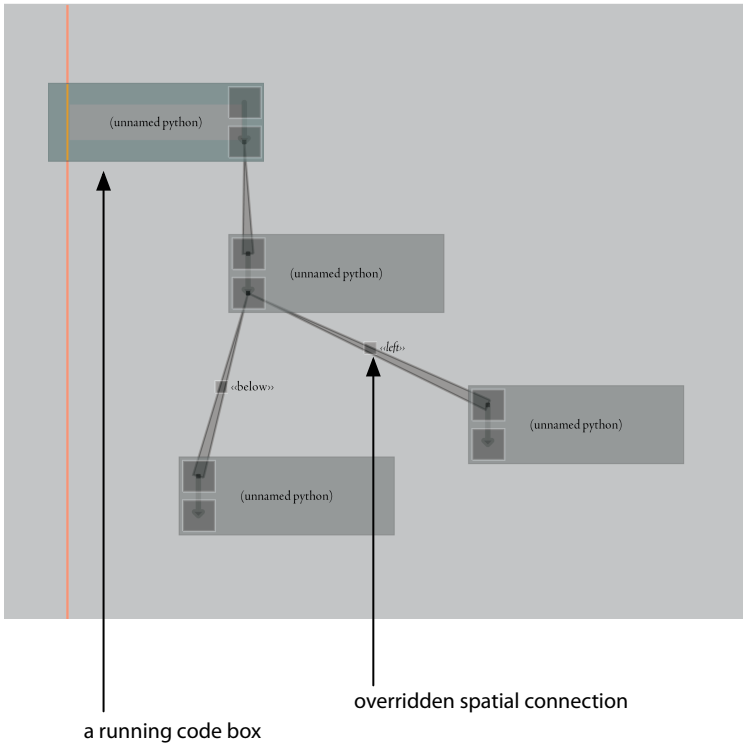


figure 156. Code elements linked together, either by hand or by code.

In a fashion similar to that of data-flow environments, we can add to our elements inputs and outputs and begin to draw connections between boxes. The values at the connections can be pushed from outputs to input from within the Python environment:

```
_output[0] = 5
```

— but these connections are not necessarily for data-flow. These connections manifest themselves as set variables inside the Python environment:

```
print _input[0]
```

however they do not necessarily "represent" the flow of data. They might represent the aliasing and thus sharing of variables between otherwise local python namespaces.

From the output module:

```
makeAliasOutput(0, "a")
a = 5
```

declares that the variable *a* will alias the zero-th output, and from the input visual element we might have:

```
makeAliasInput(0, "a")
print a
```

By going further, and propagating not values, but small objects that reference the visual elements from which they come, visual elements can conspire to appear to implement a style of data-flow programming.

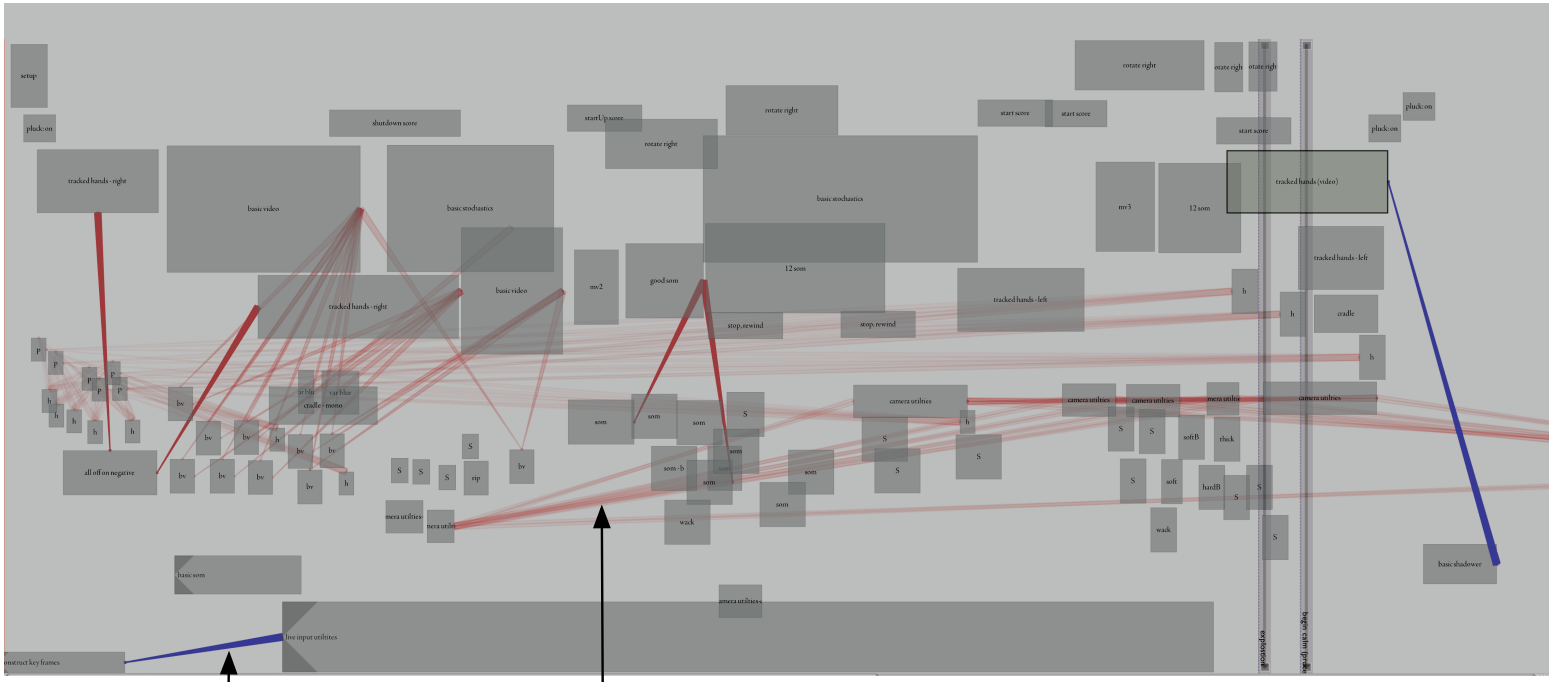


figure 157. The history of ephemeral, implicit connections between modules can be visualized as a separate "layer" to the environment. Such connections are gathered by automatically monitoring the Python interpreter.

From the output module, it looks the same:

```
makeLiveOutput(0, "a")

a = 5
```

declares that the variable *a* will alias the zero-th output and sets things up to allow this visual element to be executed if needed to evaluate *a*. From the input visual element we might have:

```
makeLiveInput(0, "a")

print a
```

What are the contents of these small objects? Since anything can, with the assistance of a Runner, ask for the execution of another element, these objects ask for just that — to ensure that it has the latest value at its inputs. These executions are safe and timely. They are aware of that runner's life-cycle state diagram and thus the element that is asking for the computation is guaranteed to both maintain the correct life-cycle contract of the element and only cause one execution per execution cycle. This hybridizes a pull-based data-flow style with a more orderly time-marker style, proving that data-flow, variable execution and alternative visual metaphors can coexist on a single interactive surface.

In data-flow environments, one connects boxes and these boxes remain visually connected — as a "visualization" of the history of interaction, and as a "user interface" that allows the connection to be broken, and restored. Since Fluid visual elements contain code, Fluid offers many other ways of "connecting" elements together by writing code rather than by interacting with the sheet using a mouse: code can look up an element by name, by regular expression; code can find the visual element to the left of it, all of the visual elements underneath it. That these global or spatial lookups can be written in code rather than made by mouse-clicks is a fundamental result of the principle that code can "see" the visual layout of the sheet. However, at the same time, there is much to be said for allowing the environment to provide a "visualization" of what that code is doing and a "user interface" for allowing these connections to be rearranged.

The solution is to construct a visualization layer that, by collecting information from the python environment, as it is executing code, annotates the sheet with the connections that the code makes and offers the opportunity for these connections — which might be one visual element connecting to an element to the "left" of it — to be frozen or reconnected. These layers are translucent, optional and overlay the sheet — they are coupled to the sheet through the plug-in architecture.

Inside the Python these connections look like:

```
target = leftOf()
```

or

```
target = find("fade out *")[0]
```

This provides enough stability and flexibility that the layer can edit the environment of the visual element to ensure that connections overridden by direct interaction continue to be overridden. The connections fade over time as they remain unmentioned by the executing code.

leftOf(), find(...) etc. return Python objects that masquerade as references to other visual elements, and exploit the *_attributes* dictionary to ensure that they remember whether or not they have been overridden in the sheet.

There are additional reasons why we would like to be able to trap and interpose all references to external visual elements by the code inside a particular visual element. We'll see the importance of being able to draw a circle around the context accessed by an element when we look at recording the history of interaction with Fluid, *page 410*.

This highlights two of the design principles: that code and visual elements should coexist on the same sheet, and that the history of using the environment should be reincorporated into the environment. This interplay between making the results of code visual and turning visualizations back into code is what constitutes one axis of "fluidity" inside Fluid. But before picking up the thread of incorporating use history into tools in a more focused fashion, I will survey some of the other kinds of layers implemented in the current Fluid system and how they are used.

I. Sutherland, *Sketchpad : a Man-Machine graphical communication syste*m, Annual ACM IEEE Design Automation Conference, 1964.

Of the most important layers available in the current version of Fluid are the constraint systems. Constraints have a long history in visual layout tools — in particular they form the very basis of one of the very first visual layout tools Ivan Sutherland's seminal Sketchpad system. But, despite Sketchpad's hybrid programming / drawing approach, visual layout constraints are completely absent from the history of visual programming environments for digital art — absent as a tool in the Max/Isadora/vvvv series, for visual layout is unimportant in these applications. Paradoxically, each of these graphical systems offer less support for fast visual layout than most drawing or painting applications.

However, when the visual layout of the sheet means something — to the code contained as well as to the user — and when the environment is the platform for a certain amount of improvisation, it is important to allow the specification of more complex and quick manipulations of visual element layout.

Two constraint systems are implemented inside Fluid; both have the same interface and appearance and allow a conventional set of constraints to be specified on the layout of the visual elements. Same, Before (and by inverting the parameters After) relative constraints on both the start and the end of elements; Same, Bigger (and thus Smaller) on the duration of elements. These constraints can be applied to visual elements that group visual elements (and distort their contained children equally when needed). Finally, elements' positions can be pinned to a particular spot — this allows all of the previous, relative, constraints to have an absolute aspect, since visual elements to represent absolute positions and sizes can be created with ease.

The perennial problem with constraints, however, is that it is extremely easy to construct over-constrained systems, and extremely hard to build fast but stable solvers for these systems. Although much work was done in Sketchpad and

afterwards to address these issues, Fluid dodges the problem creating two constraint implementations, neither of which has any claim to optimality, but rather a focus on stability and speed of execution. In the future, a more complex linear-programming-based constraint system could be implemented.

The first constraint system is a rather typical damped iterative solver that tries the best that it can, with a decaying amount of effort, to maintain each of the constraints in turn. Should a constraint be broken (due to over constraining) it is brightly indicated on the sheet. This ad hoc solution works well for under-constrained problems and tends to break stably in over-constrained situations. Never has a sheet "exploded" during all of the improvisatory use of Fluid in developing *how long..., 22* or *Imagery for Jeux Deux*.

The second constraint system is a little different. It is based on the generic radial-basis channel formulation for competing processes. In under-constrained domains it acts the same as first constraint system — it is a damped, iterative solution to the problem. However, rather than trying to find a nearby, stable solution to an over-constrained problem, the competitive channel representation gives each constraint a certain amount of time to apply. This "solver" actively explores the over-constrained partial-solution space, generating not an attempt at a single solution, but an ongoing animation. To date I have used this solely as an exploratory technique (for the generation of rhythmic cells that are perturbed in different ways), or a visualization technique for the similar Diagram based processes of *Loops Score*, *page 258*. It is expected that in the future this technique will autonomously create rich rhythmic patterns in the domain of motor systems.

Regardless, the constraint system is perfect for making visual and maintaining the ordering constraints of the visual elements' code — that one thing should take place before another, or that this visual element cannot end before another

— that allow sheets to be quickly reorganized during rehearsal before calling upon a time-marker to "scrub" with.

Another available layer is related to the constraints system — the layout snapshot. This is a duplication of a sheet (in the sense given below, *page 410*) that saves the positions and sizes of all the elements. This layout can then be blended with the current layout. New visual elements that are not present in the saved copy can remain stationary or, more usefully, can get pulled around by the nearby saved elements movement. This reuses the same techniques as found in the Diagram channel system, *page 248*. A pose-graph-based view of these snapshots exists, and in the future we might see an agent acting upon a Fluid sheet itself.

Fast visualization for the agent toolkit

During the use of the agent toolkit many programmers — inside and outside the Synthetic Characters Group, myself included — have produced carefully crafted visualizers and debugging tools for various systems. At any given time one could expect to find a motor system visualizer or two, three or four for the context-tree and so on at various stages in the development of the agent toolkit. With modern, graphical tools, building and maintaining these tools isn't hard, nor is it as time consuming as it used to be. But it does require a constant effort parallel to the development of the system being visualized. And there is a constant tension between creating a well-designed interface (code) for the interface (visual) and creating one quickly. As a result, these carefully crafted visualizers are often out of date at any particular point in time — if, that is, they get created at all.

Fluid potentially offers much more than either a hand-crafted visualizer or a traditional debugger since it integrates graphical user interface construction

tools, code execution and domain specific storage in one place. In order to bring the toolkit closer to the Fluid sheet, it is important that the toolkit can offer objects *to* the sheet on an equal level to the visual elements; that agent toolkit objects can be visual elements — that one can connect to the motor system or a pose in the pose-graph, visually and spatially. This is important even in the simplest, and least "creative" use of Fluid — having a sheet be a place where visualizations of a running system can take place.

Clearly, it isn't hard to have the toolkit load a sheet and procedurally create visual elements inside it, but some caution is needed — if these "offered" visual elements are to fully participate in the Fluid framework they need to participate in the long-term storage of the sheet. This implies that offered elements, which are free to change in number and nature from invocation, to invocation must be matched up with visual elements that are free to be edited, moved around and otherwise adorned as they are loaded from the persistent store.

Three sets of parameters must be considered in this merge — the new creation parameters offered by the toolkit, the creation parameters previously offered by the toolkit (at the last occasion that the sheet was saved) and the parameters now specified by the sheet itself. The differences between the first two are applied to the third unless there are corresponding differences between the last two — this is the classic, three-difference merge algorithm applied across a set of attributes, and the visual element position. Later, we will see another application of this algorithm to the textual contents of the visual elements, *page 410*.

For an overview of the three-way merge algorithm — **T. Mens.** A *state-of-the-art survey on software merging*. IEEE Transactions on Software Engineering, 28(5), 2002.

Once offered, these visual elements are now a bridge between the agent-toolkit and Fluid. However, both parties ought to be able to create simple layouts and interfaces that are more complicated than a simple box. Fluid, of course, allows one to surround these offered elements with code and other visual elements.

Yet at the same time we should realize that most "debugging code" exists inside the agent toolkit as textual output not user-interface construction code — how should these pieces of text describe user-interface layout? The quickest and simplest debugging output statement from deep inside the agent toolkit looks like the following:

```
stream.println(" motor system value :"+amount);
```

These statements are ubiquitous throughout programming — *stream* could be an interface to a complex logging interface or simply an interface to the system log. The sheer number of statements like the above make the use of such code seem almost inevitable. The prevalence of these lines inside the agent-toolkit seemed impervious to the increasing flexibility and availability of visual user-interface design tools prior to Fluid. All collaborations (and all programming collaborators), from *alphaWolf* to *how long...*, include these lines.

It's not hard to see why they might be more maintainable than a hand constructed interface — they are programmatically described, compiled with the system that is being investigated and require no interface for that system to be created and maintained simply to get at the misbehaving number. If, in comparison to contemporary data-flow tools we are to conceptually embed complex systems *inside* our visual elements, rather than construct complex systems *with* visual elements, there ought to be a way for these opaque complex systems to talk back to the visual elements.

So we start here, with the kind of talking that seems so prevalent, and construct a incrementally more complex "stream visual interface" to the visual element. Offered visual elements provide an object, "stream", that can be written to as above. We augment the traditional stream output with the following features which augment the visual layout of the stream and provide a lightweight bridge to the Fluid's visual elements:

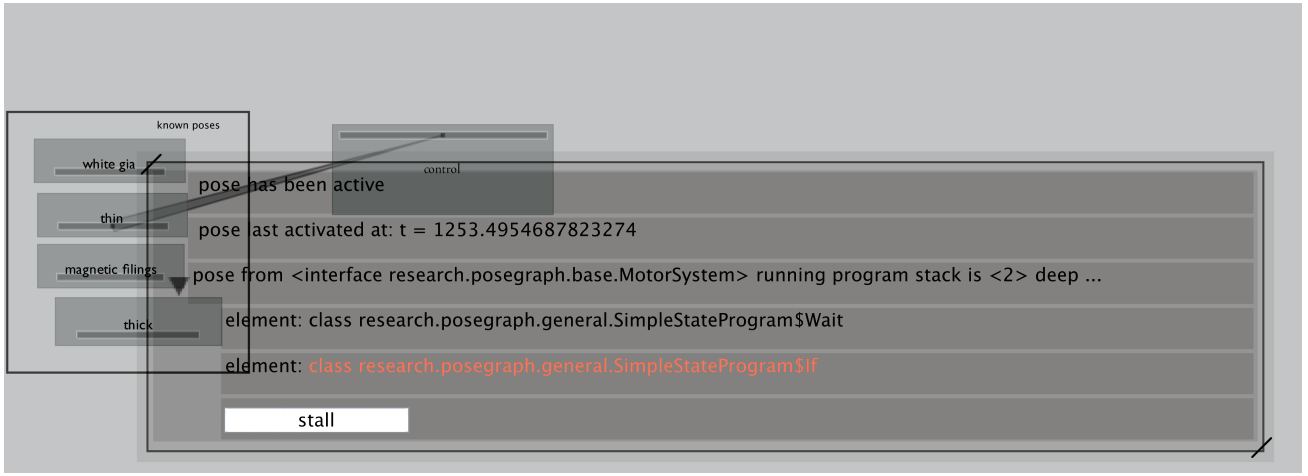


figure 158.
A dynamically created, *ad hoc* debug display and interface

Text "lines" become rows in **an outline view** — rows are collected only for a fixed number of update cycles and such cycles group the output; the elements "[[name" and "]]" bracket sub-branches of the tree. This allows the debugging output to be presented in a hierarchical, multi-resolution fashion.

html processing tags are acceptable — since we are free from the assumption of plain-text output there is no reason not to allow a subset of the rich-text format to be displayed.

Stable user-interface objects are possible — the tag "<button name='name'> label </button>" writes "label" not in a hierarchical outline-view text row, but rather a button in the row; the tag "<slider name='name'>label</slider>" makes a labeled slider.

The values for these two interface elements are written as attributes to the visual element and can be read by the agent toolkit as:

stream.getAttribute(*name*)

From the Python interface these attributes are read and written simply as the value "*_attributes*.name". The hierarchy of debugging output can be parsed (in plain text) through an object called *_debugStream*. For example:

_debugStream[2] is equal to "motor system value :5.0"

and

_debugStream.motorSystem[0] is equal to "at SIT"

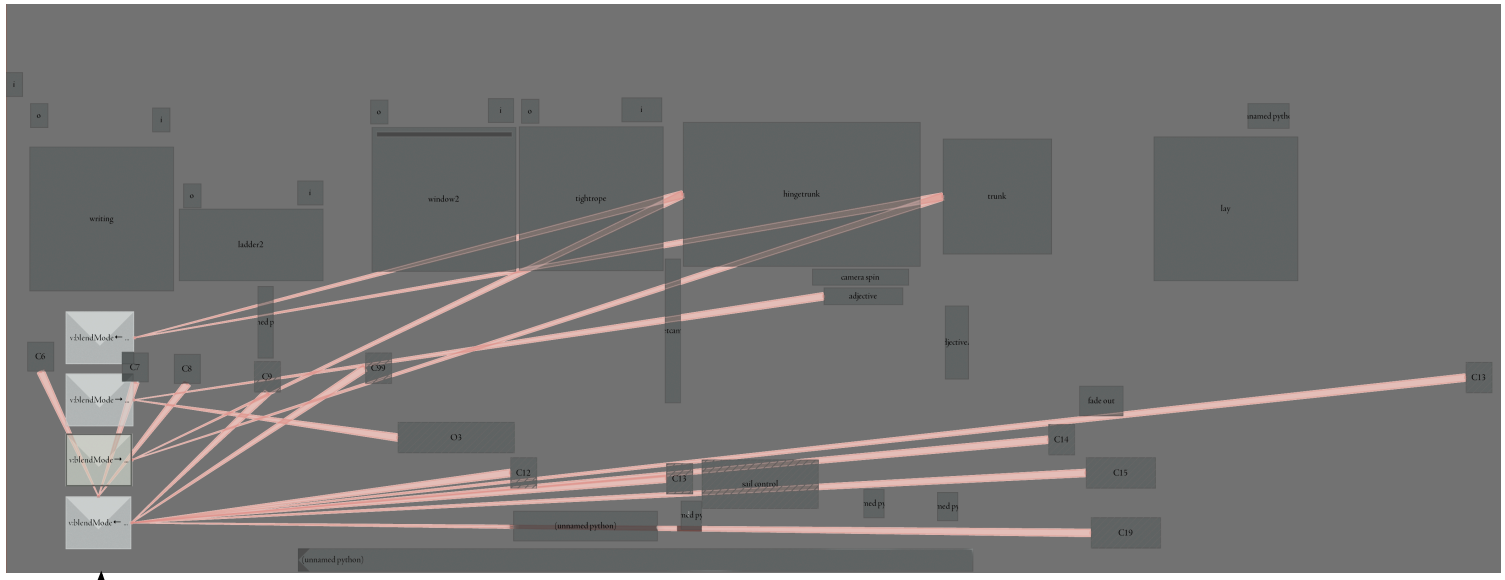
These two accesses to the debugging information mean that the visual elements that surround the offered element, and the code inside the offered element can access everything about the *ad hoc* debugging interface's output. This text-based graphical visualization completely avoids the overhead and complexities of creating visual interfaces and code interfaces that they connect to, which is a particularly error-prone area of programming. One must take special care to maintain the same behavior of a system regardless of whether anyone is looking or not. This often requires the caching away of transient data and, depending on the windowing toolkit used, may even have thread-safety issues. The near inevitability of text-based debug output, and the error-prone nature of the mix of user-interface code inside the agent-toolkit stands as one of the lessons learned from the complex collaborative endeavors from *alphaWolf* to *how long*.... This push-based debugging, although offering a more generic, more rudimentary visual presentation, meets the complex code-base where it stands — the rest of the Fluid environment can be used to customize the presentation of information. Fluid becomes a site of interactive visualization and investigation that meets the agent-toolkit on the toolkit's own terms.

During the course of creating the piece *how long...* the master sheet that controlled the piece was loaded and modified 206 times; secondary sheets, for testing elements and working on specific sections were loaded and modified 3223 times. With the exception of some 45 unexpected fatal crashes which may have resulted in data loss, a detailed history of the creation of the piece, and all other works created in Fluid since May 2004 (when the database came online), was stored. But what is a "detailed history" of working inside Fluid? and, equally importantly, how should it be made available to the tool itself?

Programmers have long surrounded even solo work with versioning systems that allow them to consciously checkpoint their work — storing it in a central database format. Concurrent versioning systems, designed for more than one programmer to work on a set of resources at the same time, are also the backbone of both the open-source movement and almost all large closed-source development models. This is very much prehistoric computer science — the core formats and algorithms for storing and generating annotated views of the changes that resources undergo in these systems have been found and fixed for decades.

For a history of one of the oldest version control systems that is still in use today: http://www.gnu.org/software/rcs/rcs.html

But the importance of making the history of the development present in the tool itself was brought to my attention in at least three ways. Firstly, by the presence of the "commented history" throughout the text of *alphaWolf*, *figure 29, page 100*. If this history was so important as to be preserved in the files themselves, despite universal struggles with the bulk and complexity of those files, perhaps this is an indication not only of the importance of developmental history but of the inadequacies of conventional version control tools (which were also, of course, very much in use during *alphaWolf*). Secondly, by the realization that I repeatedly required access to the history of many of my persistent stores



stack of read / write access to a global variable

figure 159. This figure shows a live, layered display indicating read and write access to a sheet-local variable

— the long-term learning databases of *The Music Creatures*, *page 143*, and the bundles of parameters in *Loops*, *page 114*. This history was not always in a form that conventional version control systems found easy to use.

Finally, in watching my own work patterns inside Fluid I identified a number of cases where having history at my disposal would prove useful. The simplest case, and the most important given the nature of Fluid, is history of code execution. In a fully thought-out sheet, execution is often under the control of the visual elements themselves — intersecting time-markers, moving visual elements, running scripts and so on. However, early in the development of something, or when some specific case is being explored, execution is often much more piecemeal — one has found a case that doesn't quite work right, or one is beginning to test a new component — through highlighting parts of code and executing them, using a Fluid sheet as a sketchbook to sample from rather than as a place to put ideas down. Further, in improvisations one is often moving too fast to remember what one is doing.

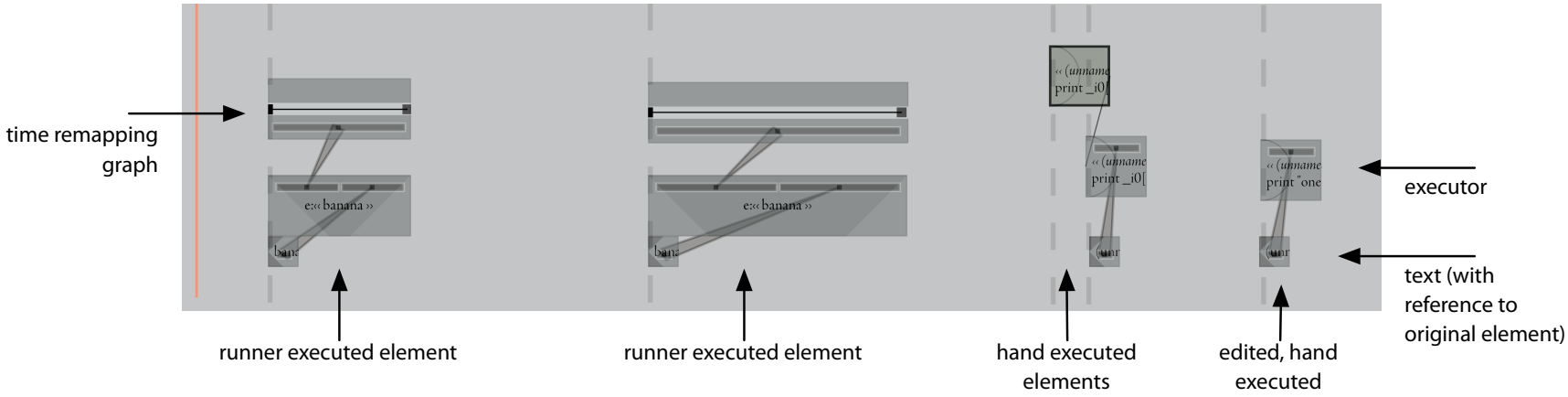


figure 160. This sheet was automatically created from the interaction history with another. This "unrolling" of a marker sweep and piecemeal execution of code inside visual elements is itself executable and remains linked to the original sheet.

What each of these cases really needs is the execution history of a sheet and its attached textual editors — what code, in what elements, when. Early in the development, when executing samples from a sheet, or samples from a long unstructured piece of textual code, one ends up trying the same pattern of execution repeatedly — to get back to the place where a problem occurs or where the horizon of knowledge lies. In improvisational contexts one needs to go back and look at what was done in the heat of the moment. The solution is to begin to look at ways of turning the execution of a sheet into a new sheet.

As a naïve start, we can take each executed element and copy it to a new sheet, a time-line sheet, where execution time runs monotonically and evenly from left to right. This, for example, "unrolls" or flattens-out any temporal manipulation that was happening to an underlying time-line sheet or "scores" an improvisation that sampled from various parts of a sheet in an *ad hoc* fashion. Since highlighted snippets of code can be executed in the textual editor, these need to have visual elements created for them. This is the (visual) equivalent of converting a marker generator in the Diagram framework to a channel representation.

This unrolling must carefully propagate a snapshot of the local environment of the visual element at the time that the element or snippet was executed to the newly created elements — otherwise the new sheet will not perform in the same way when executed. There are up to three places that this "local state" can be put in relation to the newly created visual elements: back inside the local context of the visual element, as an explicit addition to the code stored in the element, and as a separate, but (visually) linked element on the sheet.

The "local context" to a piece of code executing inside a Fluid visual element is a rather complex affair — but in all cases we can trap it by a placing a few hooks into the Python interpreter. The complete context caught by the unrolling history functionality is as follows:

Python-level local variable access — this needs to be recorded in the unrolled sheet, if it is read by the visual element or script before being written to. It can be a separate element — injecting a value into the new element's local variable space — or additional code in the new element's textual description.

context-tree variable access — very similar to local variable access; writes and reads are typically annotated on the unrolled "score" as separate visual elements. Making these global accesses explicit is a useful visualization understanding. Below we shall see more advanced uses of this score-like style.

visual element persistent attributes — visual elements have a stored (across loading and saving sheets) set of attributes that can be read or written by code, or by inspectors. Reading or writing these requires duplication in the new visual element. This is always performed by making new stored attributes.

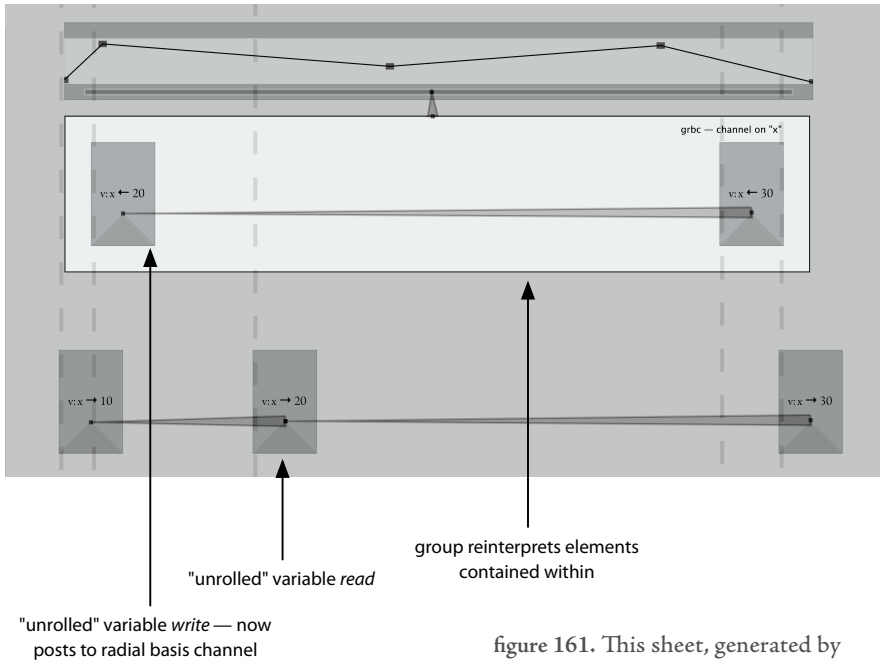


figure 161. This sheet, generated by unrolling another, has had some variable accesses grouped together inside a subgroup that "reinterprets" the code that writes to those variables. Now, rather than directly setting values, postings are sent to a generic radial-basis channel.

sheet-level access — what should operations which result in obtaining references to other visual elements return in the duplicated sheet? Should a visual element A find a visual element B that also ends up duplicated in the unrolled sheet, then we can transport the reference, making a new reference from the duplicate A' to the duplicate B'. Should B not already be duplicated, then we either have to try to copy *B* or make a new reference to the original element B. Currently this second operation seems well defined and Fluid makes a cross-sheet reference A'→ B and allows this reference (using the same techniques as we use for overriding spatial references, *page 399*) to automatically load the missing sheet, if needs be, on access.

Since these "visualizations" of the interaction history are executable, we can sweep time across the sheet and play back what was done before. We are free to take these sheets and begin to edit them — changing the order of operations, splicing them with alternative takes, etc. We are also free to re-express their contents in a different way. One highly useful modification of a sheet that is typically produced by this unrolling is to take variable access and replace it with generic radial-basis channel postings.

The new group, surrounding the variable reads and writes re-interprets the contents of the code below — wrapping the execution environment of the visual elements in a structure that maps variable access to generic radial-basis access. One channel (sharing a time-base with the sheet) per variable is created as needed by monitoring the underlying Python interpreter for global variables, and temporarily overriding the object that is used for context-tree access. Writes become postings (with window parameters set by the duration of the visual element) and all reads return python objects that masquerade as numbers, but access the corresponding channel.

Even without their modifiable executability, these unrolled sheets or score-like diagrams have been extremely useful in both remembering and showing what happened in an improvisation that took place in a theater under time constraints dictated by dancers and musicians. But this kind of use is a short-term use: logging information taken in the moment is there to be looked at soon after and understood, perhaps played and replayed with a little more, but strictly from the point of view of understanding what took place.

These sheets, as described thus far, cannot offer a longer term record of what is taking place, because they go out of date. They loose their connection to the sheets whose execution they annotated when those sheets change. They are a tool for recording only so far as they become separate from what it is that they record. Is there a deeper way to link the record of an improvisation around a sheet to the sheet while, at the same time, maintaining this connection through potentially separate evolutions of these sheets? A history of interaction with a tool that isn't a frozen record but a new view onto the material interacted with?

A network of text: copy & paste as a version control system

At the root of this issue is the problem of duplicated textual code. Currently copy and paste is a ubiquitous part of any textual interface — there is hardly any text box or textual widget that does not support this on any contemporary windowing system. However, a copy and paste operation leaves no process trace. Further, the nature of the process, this duplication of code, is invisible to extant programmers' version control systems. These systems realize that code has been added somewhere, but do not retain the connection between the copied and the copy — for that connection is lost a long time before the versioning system operates on the file. Typically this is not a significant problem in versioning control — versioning systems are so old now, that had it been a problem we might have seen a good few solutions, especially as such systems are being integrated into

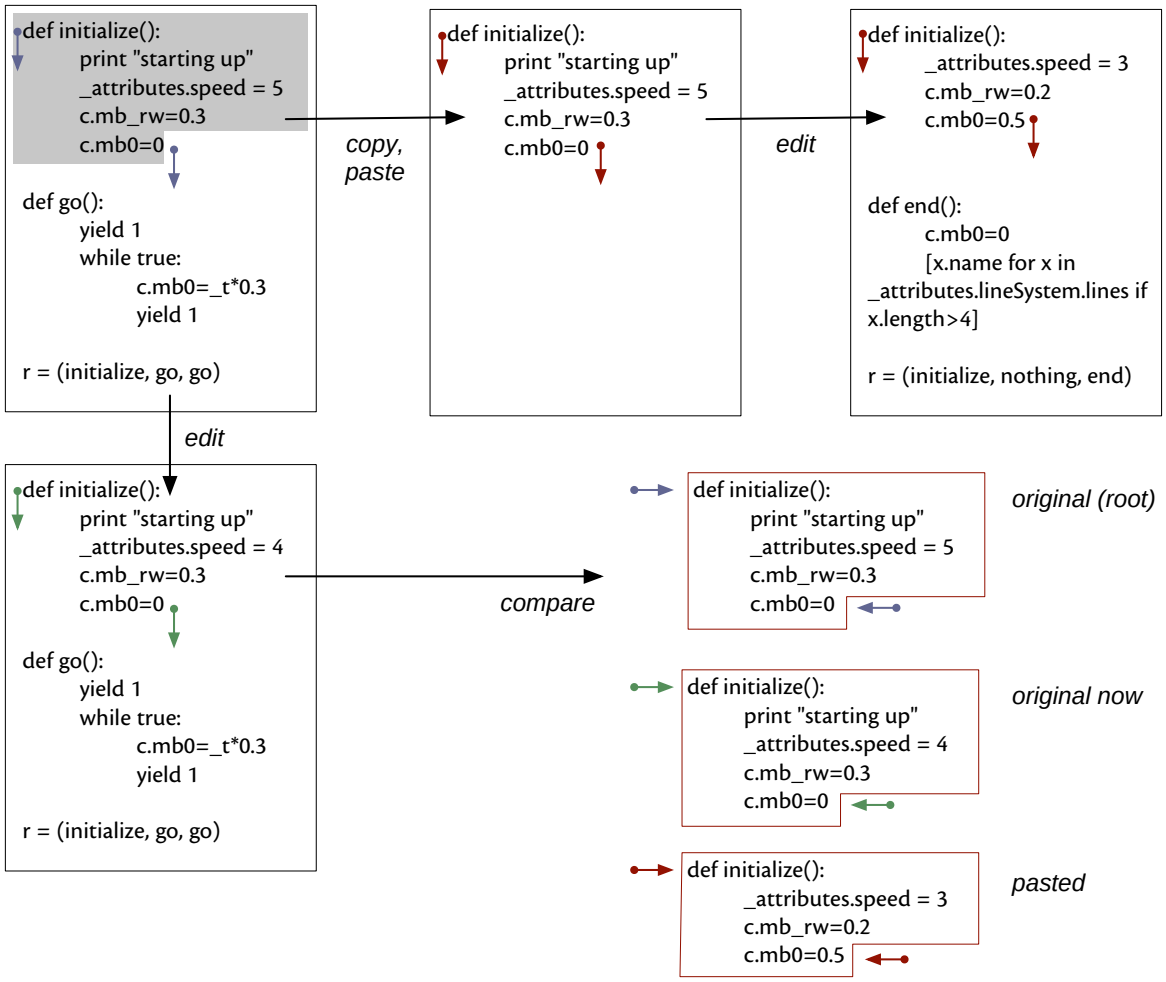


figure 162. Copy, paste and edit creates a versioning "problem" and an opportunity to use the three-way difference algorithm to inspect the history of snippets of code that are transfered around and across sheets.

programming environments. Copy and paste, after all, is often considered a symptom of a poor programming infrastructure, and some of the classic design patterns and indeed some of the motivations for object-oriented programming itself are to reduce the amount of code that has to be copied and pasted in order to program.

However, the theoretical goals of a programming language and its use in practice often diverge dramatically — despite the inability to accurately reconstruct the copy and paste history of, for example, the source code files of *alphaWolf*, one can feel its presence throughout. The simple solidity of taking one element that is known to be tested and working and duplicating it (rather than refactoring it in such a way that it can be multiply instantiated and in doing so potentially break what is known to be working) is a powerful temptation even for the best programmers when under pressure.

In any case, in Fluid, we are are operating in a completely different domain from where these arguments for careful object-oriented design typically take place. Rather than large, refactorable, and pre-thought-out complex codebases, Fluid's visual elements are an *ad hoc*, often improvised arrangement of very small parts. That copying code is simply the fastest and easiest

thing to do (rather than re-factoring the design of material inside a visual element) is much less avoidable in this domain. Rather than reinventing the theory of re-factoring to cope with the kind of fragmentary, poorly planned, spontaneous code that Fluid encourages and circumstances dictate, we do the opposite — shape the tool around the use, and open up the history of duplication to the versioning systems.

Thus, in Fluid, copy and paste operations leave persistent process traces. Fluid exploits the commonly used *rich-text format* for the storage and the presentation of code to the user. By embedding custom tags inside the text structure of a modern text formating system we can annotate the relationship between the copied and the copy in a persistent fashion. We can then recreate the common versioning system operations not in terms of files of code, but of chains of copied and pasted elements.

The rich-text format — http://www.microsoft.com/downloads/details.aspx?FamilyID=ac57de32-17f0-4b46-9e4e-467ef9bc5540&displaylang=en

The use of it here depends on the application framework's handling of alien RTF tags, which seems implied by the specification. These tags are, in the current implementation in Cocoa under Macintosh OS X, persistent across all applications that deal with text, not just internal to Fluid — they mark blocks of text as having references to a database through the use of unique IDs.

Given a snippet of text we can perform the following three operations on it with respect to viewing its history:

show resource — when a copy / paste relationship is first created, a resource is created with it. This is the central representation and marks that this particular piece of code is important and should be tracked. Everything else, the copied and the copy, has a relationship with this resource. By looking at the resource we can then see everywhere this text ended up, or where it came from. By looking at these resources, we can compare how they have changed, or how they are being used. We can begin to examine the ramifications of changing something that this code depends on, and begin to repair it when we do change it.

force changes to (resource, later, earlier, all) — this forces an overwrite of the contents of the text snippet-to the resource or to a subset of children

of the resource. The labels "later", "earlier" and "all" refer to child snippets that were created either later or earlier than this particular snippet of text.

merge changes to (later, earlier, all) — this performs a three-way difference merge with this text, its resource and each of the places linked to this text that appeared after or before this relationship was created. This difference / merge algorithm is standard as part of a concurrent version system — here, however, it isn't the work of different programmers at different times that are being considered as happening "concurrently", it is the work of one programmer in a number of places. Unlike version-control systems, the "files" (fragments of text spread across visual elements and sheets) that need to be considered are being automatically inferred. Collisions (incompatible, "simultaneous" changes of text that cannot be reconciled) are flagged for special handling.

loose snippet (resource, all) — breaks the connection that this piece of text has with the database with respect to a single database resource or with respect to all resources associated with this snippet.

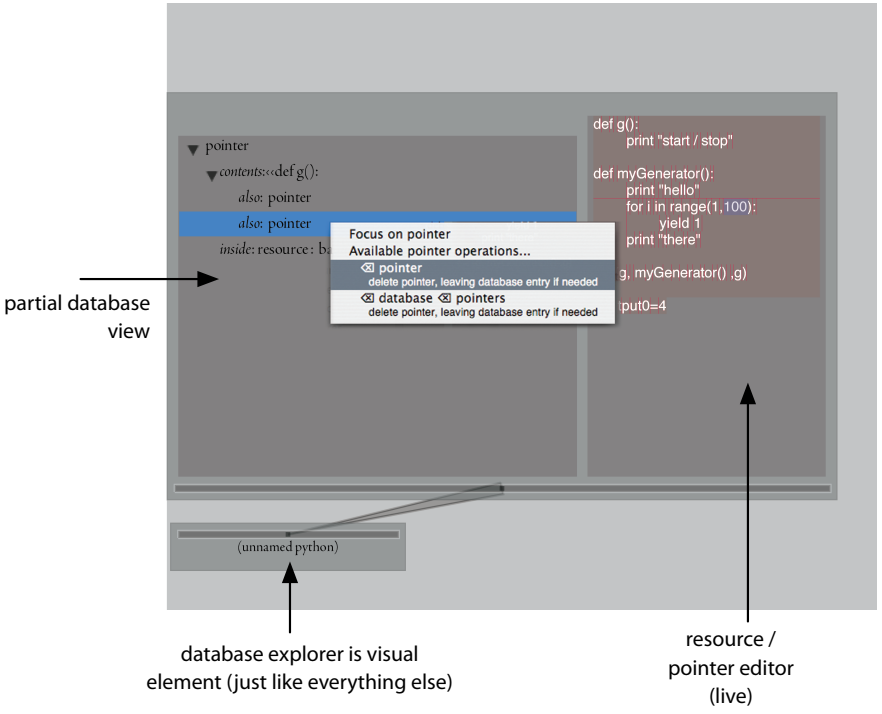


figure 163. The text database explorer interface can be manipulated from within the fluid environment — it is constructed from visual elements.

The interface shows a local, hierarchical view of the database — resources point to snippets (ordered in time) as children; snippets have a resource and a visual element associated with them; visual elements contain multiple snippets. The textual contents of all these elements can be browsed and edited without loading the associated sheets. The database of resources and concurrent snippet versions is maintained in parallel with the text storage of the individual sheets. This provides safety in redundancy (since Fluid remains an experimental and evolving system); at any time we could delete the entire database, losing the text-level history information, but maintaining the present state of the sheets together with their version history (maintained in a more traditional version control system).

Finally, while I have described this system as one for keeping track of where copy and pasted code ends up — maintaining the relationship between copied and copy on behalf of the programmer — it also serves to maintain a connection between the unrolled sheets and the sheets that were unrolled — on behalf of the Fluid system itself.

The flow of time — more controllable time markers

The central idea behind the time-marker on a sheet is to allow a visual environment to start with what I believe is the one of the most central parts of the problem of digital art — the patterning of time. And in starting here, we start by analogy to the representation most present in the temporal arts — the linear score.

While I have argued that the agents constructed for the work I've presented do an excellent job of patterning the time that they occupy, and an equally satisfactory job of provoking ways of thinking about how that time of interaction could be filled, in many works there has been a layer either above or below the agents that has had a strongly score-like flavor to it. In *Loops* I constructed a colony of creatures capped by a score and noted that this began to look like a motor system of another super-agent, *page 117*; in *how long...* I deploy agents throughout the work but organizing their sequence and overlap in a fairly linear fashion to align with the performance, *page 373*; *Lifelike* is performed in a similar way, with a more complicated overlapping of less complex agents; in *22* we are in effect seizing control over the material that the motor system of the agent uses, *page 286*; at the detailed levels of parts of *how long...* we are constructing movement out of overlapping linear sequences, *page 361*; in *Loops Score* there is, again, a score not of notes but of opportunities for action, a "perceptual score", *page 259*. In each of these examples we are not so much scripting the actions that the agents will take, depriving the metaphor of its idea of autonomy; rather we

are either scripting the manipulation of part of the perceptual world that the agents are in, or providing scripts for the agents to manipulate in turn. In both cases the lines of these linearities cut right through hundreds of files of code and we are forced to make tools such as Fluid to make these broad strokes or detailed manipulations.

But clearly, this linear, or perhaps more accurately, the monotonic, score is the point of departure not destination. Thus we should begin to break down and complicate this scripting environment to bring it closer to the agent toolkit that it intersects with.

First, I shall look at a few mechanisms for controlling the time-marker as it moves across the sheet. Then, as these mechanisms get more sophisticated they will lead to multiple time-markers — time-markers that are concurrent, and markers that are under the control of some other organizing principle.

Improvisation on a time-marker sheet often takes the form of a combination of hand-executing visual elements, sweeping the time-marker, playing back previously made time-marker "scrubs" and of course, sometimes executing individual pieces of code straight from the text editor.

For example, consider a time-marker that is under procedural control, moving from time A to time B over a certain duration. How can a visual element that this marker strikes change the flow of movement? Perhaps it might try to slow down or pause until an event occurs, perhaps it might skip ahead to catch up with some other process, perhaps it might loop backwards to the beginning of some sequence while a condition has not occurred. Thus, we might consider two kinds of alterations: alterations of the speed of onward time flow, and unconnected jumps. However, these two categories obscure two other, perhaps more useful, categories: temporary and permanent time modifications.

It is hard to overestimate the need to both calculate and fix durations, to both give and receive durations, during a collaboration around a time-based work. Sheets of durations have always occurred as common language throughout all stages of development of *how long...*, *22*, *Imagery for Jeux Deux*, and even (rather illicitly since it was for a Cunningham anti-collaborative stage work), in *Lifelike*. Even works that appear far removed from the problems of occupying a period of time have such "ballistically scored" elements: the development of *alphaWolf* would have benefitted noticeably by the addition of such a representation for handling both the large-scale life cycle of the piece (a 5-minute "growing up period" of the wolves) and the small contrivances of the scene-setting introduction (falling asleep and being woken up by the participants). Such "scripts", be they interactively modifiable, cut across whole action systems and are hard to bring about in piecemeal, distributed architectures. It is appropriate to construct tools and passages of time that have more global and graphical views over the agent and its environment.

Of course, in remaining open to the interactive possibilities of the environment, such scores or scripts often don't remain ballistic for long. The importance of maintaining interactivity under a fixed duration constraint has ramifications for any process that wants to change how time flows through a work. In an area that ought to have a fixed duration, simple control of the rate of our time-marker is meaningless at best, and dangerous at worse. Rather, it is much more important to be able to, for example, slow down the apparent movement of time in such a way that it will speed up later to exactly compensate. Such non-permanent changes are vital if we are to be serious about moving away from the fixed linear score. Of course some changes are permanent — the duration of *how long...* varies by 10 seconds in performance and perhaps even 90 seconds during rehearsal because of permanent changes in the positions and durations of sections.

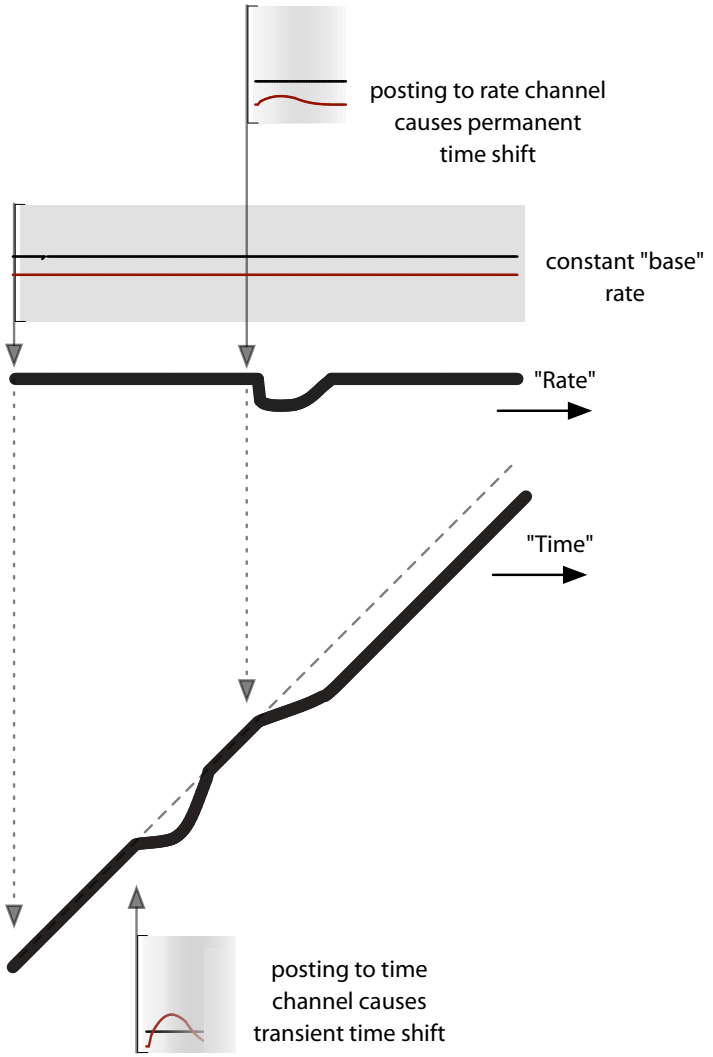


figure 164. Two generic radial-basis channels control the movement of a time marker allowing changes to the position that are both transient (in the sense that the duration of the work does not change) and permanent.

One time-marker control system to which I have had recourse in both *how long...* and *22* stacks two generic radial-basis channels: one controls the rate of increase of time which feeds into a permanent posting in a second channel that controls the time itself. This posting integrates these instantaneous rates to come up with a current time. Temporary pauses, speed-ups and even loops are placed as postings in this second channel — and placed with considerable weight in order to have an effect — permanent changes are expressed in terms of changing the value of the rate channel by introducing a (temporary) posting.

This approach works well for both *how long...* and *22*. In the former, there is a very minimal description of what agents get created and destroyed that is played out by a time marker. This high-level sheet contains visual elements that themselves contain and run sheets. In *how long...* these sub-sheets are also reasonably simple (perhaps ten elements) but at the top level, there is a single time marker controlled by this two-level generic radial-basis channel. This time marker is set to move through the piece, over thirty minutes, but at several places time is temporarily paused, waiting for cues from the global choreographic tracking system; in three places it is permanently paused effectively waiting for permission to continue, and in two of these places there is a "hand cue" (one corresponding to a particularly difficult-to-capture rapid entrance, and one corresponding to the very end of the piece). In *22* the situation is rather more complex, because the world to be scripted is more complex — the piece incorporates, in addition to the main manipulation of video and geometry, textual elements, linear graphical elements and a rather complex system of cuing signals sent to the music. In this work the usefulness of a single time-marker begins to break down.

To see how Fluid might manage a move to multiple time-markers, we should first look at how visual elements control the time-marker of the sheet that they are on. There are two levels of control — a direct-drive "scripting" interface and

an indirect or "deferral" specification.

The direct-drive interface is extremely direct — instantaneous Python statements have permanent or temporary effects on the time marker through two time objects — *time* and *tempTime*. The following examples should convey the directness and the usefulness of the lines of code that can be constructed using these objects.

***time*.now** = 40 — sets the time to be 40; this is a "permanent" change and there is no compensating speedup or slowdown. Behind the scenes, a instantaneous change to the rate channel causes the jump. Hence, *time*.now = startOf("beginning") — goes back to the start of the visual element called "beginning"; *time*.now = *time*.now -40 — jumps back 40 units.

***tempTime*.now** = 40 — sets the time to be 40; this is a "temporary" change, made by adding a posting to the time position marker that lasts, by default around 10 seconds with weight 1. For finer control: *tempTime*.now = {*to*: 40, *duration*:5, *weight*:100, *bias*:1} — sets the time to 40, for around 5 seconds, with weight 100, with a window strongly biased towards the start, giving a percussive jump to the beginning and a long "ease-out". Hence, *tempTime*.now = {*to*:*time*.now-40, *weight*: lambda *t* : *t**100} — jumps back 40 with a channel window function that gets stronger as time goes on.

***time*.line** = {*to*:40, *over*:10} — animates time from wherever the marker currently happens to be, to 40 over 10 seconds. This is, again, a permanent change and is actually performed by modifying the rate channel. Hence: *time*.now=0; *time*.line = {*to*:40, *over*:10, *weight*:0.5} — jumps to the very edge of the sheet and begins to move forward to 40. *tempTime*.line can perform a similar end by writing to the position channel.

***time*.rate** = *time*.rate/2 — a (temporary) slowdown in the rate of time propagation that results in a permeant modification of duration. Finer control

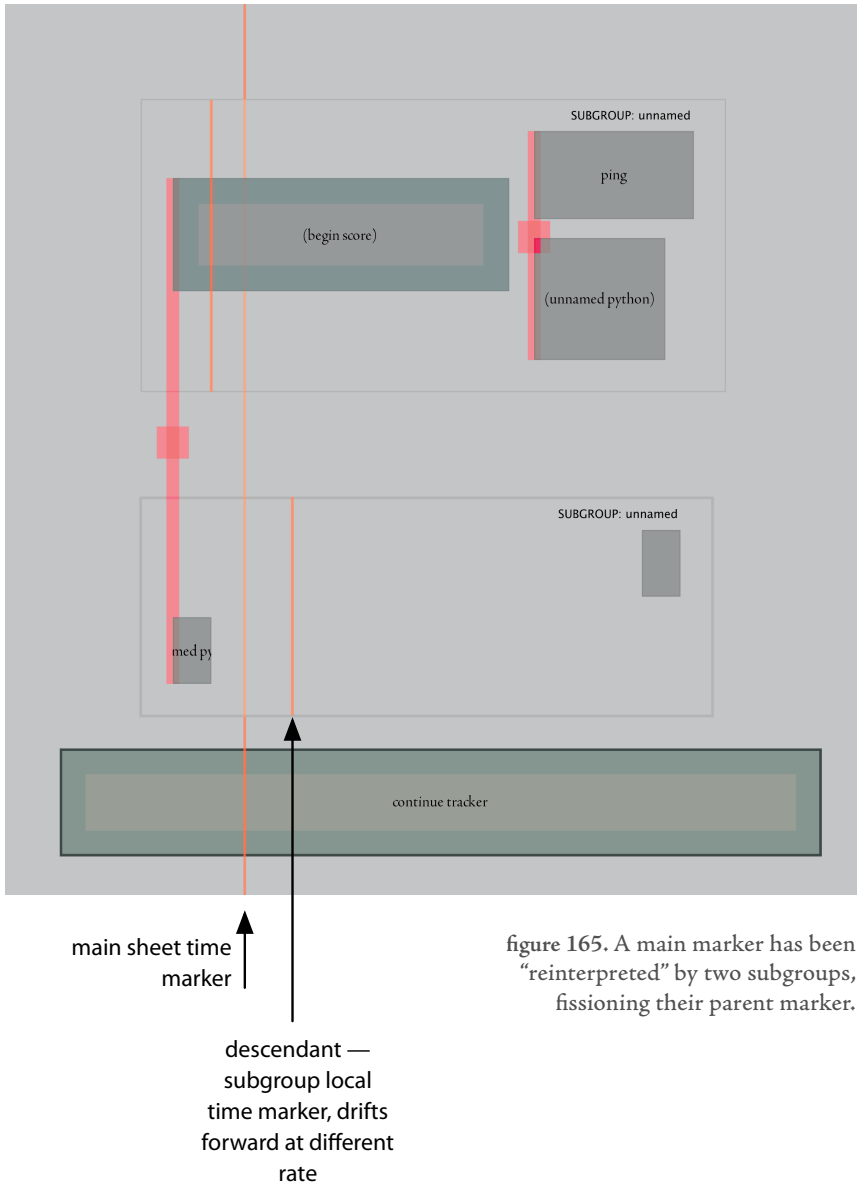


figure 165. A main marker has been "reinterpreted" by two subgroups, fissioning their parent marker.

similar to *tempTime*: for example, *time*.rate = {*to*: *time*.rate/2, *duration*:5, *weight*:10, *bias*:1}. This is achieved by posting to the rate channel. Likewise, *tempTime*.rate.

Both *time* and *tempTime* are ideal for scrambling around in an improvisation, helping one construct and blend loops of time by executing one-line statements by hand out of a pre-organized visual element's text editor. And, of course, there is nothing to stop visual elements from incorporating these statements inside their offered executable functions — in fact, this is predominantly how the score-follower and the hand triggers from the performer are integrated with the flow of time through *Imagery for Jeux Deux*. However, we can combine the above primitives to offer a more goal-directed manipulation of the flow of time through the sheet, based on discrete triggering events.

This simplest, general-purpose interface to an event supported directly by Fluid looks like:


```
interface DeferSpecification {

        DeferSpecification begin(double time);
        double passed(double time);
        void end(double time);
}
```

begin(…) informs the specification that we are about to start listening; passed(…) queries whether (1) or not (<1) the event has taken place, and is monotonic — that is, once passed(…) a specification is never not passed(…); and end(…) informs the object that the caller is no longer willing to wait. Additionally, begin(…) has the opportunity to return the specification that will be used for subsequent passed(…) and end(…) calls. Consider the case of a *next_floorwork* event, that is passed(…) when all of the dancers simultaneously go to the ground. This may happen more than once in a work — hence we register our interest in the next

such event using begin(...). This returns the object that we will query for passed(...) — allowing us to pass DeferSpecifications around globally, without passing around the means to create them afresh should we want some event's passing to remain local.

DeferSpecifications are composable using (fuzzy) boolean operators — specifications are commonly composed with an "or" operator with a time-out which limits how long a piece of code will wait. Inside the co-routine / resource framework we can bridge DeferSpecifications with standard co-routines by converting an increasing passed(...) to *progress*, a constant passed(...) to *continue*, a passed(...) = 1 to *stop* and a violation of monotonicity as *failure*, *page 156*. This bridge, of course, can be traversed in both directions.

For example, the basic set of fuzzy operations defined in : **K. Tanaka,** *An Introduction to Fuzzy Logic for Practical Applications,* Springer, 1996.

DeferSpecifications allow the creation of higher-level time-manipulation primitives. Most simply, we might pause at the start of a visual element until a specification has passed. It is most convenient to revisit the return-value of a visual element with respect to a Runner and write:


```
r = (start_executable, continue_executable, end_executable)

r = defer_until(r, specification)
```

The defer_* family of return-value decorators also understand 5-tuple as well as 3-tuple r values:

```
r = (start_executable, while_waiting_executable, transition_executable,
continue_executable, end_executable)
```

where the first is executed immediately, the second while the specification has not passed and the third as the transition from not-passed to passed.

More commonly used is a softer-pause:

```
r = defer_until(r, specification , fraction, smoothness, permanence)
```

this works the generic radial-basis channel structure a little harder, slowing down to arrive at a time that is "fraction" through this visual element when the specification becomes passed, but in any case never going beyond this fraction; the remaining parameters control the fading in and fading out of the window functions on the postings that achieve this, and how the control between permanent and temporary channels is partitioned out. Finally, we have:

r = defer_forever(*r*, *specification*)

that runs the underlying visual element should "specification" pass regardless of whether this visual element is still executing or not (in which case, it runs the "start-executable", "continue-executable" and "end-executable" in three execution cycles).

Each of these mechanisms — the defer_* method "decorators" and the *time* and *tempTime* direct objects are manipulating the time-marker for the whole sheet — the shared generic radial-basis channels ensure that competing ideas as to what this time should be are blended and faded in and out. However, the impact of changing the time is shared throughout the sheet — there is only one time-marker.

To allow two or more threads of action to drift in and out of sync in response to events would require the use of multiple sheets — which seems a less than perfect solution since it potentially deletes the spatial relationship (the "sync") which grounds both visual layouts. However, within the Diagram framework a better solution takes almost no effort to implement — we make the generic radial-basis channels for rate and time-position have context-local storage (for postings) where the context is given by the sheet grouping. Now we have an *ad hoc* but hierarchical structure in which to place postings, and whenever we manipulate the rate and position channels we get to choose at which level to place

the posting — local to the visual element, local to any group that the visual element is in, or "local" to the sheet (i.e. global).

At present the grouping visual elements make this choice for their children, by interposing a text preamble and post-amble to their children's code that causes their access to the time and rate channels to be at the group level. All other access is, by default, at the sheet level. This is the fundamental work that allows the reschedulable notations of the *parachute* / *accumulation* agents of *how long...*, the to-ing and fro-ing of *forest fire* and *stage machine* and the rhythmic traps of *22*'s scrubbing through video.

Now that we have the techniques required to fission and fuse time markers back together again, we can go further and build from time markers alternative ways of "executing" a sheet. The most developed are the graph-based structures. Inspired by the general usefulness of the pose-graph motor system and the task of creating a visualizer that allowed the visual assembly of pose-graphs from animation materials, we can create an executable graph structure using Fluid.

Of course, creating a directed, cyclic graph of connected visual elements is already supported in Fluid, *page 399*, so there is little visual programming work that needs to be done. However, we can form a graph Runner, by extending the time-marker Runner. This Runner moves through a graph structure and executes the contents of the visual elements that it encounters. The visual element return value *r* structure remains similar — scalars, lists or dictionaries of functions, methods or Updateables — but is extended with a graph return value *g* which is a dictionary of attributes that informs the graph runner that is executing the code how to act. At present there are three keys in this dictionary:

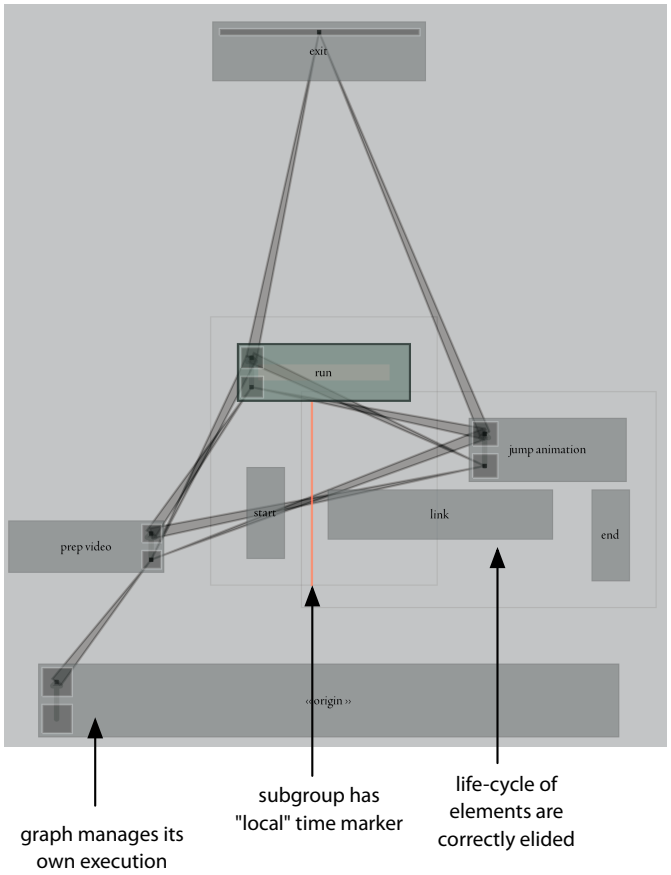


figure 166. A "graph selection" sheet. Time-markers can be local to particular visual elements — here a marker is local to the area underneath a visual element. Deactivations and subsequent reactivations between markers are elided. Note the different, but still meaningful, interpretation of the visual layout in this sheet.

duration — how long should this visual element execute for (in seconds); This value may be changed during the execution, but only when we get to the "end" of the visual element do the other keys become significant. *duration* controls the rate radial-basis channel in a two-level time-control similar to those used in time markers.

next — this refers to the visual element that we should go to next: it can be a visual element itself, the a name of an element or a regular expression over the name of the elements. It can also be the special elements *_nowhere* (to stop the runner completely), *_top* (refers to the spatially highest visual element connected to this element), *_bottom* (likewise). *next* may also be a dictionary that maps any of these elements to floating point values, in which case it is used to create an un-normalized probability distribution which is sampled from to create the next visual element.

fork — an optional list of alternative "*nexts*" that causes the graph runner to fork a copy of itself. By default these copies do (unlike time markers) re-execute the visual elements that they encounter — that is, they do not share a common parent (see *page 396*).

Even without the forking paths and the probabilistic *next* selection this structure is enough to visually create a pose-graph motor system from coarse-grained animations. To more fully exploit the visual potential of Fluid, in particular in pose-graph-like domains, we visualize the sweep of time across the graph visual element using a time-marker with a conventional runner, separate from the graph runner, local to the area beneath the visual element and sharing a common parent with all of the other local time markers.

All three of the main modeling and animation packages now have some kind of transition-graph-based character animation editor:

AliasWavefront's, Maya — http://www.aw.com . **Autodesk / Discreet**, Character Studio — http://www.discreet.com. **SoftImage's XSI** — http://www.softimage.com/products/behavior/v2/default.asp

While these are innovations in their product domains, it is clear that the user interface is still catching up with the expressive power of even computer game industries, and little of the substantially more advanced motion editing algorithms of the last 5 years' worth of SIGGRAPH have made it into the products yet, nor have the interfaces for motion editing reached the level of programmability taken for granted in other software domains.

This allows the visual creation not just of a pose-graph motor system, but of procedural processes on top of the pose-graph structure — that overlap, for example, with the beginnings and endings of animations. It is work for the future to try these ideas on a representational character, but it is my suspicion that Fluid will compare favorably with the recent interest in graph-based animation editing engines.

With the forking runners and probabilistic *next* selection it becomes possible to visually create, manipulate and improvise with the stochastic rhythmic cell generators of *Loops Score* and *how long....*

closing remarks

Fluid is a tool that has been made available to others. However, if it were not for the urge to generalize before specifying that bore it, it would be a completely specific, personal tool. Nor does it, as a work of engineering, independent of the agent-toolkit (and its lower-level graphics rendering, sound systems and network resources) offer anything sellable to the "non-programming" digital artist. Nor does it offer a segmentation of its structure into "modules" that are easily shared as currency between members of a "community". It seems to have few of the attributes of Max's social success.

However, while it does not compete with these tools, I nevertheless believe that part of its contribution is in that arena. That Fluid's database-like handling of its use-history, its self-reflexive monitoring capabilities, the way in which it enables the structuring of an environment to peer into the workings of a complex system, even the length to which it can stretch a dynamic language's syntax toward terse domain-specific tasks all have something to offer this domain, even in the absence of any widespread acceptance of the expressive need for text-based programming. It seems that should Fluid be augmented with a more

extensive, Python-based library fitting a problem domain shaped a little more traditionally, it would be a short and productive matter to turn this environment into something that could be productively used by other people. It does share some of the hallmarks of a truly learnable environment — it is non-modal, it reveals its flexibility gradually, it is itself open to inspection, and it is certainly adaptable to areas smaller than manipulating a full-blown agent toolkit.

Although Fluid clearly makes much of a visual layout's ability to be interpreted as a layout in time, even this does not in itself limit Fluid's applicability to the time-based arts. On a number of occasions, inside and outside Fluid, we have seen almost static ends achieved by dynamic means — for example, the "unrolling" of the interaction history of Fluid onto a single temporal score, *page 412*; the scoring of computer graphical processes in order to produce geometries from the action of constraints, *page 349*. In both cases the history of process, be it driven by mouse clicks or process scores, provides a temporal dimension that begs visualization. The multiple ways in which the history of these processes can be visualized, understood, and recast, all while remaining within the currency of Fluid — live-linked, text-based code — is an image that persists outside of the problem domains in which I have had the opportunity to use Fluid to date. Shades of such concerns abound in tools outside the context of interactive art — the forking "undo" stacks of image processing tools, the linear composition of mesh operations in modeling packages and the processing networks of image compositors, all mimic the surface of Fluid's relationship with its own use history. The utility and the potential for these techniques seems broader than the context that I have chosen for this chapter — the dominant interactive art tools — and hints at an "improvisatory" core of many other digital working practices.

At the same time, however, there are some aspects that would need to be reconstructed or strengthened: the development of additional layers is something that

ought to be achievable in Fluid directly, without recourse to the underlying host language; the version control system should be expanded to handle multiple simultaneous authorship on independent filesystems In general Fluid could benefit from being re-architected in such a way that it can truly be turned upon itself and made even more independent of the underlying toolkit — giving it a certain kind of pedagogical clarity. Yet in general, turning Fluid into a widely used tool is at least a less complex task than turning the agent toolkit into a widely used architecture, and perhaps this will offer an intermediate point in the distribution of the engineering contributions of this thesis.

But part of Fluid's contribution might be to question whether this arena — the single available art tool — should continue to be structured as it has been. Fluid would surely add more to the debate around the technical and conceptual bases for digital art making if there were in fact a debate raging.

With its acceptance of the expressive power of text-based programming, Fluid keeps company with what might be an emerging counter-trend toward the text-based — as evidenced by the languages Processing / Drawing By Numbers / Supercollider. Yet at the same time as these environments reject the problems and half-solutions of visual programming, they also reject the entire visual interface and run the danger of finding themselves ignorant of another emerging counter-trend in textual programming languages — one toward multiple, semi-textual views onto a program. The dominant art-making environments are all in danger as they grow of falling between the "professional" environments for large text-based programming projects — which have more support and manpower behind them — and the "easy-to-learn" visual environments. Fluid points toward a new class of hybrid environment and to a path out of this rapidly shrinking space.

Contributions & future work

This document began with a discussion of two disparate areas of human endeavor — contemporary choreographic practice and agent-based artificial intelligence. My work is located between these two areas, bridging them, exploiting each one for the other.

This thesis offers a productive critique of what I believe to be the two most persistent and widespread fantasies in digital art — its fascination with "emergence" and its reliance on the metaphors and techniques of "mapping". The first chapter of this work highlighted three axes, directions or trends in interactive art: a "conceptual" trend — moving from the hand-crafted and hand-established mapping relationship through a variety of machine-learning-based techniques; a "technical" trend — from the hardware-based pre-history of digital interactive art through fast, flexible software-based tools designed to allow a great many mapping relationships to be experimented with and tuned; and a "methodological" dimension — opposing the hand-crafted and thought-though mapping with the unexpected complexities of emergence. Beyond the termini of these axes I place AI's agent in general, and, of course, the kinds of agents constructed in this thesis in particular. The articulation of these directions was not intended

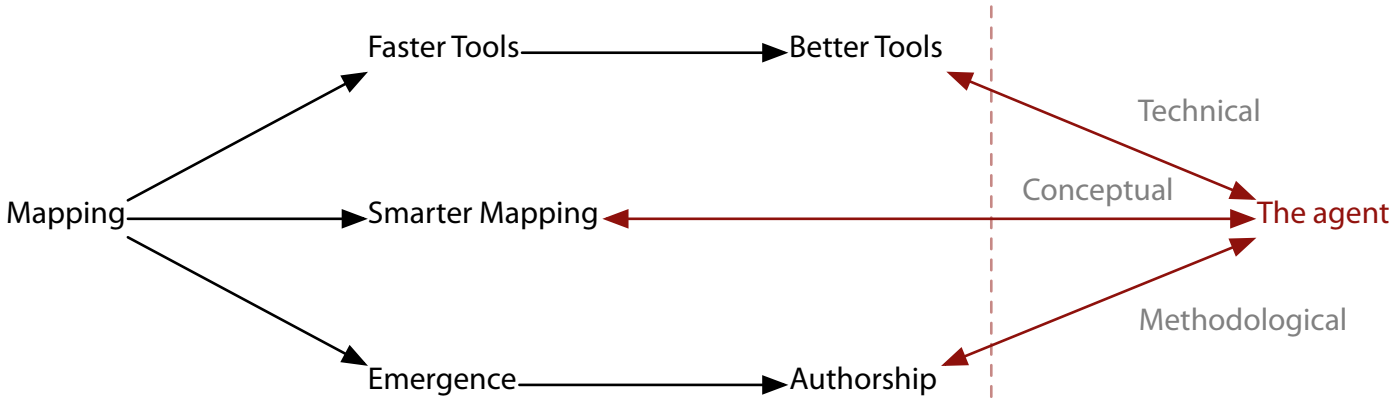


figure 167. The three axes introduced in chapter 1 (figures 3, 4 and 10 on pages 37, 37, 51 respectively.

to organize or predict the *quality* of the art produced by particular practitioners at particular positions on this illustration. Rather, it aimed to indicate the currents in the academic research and literature already present, and to create a frame through which my work could be viewed. In starting with the agent, I sought to start where I considered interactive art to be heading and moved in a counter direction. This counter-move is more than just a theoretical posture; rather I place the very interactivity, relevance and success of the artwork at stake in my ability to traverse these trends backwards toward constructing apparent interactive relationships, finding tools that ameliorate the difficulties of my approach, and techniques that help navigate the complexities of the agent-based.

Along the conceptual axis, I take the agent, in all of its complexities, and look for mechanisms to absorb the recent techniques for developing relationships (i.e."smart" mappings) and techniques to allow simple relationships, which may cut across many parts of the complex agent, to be authored. My carefully deployed learning techniques and perceptual structures can be seen in this light. Along the technical axis, I have developed tools, algorithms and frameworks that solve two classes of problems. Firstly they allow a "modular" approach to the construction of agents to be retained during the art-making process — I

have offered a variety of communicative blackboards and modifications and extensions to programming languages that share this goal. This modularity, if truly retained, escapes an otherwise perennial tension present at the heart of complex software systems between generic, broadly applicable frameworks that become hard to use because of this breadth and specific crafted solutions that are hard to *disassemble*, reuse technically or use conceptually to structure future works. Secondly, my technical contributions, my tool *Fluid* and my "glue systems", span many such "modules" specifically so that modifications and interactive potentials that cut across the whole agents can be constructed. Finally, along the methodological axis, both my tools and interstitial contributions speak to the problems of authoring systems that have emergent (that is, both unforeseeable and copious) consequences.

Much of this counter-move from agent to interaction, of course, has something to say about the limits of the present day field and forms the basis of this thesis's critique of the existing literature and approaches. Along each thread of this document — the development and maintenance of the potential of algorithmic systems, the envisaging and use of tools, and the deployment of "tactical formalisms" in a collaborative work — I show how an alternative metaphor, that of the agent, directly confronts what emergence and mapping ignore.

summary of technical contributions

I have offered specific implemented examples of the use of simple learning techniques to control the potential developed by complex agent-based systems — in the "stack" of emergence and authorial control of *Loops*, in the long-term learning database of *The Music Creatures* and in the agents of *how long...?*. I have been closely involved in developing an action-selection technique — the approach of the c5 agent toolkit — and have then extended it to **the diagram framework**, which radically expands the vocabulary available to the agent author for the

purposes of shaping and constraining the temporal patterns it creates. I have been closely involved in and learnt the lessons from creating complex agents in collaboration, identifying a set of problems and solutions that lie half-way between artificial intelligence and software engineering, leading to **the context tree**. I have identified two reusable design patterns for the creation of agent's perception systems and proved them in a wide range of particular instantiations — **the b-tracker framework** and **the distance mapping algorithm**. Generalizations and re-specifications abound in my work — I have created the **generic pose-graph representation** that allows the rapid creation of agents with a wide range of bodies and source material; my **generic radial-basis channels**, my **language interventions** and the **context tree** all decouple elements of the agent, allowing them to be quickly repurposed and recast. I have surrounded these complex assemblages in a set of tools and representations that allow them to not just be demonstrated but integrated into ongoing art practice; these tools are collected in **Fluid**. I have created graphical rendering techniques and intermediate body representations for agents — **the re-projection renderers** — that open the possibilities of ambiguous visual forms back out to the agent itself.

The nature of these technical contributions needs some careful consideration in two ways. Firstly, for the purposes of both a dissertation and a broader academic context it is important to consider what it is that they actually *contribute*. Secondly, for the purposes of considering how these techniques might be perpetuated in intellectual discourse outside this document, we need to consider *how* it is they are structured.

On the one hand it is my belief that they do little, if anything, to extend the *theoretical* reach of artificial intelligence as measured by the standard data-sets and conventional micro-worlds of machine learning. However, I believe each of them makes significant and original contributions to the *practical* reach of the agent. This disconnect between the directly quantifiable and the pragmatically

useful is independent of the predominant art context of my work. It is not that my decision to deploy my technical contributions in the service of making artworks somehow thwarts AI's methods for evaluating contributions. Rather, it is due to both the lack of quantification inherent in the field — particularly when close to large-scale, heterogeneous AI systems and even more so when discussing design approaches and structuring frameworks rather than specific algorithms.

My technical contributions appear as both general structure and multiple, specific instances; often the specific instances are present in, perhaps even dominant in, specific technical fields. Each contribution possesses this double nature: the b-tracker framework has in a very real sense no algorithmic core, it is a framework, a structuring template that gets populated based on the task at hand. In doing so, the resulting system may recapitulate computer music's score follower or computer vision's tracking algorithms as well as providing novel hypothesis trackers that are hybrids or just plain different. And while in these cases, my resulting "implementation" (which is nothing more than a particular specification of the variable parts of the b-tracker framework) work well in these areas, any evaluation of the technical competencies of these particular instances of the frameworks do not quite get at the heart of the quality structuring contribution itself. And despite this algorithmic displacement, the b-tracker framework refers not just to a chapter of this document but also to specific, singular implementation, a specific body of code, that is present enough to also be the site of fixed visualization tools and offer interactive surfaces and abstraction barriers up to other modules inside the agents that I create. The distance mapping algorithm generalizes and reinterprets statistical techniques such as multi-dimensional scaling — techniques that have been around for decades that I have no claim over — but recasts them in such away that they have broad use to the problems that interactive artists face. The use (e.g. of multi-dimensional scaling approaches to a broad range of mapping problems) is novel,

but it is the recasting itself (e.g. the articulation of the general problem and solution and a constructive proof that there exists an implementation that survives this generality) that I believe is lasting and significant.

Thus I am often left seeking, for the purposes of locating my contribution, mechanisms of "proof" of the technical contributions not at the level of specific implementations but at the level of the framework. Towards this end, I can see three lines of reasoning. Firstly, one argument exploits the range of artworks — installations, compositions, interactions, works for live theater — together with works with an explicit and clear biological referent — *Dobie*, *alphaWolf* — that have used these technical underpinnings as a step towards securing the quality of these techniques. Secondly, while the argument that many of these artworks were constructed quickly (in the case of *Loops* in particular) may appear structurally unsound, I believe that there are certain thresholds of speed and facility that, when crossed, allow new kinds of artworks to be created, and new kinds of collaborations to succeed. That the score follower for *Imagery for Jeux Deux* that in the original conception of the work was thought to be unnecessary was constructed and tested in an afternoon and ultimately became fundamental to the interaction of the piece, and that the recapitulations of *triangle* were constructed during a break in rehearsal, point towards the crossing of such qualitative thresholds.

Thirdly, I offer the range of researchers working *with* rather than *on* my technical ideas as an argument for the strength of their contribution. They have had widely differing concerns and agendas and many have constructed their own work around the agent-toolkit that incorporates my techniques and code. This second layer of validation offers an alternative plane of collaboration — one where I assume, as I do in my artworks, responsibility for some of the technical path, but reject responsibility for the artifactual destination. In this fashion, one might point to the use of the pose-graph motor system to control robots (hy-

for a use of the pose-graph in robotics: **C. Breazeal, D. Buchsbaum, J. Gray, D. Gatenby, B. Blumberg.** *Learning from and about Others: Towards Using Imitation to Bootstrap the Social Understanding of Others by Robots*, in : L. Rocha and F. Almedia e Costa (eds.), *Artificial Life* 11 (1-2). 2005.

for a use of the contex-tree in simulation theory: **J. Gray**, *Goal and Action Inference for Helpful Robots Using Self as Simulator*, Masters of Science thesis, Media Arts and Sciences. MIT. 2004.

bridizing computer animation techniques with expressive robotic control); or the use of the context tree to create agents that simulate other agents (the embedding of a "virtual" agent within another by using the hierarchical context). Since I comprehensively lack the skills or opportunity to work in robotics or the background in the simulation theories of cognitive modeling I cannot retroactively claim these tasks as motivation for the pose-graph motor system or the context tree. Since these extended uses remain within the realm of messy, large-scale AI research there are no critical results on standard data-sets, no disprovable predictions strong beyond the number of "free-parameters" that my techniques possess, that I can borrow for the purposes conclusive proof. Instead I might claim what might be large-scale AI's only equivalent of the scientific standard of replication — a relatively independent reuse of AI design ideas and implementation.

But in a broader context I refuse to shy away from these harder-to-evaluate approaches and framings not simply because of the practical utility that they offer me in my varied collaborations, the practical fluidity that they allow in my work or the thrill of seeing them adopted, expanded upon and reused in domains distant from my own opportunities. Rather, I believe that such frameworks, such reframings of algorithms and data-structures, are precisely the research project that both artificial intelligence and digital art require at this very time. This opinion, in a AI context, is sufficiently well stated elsewhere — in the work of Minsky and others. In a digital art context it bears restating. Having gone beyond a simple technical facility, the speed with which well-known and well-worked-out algorithms may be either coded (in a "text-based" practice) or called-up (in a more typical "visual-programming" environment), the research vista, the methodological frontier that lies beyond the simple mining of the flexibility and speed of computers, is to find the structures and frameworks that allow the understanding, generalization and re-recognition of common algorithms in a new light of digital art. That many of my contributions are in the

interstices of code-practice indicates a recapitulation of my emergence and authorship counter-tension at a different level of practice. That the central technical problems faced by digital art (and artificial intelligence) might be shifting from the finding of powerful algorithms and data-structures to figuring out how to deploy them given that they already exist.

Without the technical contributions my artworks are inconceivable, in all senses of the word: they could not be articulated, started, or finished. Without the artworks, these technical contributions would be unmotivated, unproven, unfulfilled. The techniques are neither directly present in the surface of the artworks nor vanish completely from them, no more than the style in which these frameworks are constructed is independent of the art that I have made and provoked.

The artworks presented here are more than the techniques behind them and, simultaneously, the techniques that I have developed here are not wholly consumed by the artworks that exploit them. Indeed, one crucial indication of the technical success of an artwork is tied up in this very attitude. As artistic conditions (collaborations, available materials and interactions) provoke technical contributions that are (by personal preference, and by practical necessity) flexible, generic, or modular in nature this effort is satisfying and worthwhile when an artwork escapes the ability to think through the potential field generated by the technique, finding the utterly unexpected deep within this field. Simultaneously the technical approaches are in themselves satisfying and worthwhile when they remain unexhausted by the pieces that they permit, pointing towards unexplored vistas after the works themselves have been "finished".

Again, this criterion for success speaks also of the methodological importance for the modular, reusable and the generic in my work and goes some way to legitimize, at a technical level, the apparent indirection inherent in the agent-

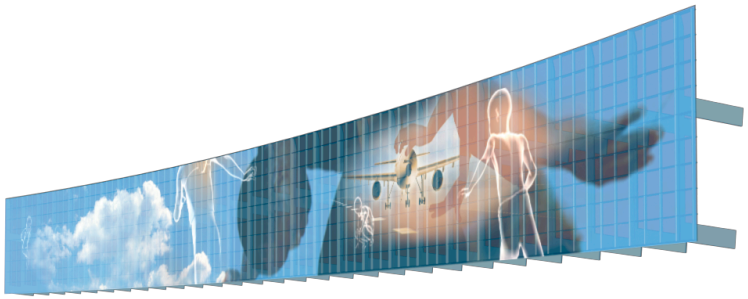


figure 168. *Horizon* installation pre-visualization.

based. Rather than being a complicating and eccentric place to begin work, perhaps my agent metaphor and practice offers a vastly shortened route to this technical territory.

The future work of this thesis, my future artwork, is at the very least to continue mining the potential of the technical contributions of this thesis while using new artworks to, in turn, provoke new developments. Rather than discuss in abstract terms where my techniques might lead my art and where my art might lead my techniques, I would rather discuss two concrete, commissioned artworks that I believe illustrate and extend the two main threads of my work.

Horizon, 2005-7

figure 169. *Horizon* sequence pre-visualization — *forest, hide and seek.*

The first thread can be drawn through parts of *alphaWolf*, *Dobie* aspects of *The Music Creatures* and ultimately *22* — agents with complex bodies and large stores of animation material which require complex blending, layering and manipulation and yet have a strong, representational, figurative requirement. This thread leads to a project entitled *Horizon*, commissioned for the main hall of the forthcoming international concourse-F at Atlanta's Hartsfield-Jackson Airport.

On a 360-by-30-foot custom-made LED display, this permanent artwork will finally present the opportunity to make a piece that runs live, without repetition, on an "architectural" time-scale. It will be the largest live permanent digital artwork to date.

Our imagery, which inverts the scale of child and airport, will draw on extensive motion capture libraries of children's motion — playing hide-and-seek, manipulating the skyline of Atlanta, operating the mechanics of the airport. These game-like forms will be played out by characters assuming the figuration of children, but like *22*, this figuration is not the stable affair of computer games and special effects, but rather the unstable, shifting forms of childhood dreams.

figure 170. *Horizon* pre-visualization — view from baggage claim.

To create the choreography of these multi-agent games, I expect to use and extend the Diagram framework — extending its ability to coordinate multiple action to the choreography of multiple agents — and a pose-graph motor system — extended specifically to perform well over both extremely large libraries of animations and variable bodies.

The use of the agent metaphor in structuring this work allows the imagery to be open to extension — incorporating events from the changing life of the city as the years go by — and responsive to external influences: the flux of passengers, the time of day, the weather and the season. This couples the artwork to the environment of the airport, and the time-scale of the installation setting, in a way that a finished film could not hope to achieve. The techniques developed in this thesis that allow the rapid creation, tuning and automatic "balancing" of agents will be fundamental in constructing an artwork on an display-scale setting that inherently cannot be prototyped accurately. The piece is set to open with the completion of the terminal building in 2007.

The Enlightenment, 2005-6

The second thread through my work can be traced through *Loops*, *The Music Creatures*, *Loops Score* and parts of *how long...?* and *Imagery for Jeux Deux* — the visual depiction of both generative and analytic musical processes, the notational and the enumerative. This thread leads to a project entitled *The Enlightenment*, commissioned as part of the 40th season of the Lincoln Center for Performing Art's *Mostly Mozart* festival to celebrate the 250th anniversary of Mozart's birth. Ten high-resolution displays distributed down the length of the front of Avery Fisher Hall will make this work the highest-resolution live digital artwork to date, and truly allow the creation of digital imagery that can be observed at a range of distances. As in *The Music Creatures*, each screen will house a sound-producing agent communicating with its neighbors; as in *Loops Score*

figure 171. *The Enlightenment* installation pre-visualization.

the "source material" for the agents will be fixed in advance; as in *how long...?* and *Imagery for Jeux Deux* the physicality of performance will brush up against an aesthetics of notation and description.

But this new artwork presents the opportunity to unite the aesthetics of effort, intention and transience with the concerns of truly "human-level" music — the central issue arguably dodged by both my musical works to date. The source material will be the last 30 measures, the dizzying display of five-part invertible counterpoint in Mozart's Symphony No. 41 "The Jupiter". And over an constant installation period of three months the agents will recompose, recast and rediscover the unities and possibilities of the material Mozart deploys in this passage, exploiting a library of video and sound captured from performances of the work. Both acoustically and visually, *The Enlightenment* will be patterned on the scale of the hour, the day, the week and the month. In some senses it will be a three month long composition.

While my previous works might have used unconventional means (the agent), hardware (motion capture) and tools (Fluid) they have not yet enabled access to non-traditional audiences, scopes or venues. These new works place the artificially intelligent agent not just in new art contexts but in unexpected contexts for digital art. The "openness" of my open forms, the enticing time-scales hinted at by *Loops* and *The Music Creatures* and the surplus potential evident in the rehearsals of my works for dance theater seem to demand a move away from the traditional gallery installation or the confines of a fixed duration performance. This move comes with considerable challenges. While I have had success, as far as my works have led, in creating pieces that are far beyond what one can *think-through* as an artist by developing techniques that allow the navigation of open interaction, can my open forms remain open while I construct pieces that are far longer than any single rehearsal, far longer than I can possibly *work-through*,

even once? Both *Horizon* and *The Enlightenment* stand on a new threshold for digital art.

the experience of the agent

The agent metaphor offers the opportunity to reframe the problems of algorithmic art in terms that meet the computational sensibility of contemporary choreography that I identified in my opening chapter. My most recent collaborations with choreographers have resulted in what I believe is the most sustained example to date of a dialogue centered on this computational sensibility. I have constructed networks of computational representations that are complex enough to yield surprising forms, material and relationships, and controllable enough to allow the unexpected to be assured. I have sought the technologies required to bring these forms to human movement and human movement to these agents. By careful formulation and generalization of learning, programming and visualization techniques, I offer the extensions, frameworks and tools that this agent metaphor needs in order to be more than just an organizing principle. My technical contributions alleviate the difficulties posed while exploiting the opportunities offered by the indirection inherent in the agent-based.

This indirection is the aesthetic center of my body of work; my agents are autonomous enough, intelligent enough, to maintain a dynamic disequilibrium with their environments. Because of this relationship to their setting, I believe my agents embody an aesthetics of intention, effort and transience unobtainable by more "direct" means. I offer new "open forms" that are solutions to the paradox of "scoring" an autonomous system.

A number of times throughout this thesis I have documented this aesthetics that I believe is attainable from this agent-based practice and indicated, or at

least hinted at, the moments when it truly comes to the fore in my work: the musical renegotiations of *The Music Creatures* — the error-prone echos and the observable attempts by the creatures to traffic musical material, the intentionality of *network*, the broken clock-like movement of *tile*; the endless recomposing of *Loops*; the quirky, fragile rhythmic material of *Loops Score* that produces material that is unexpected yet somehow inevitable; the layered excesses and inadequacies of *how long...* — the goal of *triangle*, the fleeting shifts of *memory score*; the moments when, having drifted apart, the imagery, movement and narrative of *22* collide. All of these aspects, all of these moments, seem at their core to be both technical and aesthetic consequences of the construction of autonomous agents. But what precisely links them all and, more importantly, but perhaps even more speculatively, what accounts for my personal attraction to these phenomena? What are the experiential intentions of my work?

It remains impossible and impractical for me to find a definitive statement on these matters, yet my sense that there remains a stable and common core to these "aesthetic moments" that escapes merely the technical relationships between the works begs some attempt at explanation.

One route that offers some promise in this direction is to return to the opening chapter's brief discussion of the status of the "formal systems" developed in both contemporary choreography and, by influence and extension, my work. In these fields I have referred to these approaches as deploying "tactical formalisms", that is, formal techniques that arrive methodologically prior to the discovery of their consequences, and perhaps even their natures, that are protected from interrogation during their articulation and given privileged status during the mining of the potential that they develop. These formalisms are deprived of any totalizing wider role by an equal, but opposite, tendency to question and undermine these very formal ideas between works, between explorations. I believe that in my work my set of experiential goals are actualized when these formal approaches

come into oblique contact with a seemingly opposed set of concerns — that of realism.

At the simplest level the search for this tension between the formal, the autonomous, and the realist might explain my continual return to human materials — be it human motion — *Loops*, *how long...* — the fundamentals of human music (rhythm, timbre) — *The Music Creatures* — the human voice and language — *Loops Score*. These, often explicitly, counter-balance my agent-based formal indirections. Alternatively, perhaps it predicts my interest in developing "ambiguous" computer graphical techniques that can transit between the realist and the abstracted — most notably in *22*. Perhaps it suggests a deeper reason for my sincere, but admittedly uneasy, engagement with biological referents — in *The Music Creatures* specifically and in the the c5 agent toolkit in general. But these aspects again draw us back to a more technical level than this discussion was intended to take, or at least one internal to the work rather than external, and does not explain the considerable autonomy I give to my formal approaches.

There is a long tradition, however, of attempting to rehabilitate the realist project, freeing it from what, for example, Paul Klee called the "painfully precise investigation of appearances", of transferring concern from appearance to *functioning* — from "anatomy to physiology". Perhaps one might identify in my work, or at least in my aesthetic intentions, couched in and supported by the presence of the human, in motion and in sound, a realism that lies one further step removed from the optical than this physiology.

Quotes are from
P. Klee, *Paul Klee Notebooks, Volume 1 : The Thinking Eye*,
J. Spiller, (ed.) George Wittenborn, NY, 1961.

Perhaps this occulted presence of nature accounts for the longevity of contemporary choreography far from the "hook" of narrative, mimesis or emotion. Perhaps there is a point of contact here with the longevity of the Oulipo literary group's "formal" or "axiomatic" investigations. Returning to my first chapter's concerns with the interplay between figuration and abstraction, perhaps this is what Francis Bacon isolates as the tension between order and representation, between one level and another:

> "One of the reasons I don't like abstract painting, or why it doesn't interest me, is that I think painting is a duality, and that abstract painting is an entirely aesthetic thing. It always remains on one level. It is only really interested in the beauty of its patterns or its shapes. [...] I think that great art is deeply ordered. Even if within the order there may be enormously instinctive and accidental things, nevertheless I think that they come out of a desire for ordering and for returning fact onto the nervous system in a more violent way."

from pp. 58-9, in **D. Sylvester,** *Interviews with Francis Bacon*, Thames-Hudson, 1975.

Perhaps too this is what musicologist Michel Imberty, searching for a naturalistic foundation for music, and by my extension the "temporal arts", has encountered:

> All [music's] temporal substance is nourished by our way of being in the world; that is, in our time, out culture, our perceptions, our bodies, our emotions, and our sentiments. It is not communication but a representation of our ability to communicate, it is a stylized game for our opening to the world, it is communication without an object to communicate. In this sense, music is indeed, the symbol of our fundamental relation to time, life, and death.

M. Imberty, *The Questions of Innate Competencies in Musical Communication*, in: **N.L. Wallin, B. Merker, and S. Brown,** *The Origins of Music*, MIT Press, Cambridge MA, 1999

To give a name to the place where the anatomical, the physiological, and the formal intersect, I propose that this at this core is an *infra-realism*, in which the audience (and the artist) recognizes not an precisely analogous or parallel mode of functioning but the very functioning-like aspect of bodies and their imbalances with the world. A captivating recognition that *Loops Score* does not resonate with the rules of language *per se* but draws its strength from a parallel recognition that language incorporates an alien mechanic current; that *The Music Creatures* point towards a formal quality possessed not just by the algorithmic

For a selection of essays by members of the Oulipo: **W. F. Motte Jr** (trans., ed.), *Oulipo: a primer of potential literature*, University of Nebraska Press, 1986.

For a more encyclopedic introduction: **H. Mathews, A. Brotchie,** *The Oulipo Compendium,* Atlas Press, 1998.

machinations of western art music but even the song of birds; that in *how long...* my images point to the arbitrary yet necessary core of both choreography and human movement in general. Perhaps this is the thread that ties my technical relationship to motion-capture, through my conceptual choice to begin with the autonomous agent rather than a "more direct" interactive relationships, to my deployment of such localized formal systems, my preference for imbalance and transience, the aesthetics of my visual imagery, and finally all the way through to my experiential intentions.

To gain access to this territory within the context of interactive digital art I have had to abandon the conventional points of origin, the standard tools, and the traditional methodologies and create my method, technique and structuring concepts afresh. I have sought the collaboration of a diverse range of artists and AI researchers. Although I claim that the the technical contributions are strong and the artworks successful, and I believe that I have proved this as much as it can be proven by applying these techniques to an extremely diverse range of works, my firmest belief is that the technical contributions are of interest to digital artists who are both willing and able to interrogate their own technical practices. In general I hope that my thesis expands digital art's working practices — changing the starting points of pieces, the methods and in particular the tools used on the journey and the possibilities open to artists. I hope that my thesis, like my works, indicates and develops a field of previously unknown potential and demonstrates techniques for navigating this field.

Works cited

Adobe Systems Inc., PDF Reference, 5th edition. Available online at: http://partners.adobe.com/public/developer/pdf/index_reference.html

Adobe Systems Inc., The Adobe Postscript Reference Manual, 3rd edition. Available online at: http://partners.adobe.com/public/developer/ps/index_specs.html

Adobe Systems Inc., Illustrator, software product — http://www.adobe.com/illustrator

AliasWavefront, Maya, software product — http://www.aw.com.

Apple Computer, Inc., Keynote, software product — http:// www.apple.com/keynote

Aquinas, T. Summa Theologica.

Autodesk / Discreet, 3D Studio Max, software product — http://www.discreet.com/3dsmax

Autodesk / Discreet, Character Studio, software product — http://www.discreet.com

The **Apache project**, Excalibur project, open source software, http://excalibur.apache.org/

Apache Jarkarta, The HiveMind project http://jakarta.apache.org/hivemind/ioc.html

Aristotle, Poetics (1449b-1450a).

AspectJ project, open source software, http://eclipse.org/aspectj

Blumberg, B., M. Downie, Y. Ivanov, M. Berlin, M. P. Johnson, B. Tomlinson, Integrated learning for interactive synthetic characters. SIGGRAPH 2002. Proceedings of the 29th

Badler, N. , C. Phillips, B. Webber, Simulating Humans: Computer Graphics, Animation, and Control. Oxford University Press, 1993.

Bates, J. , A. B. Loyall, W. Scott Reilly, An Architecture for Action, Emotion and Social Behavior. Technical Report CMU-CS-92-144, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. May 1992.

Bates, J. The Nature of Character in Interactive Worlds and The Oz Project, Technical Report CMU-CS-92-200, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. October 1992.

Bedau, M. Artificial Life VII: Looking Backward, Looking Forward. Artificial Life 6: 261-264 (2000)

Belongie, S. , J. Malik, J. Puzicha, Shape Matching and Object Recognition Using Shape Contexts, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (4), 2002.

Bencina, R. The Metasurface — Applying Natural Neighbour Interpolation to Two-to-Many Mapping, Proceedings of the 2005 conference on New Interfaces for Musical Expression, Vancouver, Canada.

Benjamin, W. On the Mimetic Faculty, In: Reflections, P. Demetz (ed.), Harvest / HBJ, 1978.

Bezdek, J. C. Fuzzy Mathematics in Pattern Classification. Ithaca, NY: Cornell University; 1973.

Bishop, C. M. Neural networks for pattern recognition, Claredon Press, Oxford. 1995.

Blackwell, A. F. , K. N. Whitley, J. Good, M. Petre, Cognitive Factors in Programming with Diagrams. Artificial Intelligence Review 15: 95-114, 2001.

Blumberg, B. M. Old Tricks, New Dogs: Ethology and Interactive Creatures. Media Laboratory. PhD Thesis. MIT. 1996.

Blumberg, B. Action-selection in hamsterdam: lessons from ethology. Proceedings of the third international conference on Simulation of adaptive behavior, Brighton UK. 1994.

Blumberg, B. Swamped! Using plush toys to direct autonomous animated characters. Proceedings of SIGGRAPH 98: conference abstracts and applications., ACM Press, 1998.

Blumberg, B. (void*): A Cast of Characters. Proceedings of SIGGRAPH 99: conference abstracts and applications. ACM Press, 1999.

Boulanger, R. (ed.) The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing and Programming, MIT Press, 2000.

Boulez, P. Le Pays Fertile: Paul Klee, Editions Gallimards, Paris, 1989.

Breazeal, C. Sociable Machines: Expressive Social Exchange Between Robot and Human. Artificial Intelligence Laboratory. Cambridge, MA, MIT. 2000.

Breazeal, C., D. Buchsbaum, J. Gray, D. Gatenby, B. Blumberg, Learning From and About Others: Towards Using Imitation to Bootstrap the Social Understanding of Others by Robots. Artificial Life, 11 (1), 2005.

Breazeal, C. A Motivational System for Regulating Human-Robot Interaction, Proceedings of AAAI-98.

Brooks, R. Intelligence without Reason, MIT AI Lab Memo 1293.

Brooks, R. , C. Breazeal , R. Irie, C. C. Kemp, M. Marjanovic, Brian Scassellati, Matthew M. Williamson, Alternative Essences of Inteligence, Proceedings of the American Association of Artificial Intelligence 1998, AAAI Press, CA.

Brooks, R. Elephants don't play chess, Robotics and Autonomous Systems 6 (1990) 3-15,

Brooks, R. Intelligence without Representation, Artificial Intelligence, 47 (1991) 139-159.

Brooks, R. Challenges for Complete Creature Architectures, In Proceedings of the first international conference on simulation of adaptive behavior, Paris, France, 1991.

Brooks, R. Elephants don't play chess, Robotics and Autonomous Systems 6 (1990) 3-15,

Brooks, R. How To Build Complete Creatures Rather Than Isolated Cognitive Simulators, in: Architectures for Intelligence, K. VanLehn (ed), Erlbaum, Hillsdale, NJ, Fall 1989, pp. 225-239.

Buckland, M. Programming Game AI by Example, Wordware Publishing , 2004.

Burke, R. , D. Isla, M. Downie, Y. Ivanov, B. Blumberg, CreatureSmarts: The Art and Architecture of a Virtual Brain. Proceedings of the Game Developers Conference: 147-166. 2001.

Burtner, M. The Metasaxophone: concept, implementation, and mapping strategies for a new computer music instrument. Organised Sound 7(2): 2002.

Buschmann, F. , R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System Of Patterns. West Sussex, England: John Wiley & Sons Ltd., 1996.

Calvert, T. W. , A. Bruderlin, S. Mah, T. Schiphorst, C. Welman, The Evolution of an interface for choreographers. Interchi'93 — ACM Press, April 1993.

Cage, J. Notations, Something Else Press, 1969.

Cage, J. Silence, Wesleyan University Press, 1961.

Cassell, J. , H. H Vilhjálmsson, T. Bickmore, BEAT: The Behavior Expression Animation Toolkit. In: SIGGRAPH 2001, International Conference on Computer Graphics and Interactive Techniques. Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM, 2001.

Cassell, J. , T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. Vilhjálmsson, H. Yan, Embodiment in Conversational Interfaces: Rea. Proceedings of the CHI'99 Conference. 1999.

Cemgil, A. T. Bayesian Music Transcription, PhD dissertation, Radboud University of Nijmegen, 2004.

Chadabe, J. The Limitations of Mapping as a Structural Descriptive in Electronic Instruments. Proceedings of New Instruments For Musical Expression, Dublin.

Connell, J. H. A colony architecture for an artificial creature, MIT Ph.D. Thesis in Electrical Engineering and Computer Science, MIT AI Lab Tech Report 1151 (June 1989).

Cont , A. , T. Coduys , C. Henry, Real-time Gesture Mapping in Pd Environment using Neural Networks. Proceedings of the 2004 Conference on New Interfaces for Musical Expression.

Cook, P. Principles for Designing Computer Music Controllers, ACM CHI Workshop in New Interfaces for Musical Expression (NIME), Seattle, April, 2001.

Cooper, S., W. Dann, R. Pausch Teaching Objects-first in Introductory Computer Science, Proceedings of SIGCSE 2003

Copeland, R. Merce Cunningham: the Modernizing of Modern Dance, Routledge, 2004.

Cope, D. Virtual Music, Computer Synthesis of Musical Style, MIT Press, Cambridge MA. 2001.

Cunningham, M., D. Vaughan, Other animals: Drawings and Journals. Aperture, New York, 2002.

Cycling'74 / Ircam, Max/MSP/Jitter, software product — http://www.cycling74.com

Cytowic, R. E. Synesthesia: A Unity of the Senses, New York: Springer-Verlag, 1989.

Dannenberg, R. Dynamic Programming for Interactive Music Systems, in Readings in Music and Artificial Intelligence, E. R. Miranda, (ed.), Contemporary Music Studies series, Vol. 20, Harwood Academic Publishers, 2000.

Debevec, P. , C. J. Taylor, J. Malik, Modeling and Rendering Architecture from Photographs:A hybrid geometry- and image-based approach. In: SIGGRAPH 1996 International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM, 1996

deLahunta, S. Dialogues on Motion Capture, Proceedings of IDAT 1999.

deLahunta, S. Virtual Reality and Performance, PAJ: A Journal of Performance and Art 24.1 (2002) 105-114

Deleuze, G. Francis Bacon: the logic of sensation, D. W. Smith , T. Conley (trans), University of Minnesota Press, 2003.

Deloura, M. Game Programming Gems 1-3, Charles River Media, Cambridge, MA, 2000-2.

Desain, P., H. Honing. Letter to the editor: the mins of Max. Computer Music Journal, 17(2). 1993.

Dobrian, C. , F. Bevilacqua, Gestural Control of Music Using the Vicon 8 Motion Capture System. Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Downie, M. behavior, animation, music: the music and movement of synthetic characters. S. M. Thesis, January, 2000.

Eberly, D. H. 3D Game Engine Architecture : Engineering Real-Time Applications with Wild Magic, Morgan Kaufmann, 2004.

Eberly, D.H. 3D Game Engine Design : A Practical Approach to Real-Time Computer Graphics, Morgan Kaufman, 2000.

Eberly, D. H. Wild Magic, graphics and algorithms library: http://www.geometrictools.com

Eck. D., J. Schmidhuber, Finding Temporal Structure in Music: Blues Improvisation with LSTM Recurrent Networks. H. Boulard, editor, Neural Networks for Signal Processing XII, Proceedings of the 2002 IEEE Workshop. 747{756, New York, IEEE, 2002.

Eco, U. The Open Work, A. Cancogni (trans.) Harvard, 1999.

Elrad, T. , R. E. Filman, A. Bader, Aspect Oriented Programming : Introduction, Communications of the ACM, October 2001.

Evangelista, G. Flexible Wavelets for Music Signal Processing. Journal of New Music Research, 30 (1). 2001.

Farbood, M. , B. Schoner. Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains, Proceedings of International Computer Music Conference. Havana, Cuba. 2001.

Fernández-Madrigal, J-A., J. González, Multihierarchical Graph Search. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (1), January 2002.

Forgy, C.L. , Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence, 19, 1982

Forsythe, W. quoted in T. Ozaki, (P. Vigilio trans.), An Interview with William Forsythe. Previously available online: http://frankfurt-ballett.de/articles2.html however, since the dissolution of the Ballett Frankfurt, this paper is currently available through the internet archive: http://web.archive.org

Forsythe, W. quoted in T. Ozaki, (P. Vigilio trans.), An Interview with William Forsythe. (availability as above).

Friedman, D. P. , M. Wand, C. T. Hayes, Essentials of Programming Languages. MIT Press, 2001.

Fry, B., C. Reas: Processing, programming language, http://www.processing.org

Fry, C., M. Plusch, Water, programming language, http://www.waterlanguage.org/

Gamma, E. , R. Helm, R. Johnson, J. Vlissides, Design Patterns, Elements of reusable object oriented software. Addison-Wesley, 1995.

Garnett, G. , C. Goudeseune. Performance Factors in Control of High-Dimensional Spaces. Proceedings of the International Conference

on Computer Music. 1999. San Francisco, International Computer Music Association.

Gaye, L. , R. Mazé, L. E. Holmquist, Sonic City: The Urban Environment as a Musical Interface, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Glassner, A. Principles of Digital Image Synthesis. Morgan Kaufmann Publishers, Inc., San Francisco, 1995.

Glinert, E. P. Visual Programming Environments: Applications and Issues & Paradigms and Systems. IEEE Computer Society Press, 1990.

Goldberg, A. Improv: A System for Real-Time Animation of Behavior-Based Interactive Synthetic Actors. In Lecture Notes in Artificial Intelligence, Vol 1195, R. Trappl P. Petta (Eds.), 1997.

Golub, G H., C F. Van Loan, Matrix computations, second edition, The Johns Hopkins University Press, 1989.

Golub, G. H. , P. Comon, Tracking a few extreme singular values and vectors in signal processing Proceedings of the IEEE Volume 78, Issue 8, Aug., 1990.

Gosling, J. , B. Joy, G. L. Steele Jr., G. Bracha, The Java Language Specification, 3rd edition, Addison-Wesley, 2005.

Gosling, J. The Jackpot project, http://today.java.net/jag/page15.html

Grand, S. , D. Cliff, A. Malhotra, Creatures: artificial life autonomous software agents for home entertainment. Proceedings of the first international conference on Autonomous agents, ACM, 1997.

Gratch, J. , J. Rickel, E. Andre, N. Badler, J. Cassell, E. Petajan. Creating interactive virtual humans: Some assembly required. IEEE Intelligent Systems, 2002.

Gray, J. , *Goal and Action Inference for Helpful Robots Using Self as Simulator*, Masters of Science thesis, Media Arts and Sciences. MIT. 2004.

Hayes-Roth, B. A Blackboard Architecture for Control, Artificial Intelligence, 1985.

Hewitt, D. , I. Stevenson, E-mic: Extended Mic-stand interface controller, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Hiller, L., L. Isaacson, Musical Composition with a High-Speed Digital Computer. Journal of the Audio Engineering Society. 1958.

Hodgins, J. K. , J. F. O'Brien, J. Tumblin, Judgments of Human Motion with Different Geometric Models, Transactions on Visualization and Computer Graphics, 4 (4), December 1998

Hoskinson, R. , K. van den Doel, S. Fels, Real-time Adaptive Control of Modal Synthesis, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Hough, P.V.C. Machine Analysis of Bubble Chamber Pictures, International Conference on High Energy Accelerators and Instrumentation, CERN, 1959.

Hunt, A. , M. M. Wanderley, M. Paradis, The importance of parameter mapping in electronic instrument design. Journal of New Music Research 23(4) 2003.

Huttenlocher, D.P. , W.J. Rucklidge, A multi-resolution technique for comparing images using the Hausdorff distance", Proceedings of Computer Vision and Pattern Recognition, 1993.

Iazzetta, F. Meaning in Musical Gesture, in: Trends in Gestural Control of Music, M. M. Wanderly and M. Battier (eds.) IRCAM, 2000.

Infomus Lab, eyes-web, software product
— http://www.infomus.dist.unige.it/eywindex.html

IRCAM, j-max, open source software product
— http://freesoftware.ircam.fr/

Isla, D. The Virtual Hippocampus: Spatial Common Sense for Synthetic Creatures S.M. Thesis, MIT. 2001.

Isla, D., R. Burke, M. Downie, B. Blumberg. A Layered Brain Architecture for Synthetic Creatures. Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI). 2001.

Jackendoff, R. A comparison of rhythmic structures in music and language. In P. Kiparsky and G. Youmans (eds.) Phonetics and Phonology, Vol. 1. Rhythm and Meter (pp. 15-44), Academic Press, New York . 1990.

The **Jaxen XPath engine,** open source software
— http://jaxen.org/faq.html

Johansson, G. Visual perception of biological motion and a model for its analysis. Perception and Psychophysics, 14: pp. 201-211. 1973.

Jones, B. T. (with P. Gillespie), Last Night On Earth, Pantheon, 1997.

Jython, programming language implementation,
— http://www.jython.org

Kaelbling, L.P. , S. Rosenschein, Action and Planning in Embedded Agents, In: Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, edited by P. Maes, MIT Press/Bradford Books, 1990.

Kaiser, P. in Performance Research, 4 (2), Summer 1999. Available online at: http://www.kaiserworks.com/ideas/forsythe1.htm

Kaiser, P. , S. Eshkar, Ghostcatching, installation, 1997.

Kaltenbrunner, M. G. Geiger, S. Jordà, Dynamic Patches for Live Musical Performance, Proceedings of the 2004 Conference on New Interfaces for Musical Expression.

Klee, P. The Thinking Eye, George Wittenborn, NY, 1961.

Kohonen, T. Self-Organizing Maps. Springer Series in Information Sciences, Vol. 30, Springer, Berlin, Heidelberg, New York, 1995.

Kovar, L., M. Gleicher, F. Pighin, Motion Graphs. 2002. Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, 2002.

Kovar, L., M. Gleicher, Automated Extraction and Parameterization of Motions in Large Data Sets. Transactions on Graphics, 23, 3. SIGGRAPH 2004.

Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 7(1) 1956.

Kruskal, J. B., M. Wish. Multidimensional Scaling. Sage Publications. Beverly Hills. CA. 1977

Kuhn, H. W. The Hungarian Method for the Assignment Problem, Naval Research Logistics Quarterly, Vol. 2, 1955, pp. 83-97.

Langton C. G. (ed.), Artificial Life, Addison-Wesley, CA, 1989.

Laird, J. E. It knows what you're going to do: adding anticipation to a Quakebot,International Conference on Autonomous Agents, Proceedings of the fifth international conference on Autonomous agents, 2001.

Latombe, J.C. Robot Motion Planning. Kluwer Academic Publishers, Boston, MA, 1991.

Lerdhal, F. , R. Jackendoff, A Generative Theory of Tonal Music, Cambridge, MA, MIT Press, 1983.

Lesschaeve, J. (ed) The Dancer and the Dance,Merce Cunningham in conversation with Jacqueline Lesschaeve. Marion Boyars, New York, 1985.

Levin G., Z. Liberman, The manual input sessions, performance, 2004.

Lewis, G. Interacting with Latter-Day Musical Automata. Contemporary Music Review 18/3 (1995): 99-112.

Lewis, J. P., M. Cordner, N. Fong. Pose Space Deformations: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation. In Proceedings of ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series. 2000.

Lyons, M. J. , M. Haehnel, N. Tetsutani. Designing, Playing, and Performing with a Vision-based Mouth Interface, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Machover, T. Jeux Duex for Hyperpiano and Orchestra, Musical Score, Boosey & Hawkes, New York. 2005.

Maeda, J. Drawing By Numbers, MIT Press, 2001.

Maes, P. Modeling Adaptive Autonomous Agents, Artificial Life, 1 (1), 1994. pp. 135-62.

Mandelis, J., P. Husbands, Don't Just Play it, Grow it! : Breeding Sound Synthesis and Performance Mappings. In: Proceedings of the 2004 conference on New Interfaces for Musical Expression, Hammamatsu, Japan.

Markosian, L. , M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, J. F. Hughes. Real-Time Nonphotorealistic Rendering, In: SIGGRAPH 1997 International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 24rd annual conference on Computer graphics and interactive techniques. ACM, 1997.

Marler, P. Song-learning behavior. The interface with neuroethology. Animal Behavior Vol. 30 pp. 479-82, 1991.

Marler, P. Three models of song learning: evidence from behavior, J. Neurobiology, 33:501-516. 1997.

Marler, P. Origins of music and speech: insights from animals. in: N. L. Wallin, B. Mallin, and S. Brown, The Origins of Music. The MIT Press, Cambridge, MA. 1999.

Marrin-Nakra, T. Inside the Conductors Jacket: Analysis, Interpretation, and Musical Synthesis of Expressive Gesture. PhD Thesis. MIT, 2000.

Marrin-Nakra, T. Synthesizing expressive music through the language of conducting. Journal of New Music Research. 2002, 31 (1). 2001.

Mateas, M. An Oz-Centric Review of Interactive Drama and Believable Agents. Technical Report CMU-CS-97-156, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. June 1997.

Mateas, M. Interactive Drama, Art, and Artificial Intelligence. Ph.D. Thesis. Carnegie-Mellon University, December, 2002.

Mateas, M., A. Stern, Architecture, Authorial Idioms and Early Observations of the Interactive Drama Facade. Technical Report CMU-CS-02-198, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. December 2002.

Mathews, H. , A. Brotchie, The Oulipo Compendium, Atlas Press, 1998.

Mattsson, M. , J. Bosch, M. E. Fayad, Framework integration problems, causes, solutions. Communications of the ACM, 42 (10). 1999.

Maxis, The Sims, published by Electronic Arts. 2000-.

McAuley, J. D. On the Perception of Time as Phase: Toward an Adaptive-Oscillator Model of Rhythm. Ph.D. thesis, Indiana University, 1995.

McGregor, W. AtaXia, 2004.

Mens, T. A state-of-the-art survey on software merging. IEEE Transactions on Software Engineering, 28(5), 2002.

Merrill, D. Head-Tracking for Gestural and Continuous Control of Parameterized Audio Effects, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Meso, vvvv, software product — http://vvvv.meso.net/

Microsoft Corp. DirectX, software API — http://msdn.microsoft.com/directx/

Microsoft Corp. The rich text format — http://msdn.microsoft.com/

Mignonneau, L., C. Sommerer, Creating Artificial Life for Interactive Art and Entertainment. Leonardo, Vol. 34, No. 4, pp. 303-307, 2001.

Millennium Interactive Ltd. Creatures, 1996.

Minksy, M. Society of Mind, Simon & Schuster, New York, 1988

Minsky, M. , P. Singh, A. Sloman, The St. Thomas common sense symposium: designing architectures for human-level intelligence. The AI Magazine, Summer 2004.

Minsky, M. Music, Mind, and Meaning, Computer Music Journal, Vol. 5, Number 3. 1981.

Minsky, M. Semantic Information Processing, MIT Press, 1968.

Minsky, M. A Framework for Representing Knowledge. MIT AI Lab, Memo 360, June 1974.

Minsky, M. Emotion Machine, forthcoming.

Miranda E. R. (ed.), Readings in Music and Artificial Intelligence. Routledge, 1999.

Miranda, E. R. Regarding Music, Machines, Intelligence and the Brain: An Introduction to Music and AI. In E.R. Miranda (ed.) Readings in music and artificial intelligence, Hardwood Academic Publishers, 2000.

Morita, T. , T. Kanade, A Sequential Factorization Method for Recovering Shape and Motion from Image Streams. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19 (8), 1997.

Muybridge, E. Animals in Motion, Dover, 1957.

Nyman, M. Experimental Music: Cage and Beyond. 2nd ed. Cambridge, UK: Cambridge Univ. Press, 1999 (p. 4).

Obermaier, K. Apparition, performance, 2004.

Obermaier, K. Vivisector, performance, 2002.

OpenGL ARB, OpenGL vertex blending — http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_blend.txt

Palacio-Quintin, C. The Hyper-Flute, Proceedings of the 2003 Conference on New Interfaces for Musical Expression.

Pelleg, D. , A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In Proceedings of the 17th International Conference on Machine Learning, pages 727-734. Morgan Kaufmann, San Francisco, CA, 2000.

Peretz, I. Brain specialization for music : New evidence from congenital amusia. In: Biological foundations of music. Annals of the New York Academy of Sciences, 930, pp. 153-165, 2001.

Perlin, K. Real Time Responsive Animation with Personality, IEEE Transactions on Visualization and Computer Graphics, 1 (1), 1995.

Perlin, K. , A. Goldberg, Improv — a system for scripting interactive actors in virtual worlds. In: SIGGRAPH 1996 International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM, 1996.

Peitgen, H. , H. Jurgens, D. Saupe, Chaos And Fractals: New Frontiers of Science, Springer-Verlag, 1992.

The **PicoContainer framework**, open source software, http://www.picocontainer.org/

Pinhanez, C. S. Computer theater. Technical Report 378, MIT Media Laboratory Perceptual Computing Section, May 1996.

Porter, T. , T. Duff. Compositing digital images. Computer Graphics, 18 (3), July 1984.

M. Pukette, pd, open source software product — http://www-crca.ucsd.edu/~msp/software.html

Python, programming language, http://www.python.org

Rabin, S. AI Game Programming Wisdom 1-2, Charles River Media, Cambridge MA, 2002-3.

Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2): 257-286, February 1989.

The **revision control system**, open source software tool, http://www.gnu.org/software/rcs/rcs.html

Reynolds, C. An online resource for non-photorealist rendering techniques. http://www.red3d.com/cwr/npr/

Rinaldo, K. Technology Recapitulates Phylogeny: Artificial Life Art, Leonardo 31, No. 5,371-376 (1998).

Roads, C. The Computer Music Tutorial. MIT Press, 1996.

Ronald, E. M. A. , M. Sipper, M. S. Capcarrere, Design, Observation, Surprise! A Test of Emergence. Artificial Life 5: 225-239 (1999).

Rose, C. Verbs and Adverbs: Multidimensional Motion Interpolation Using Radial Basis Functions. Department of Computer Science. Princeton, NJ, Princeton University. 1999.

Rost, R. J. OpenGL Shading Language, Addison-Wesley, 2002.

Roubaud, J. Mathematics in the Method of Raymond Queneau, reprinted in: W. F. Motte Jr (trans., ed.), Oulipo a primer of potential literature, University of Nebraska Press, 1986.

Rowe, R. Interactive Music Systems: Machine Listening and Composing, MIT Press, Cambridge MA, 1992.

Saul, L. K. , D. D. Lee, C. L. Isbell, Y. LeCun, Real time voice processing with audiovisual feedback : toward autonomous agents with perfect pitch, Advances in Neural Information Processing Systems 15. NIPS 2002.

Shao, W., V. Ng-How-Hing, A general joint component framework for realistic articulation in human characters. In ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics.

Sims, K. Artificial Evolution for Computer Graphics. in Computer Graphics, 25(4), July 1991.

Sims, K. Evolving 3D Morphology and Behavior by Competition. In: Artificial Life IV Proceedings, R. Brooks & P. Maes (eds.) , MIT Press, 1994.

SoftImage Corp. XSI, software product —http://www.softimage.com/products/behavior/v2/default.asp

Sommerer, C., L. Mignonneau, Art as a Living System: Interactive Computer Artworks. Leonardo, Vol. 32, No. 3, pp. 165-173, 1999.

Sparacino, F. G. Davenport, A. Pentland, Media in performance: Interactive spaces for dance, theater, circus, and museum exhibits. IBM Systems Journal, 39 (3&4), 2000.

The **Spring framework**, open source software, http://www.springframework.org/

Starr, F. (ed.) Changes: Notes on Choreography, Something Else Press, New York, 1968.

Steele, G. L. Common LISP: The Language, Bedford: Digital Press, 1990.

Steele, G. L. Growing a Language, Journal of Higher-Order and Symbolic Computation 12(3) 1999.

Steel, G. L. , G. J. Sussman, The Revised Report on SCHEME, MIT AI Lab Memo 452. 198.

Strassmann, S. Hairy Brushes. In SIGGRAPH 1986 International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 13rd annual conference on Computer graphics and interactive techniques. ACM, 1986.

Stauffer ,D. , A. Arahony, Introduction to Percolation Theory, Taylor & Francis. 2001.

Strothotte, T. , S. Schlechtweg, Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation. Morgan Kauffman, 2002.

Sulcas, R. William Forsythe: The poetry of disappearance and the great tradition. previously available online: http://frankfurt-ballett.de/articles2.html however, since the dissolution of the Ballett Frankfurt, this paper is currently available through the internet archive: http://web.archive.org

Sutherland, I. Sketchpad : a Man-Machine graphical communication system,Annual ACM IEEE Design Automation Conference, 1964.

Sutton, R. S. , A. G. Barto, Reinforcement Learning, MIT Press, 1998

Sylvester, D. Interviews with Francis Bacon, Thames-Hudson, 1975.

Teicher H. (ed.) Trisha Brown: Dance and Art in Dialogue. MIT Press, Cambridge, MA, 2002.

Temperly, D. The Cognition of Basic Musical Structures, MIT Press, 2001.

Thorpe, W. H. Bird-song; the biology of vocal communication and expression in birds. Cambridge University Press, Cambridge, UK. 1961.

Todd, P. M., G.M. Werner, Frankensteinian methods for evolutionary music composition. In: N. Griffith and P.M. Todd (eds.), Musical networks: Parallel distributed perception and performance Cambridge, MA: MIT Press/Bradford Books 1999.

Todd, P.M., D.G. Loy (Eds.) Music and Connectionism. Cambridge, MA, MIT Press 1991.

Tomlinson, B., Synthetic Social Relationships for Computational Entities. PhD Thesis, MIT, June 2002.

Trehub, S. E. , E. G. Schellenberg, D. S. Hill, Music perception and cognition: A developmental perspective. In: I. Deliége, and J. A. Sloboda, Music perception and cognition. Psychology Press, Sussex, UK. 1997

Troika Ranch, Future of Memory, performance, 2003.

Troikatronix, Isadora, software product — http://www.troikatronix.com

Vaughan, D., Merce Cunningham: 50 Years, Aperture, New York, 1999.

Vercoe, B. , M. Puckette. Synthetic Rehearsal: Training the Synthetic Performer. Proceedings of the 1985 International Computer Music M. Isard and A. Blake, Condensation — conditional density propagation for visual tracking. International Journal Computer Vision, 29, 1, 1998

Viterbi, A. J. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory 13(2):260-267, April 1967.

Wall, L. , T. Christiansen, J. Orwant, Programming Perl, O'Reilly, 2000.

Wallin, N. L., B. Mallin,S. Brown, The Origins of Music. The MIT Press, Cambridge, MA. 1999.

Wessel, D. , M. Wright, Problems and Prospects for Intimate Musical Control of Computers, Computer Music Journal, 26 (3), MIT Press, 2002.

Whitelaw, M. Metacreation: Art and Artificial Life, MIT Press, Cambridge, MA, 2004.

Whitelaw, M. The Abstract Organism: Towards a Prehistory for A-Life Art, Leonardo Vol. 34, No. 4, 2001.

Winkler, T. Interactive Signal Processing for Acoustic Instruments, Proceedings of the 1991 International Computer Music Conference. San Fancisco, International Computer Music Association.

Winker, T. Motion Sensing Music. Proceedings of the International Conference on Computer Music 1998, San Francisco, International Computer Music Association.

Winkler, T. Making motion musical: Gesture Mapping Strategies for Interactive Computer Music. Proceedings of the 1995 International Computer Music Conference. San Francisco, International Computer Music Association, pp. 261-4.

Winkler, T. , Live Video and Sound Processing for Dance. From Video, Technology and Performance Festival, Brown University April 4-5, 2003. Available online at: http://www.brown.edu/Departments/Music/faculty/winkler/papers/

XML, a ubiquitous W3C committee standard — http://www.w3.org/XML/

XPath, W3C spectfication — http://www.w3.org/TR/xpath

S-Y. Yoon, B. Blumberg, G. Schneider, Motivation Driven Learning for Interactive Synthetic Characters. Proceedings of 4th Int'l. Conf. on Autonomous Agents. 2000.

Zimmer, E., S. Quasha, Body against Body: The Dance and other collaborations with Bill T. Jones and Arnie Zane. Station Hill Press, New York, 1990.

ZKM (Center for Art and Media Technology) / Ballett Frankfurt, Improvisation Technologies (CDROM), 1993.

Zhang C., T. Chen A survey on image-based rendering-representation, sampling and compression. In: Signal Processing: Image Communication, January 2004, 19, (1) , pp. 1-28

Colophon

The body text of this document is set in **Adobe Jenson Pro**, the code excerpts and diagram labels are in **Cronos Pro**. The text and layout was assembled using Apple's **Pages**. The diagrams were edited in The Omni Group's **OmniGraffle Pro**. Equations were created using Doug Rowland's **Equation Service** interface to **LaTeX**. Outlining and assent management was performed in The Omni Group's **OmniOutliner**. Cross references, figure labels, bibliography and the table of contents were automatically produced by a homemade system of **Java** and **Jython** exploiting the use of XML by Pages. Many diagrams were procedurally generated by code and automatically exported to OmniGraffle's XML format, a small number were exported directly to PDF via Apple's **Cocoa application framework**. Almost all images were processed at some point by Adobe **Photoshop**.