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 improvisational 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 *The 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. As such, 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 retelling 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.

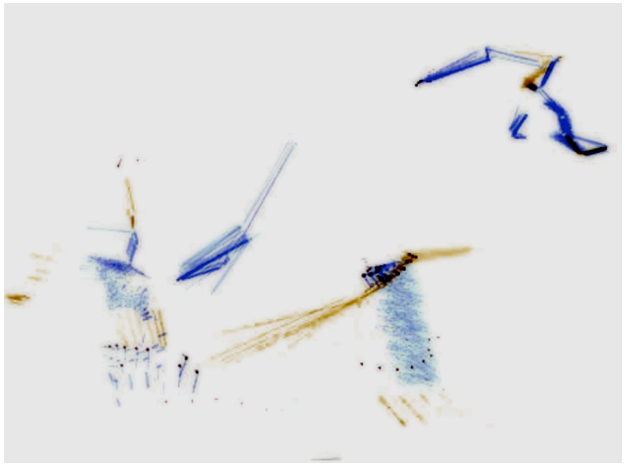


figure 88. Parts of the work *Lifelike* began to explore the passage between the abstract and abstraction — here is a horse figure, based on the decidedly "offline motion capture" of photography pioneer Eadweard Muybridge, see: **E. Muybridge**, *Animals in Motion*, Dover, 1957.

In *22*, the imagery (and the music, by composer Roger Reynolds) enter the work as additional parallel tracks, co-existing with the performer's movement and words and occasionally threatening the flow of the other media. The imagery very clearly "interacts" 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 generalized 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 in which 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 either from the rest of the scene or from 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 is only ever 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.

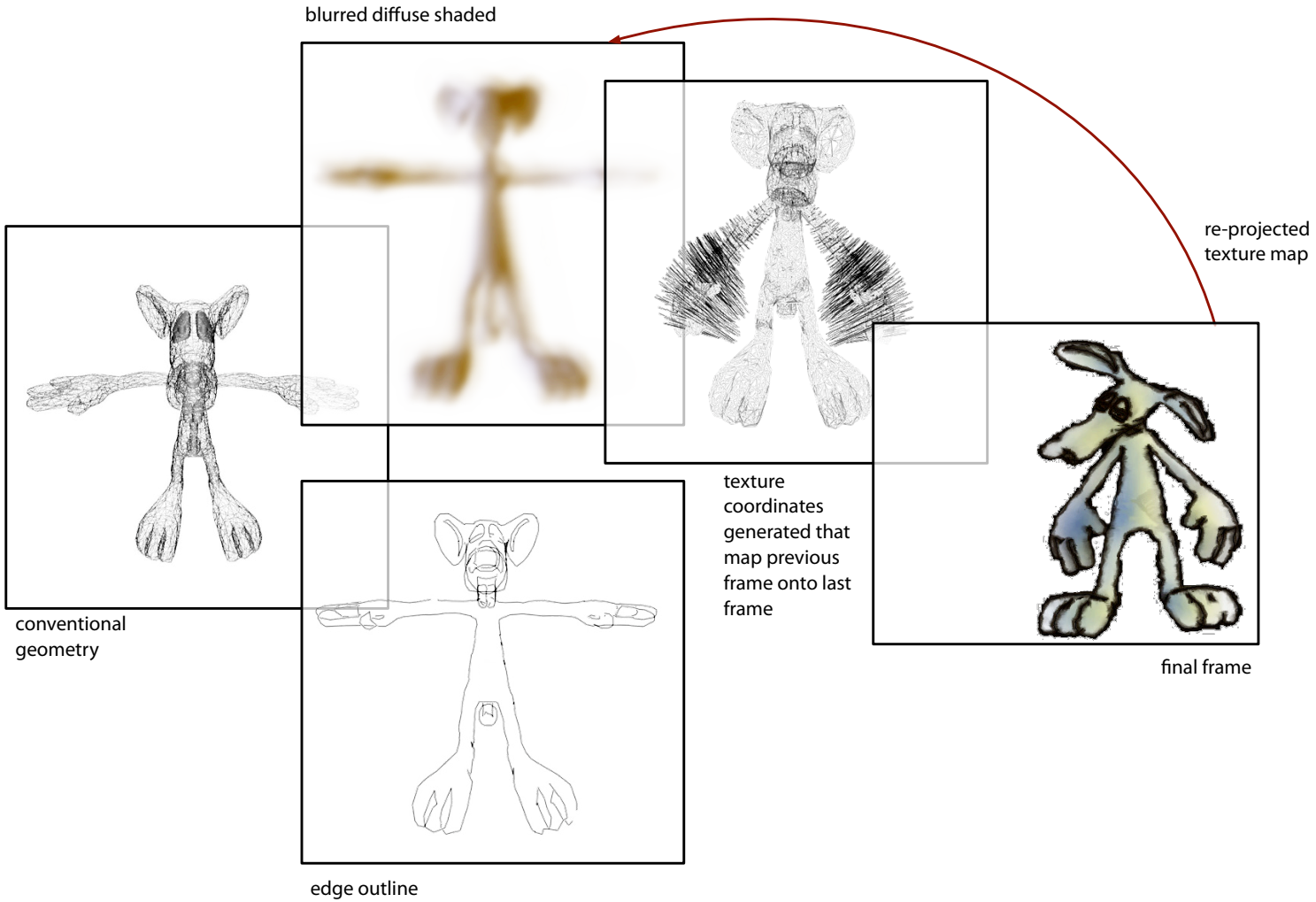


figure 89. This re-projection renderer character — *Max* — illustrates the basic re-projection shading and edge tracing setup.

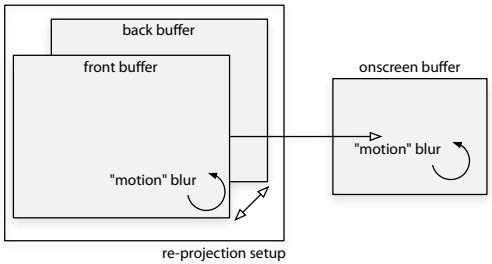


figure 90. The standard re-projection setup renderers offscreen at a high resolution (with a separate level of motion blur) . The back buffer is used to texture material in the front buffer.

Autodesk / Discreet —
http://www.discreet.com/3dsmax

Firstly, it is completely tied to the geometric content of the scene and secondly, it is 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 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. — and 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 the one fifth of the prepared geometry/video library made it into this dance piece, it is expected that these further uses of this re-projection shader will be explored in a separate installation.

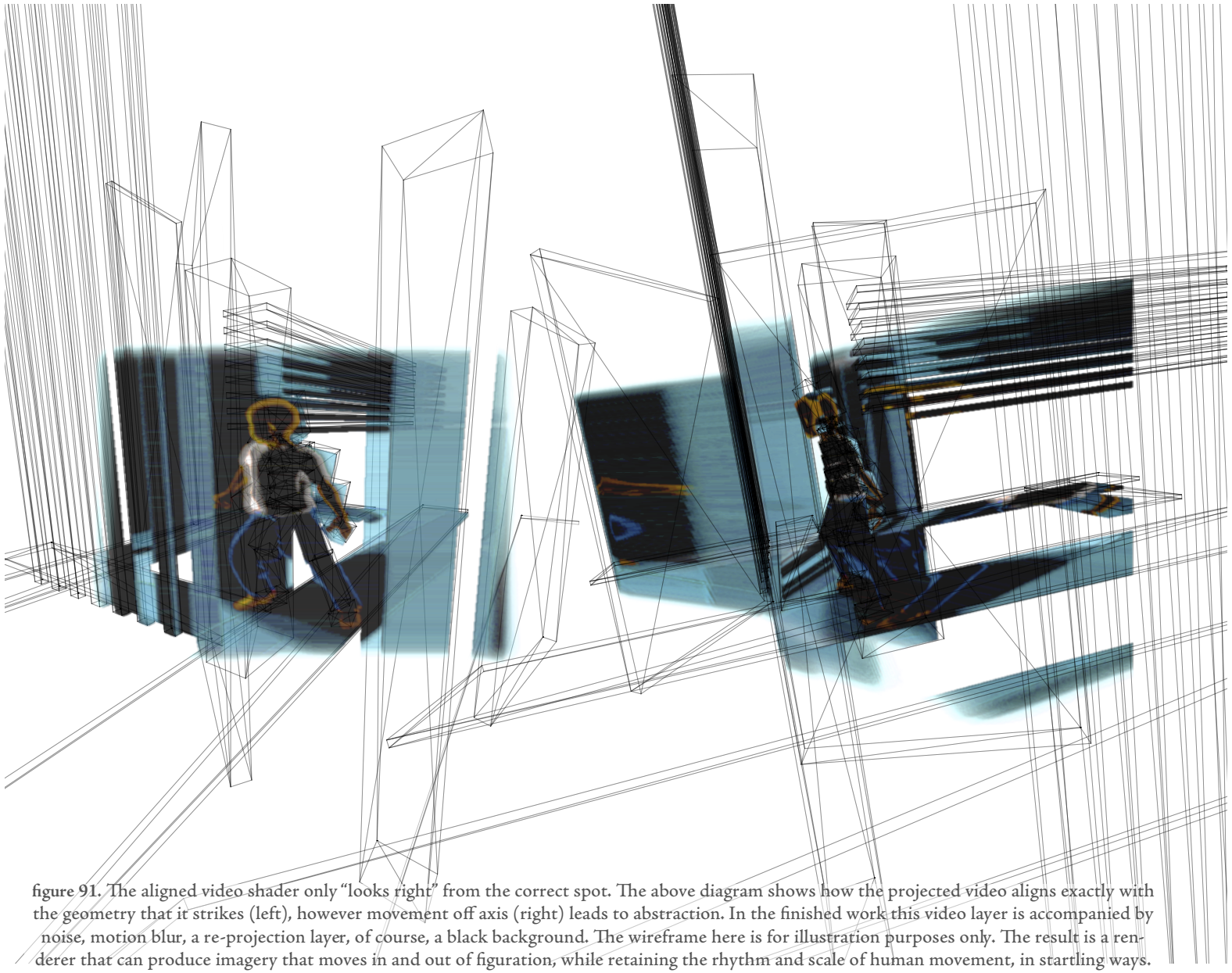


figure 91. The aligned video shader only "looks right" from the correct spot. The above diagram shows how the projected video aligns exactly with the geometry that it strikes (left), however movement off axis (right) leads to abstraction. In the finished work this video layer is accompanied by noise, 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.

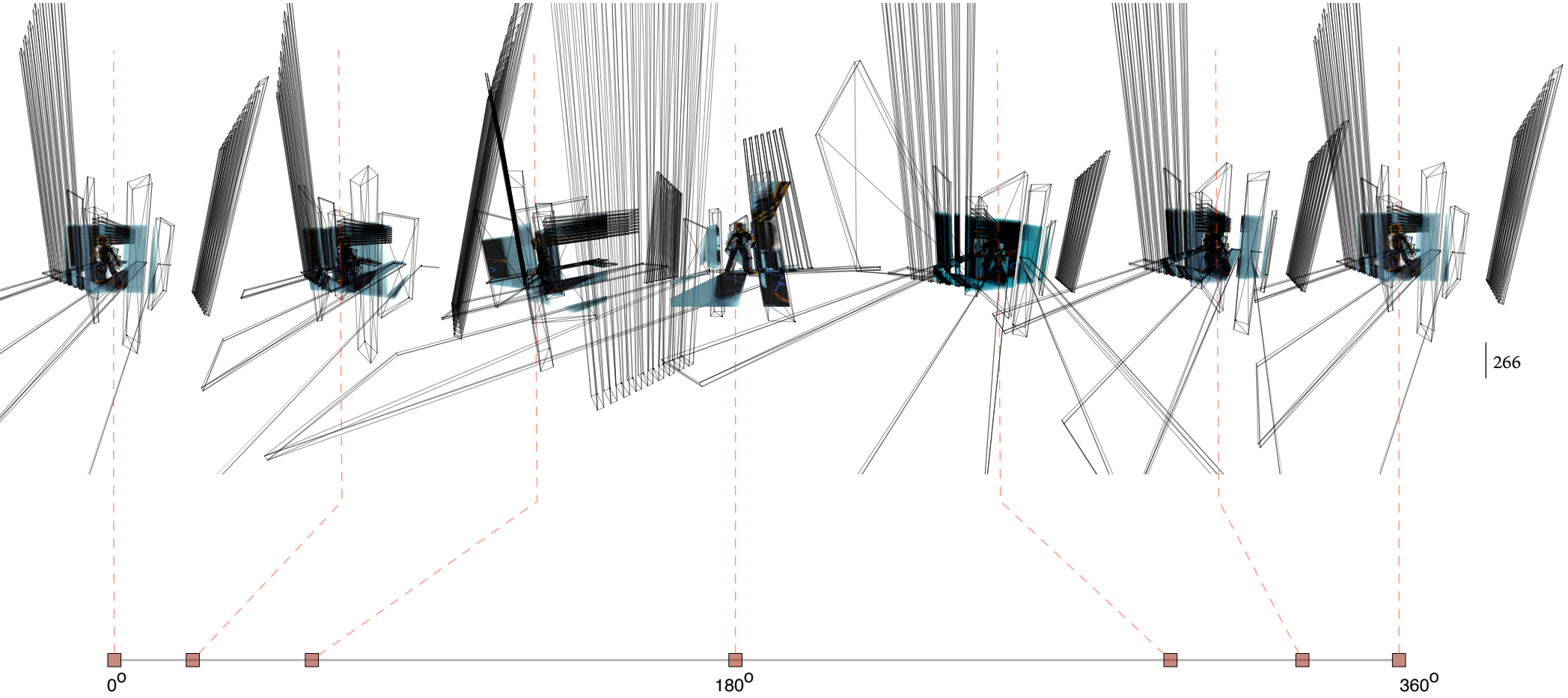


figure 92. The same scene as the previous diagram, slowly rotating around the child figure — which moves in and out of clarity.

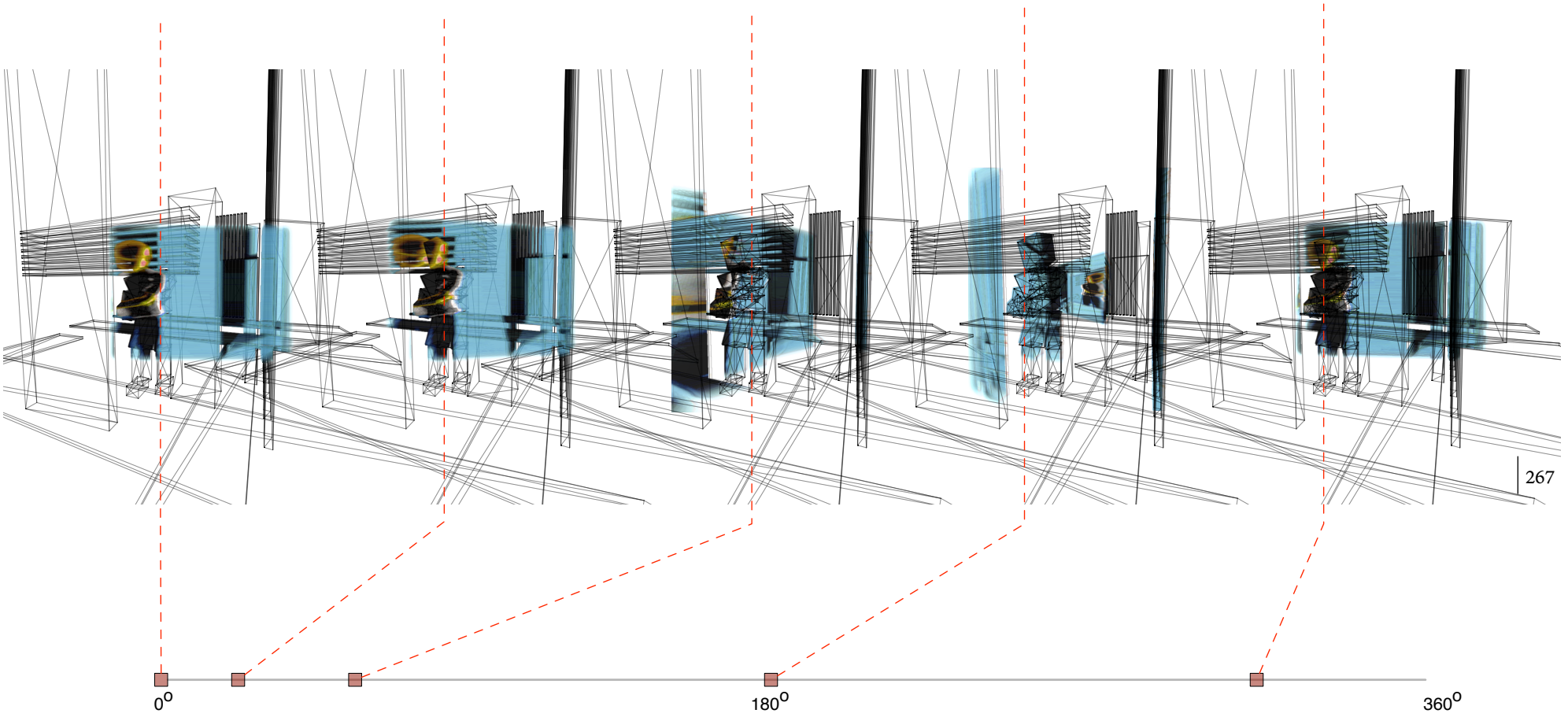


figure 93. Virtual camera movement is not the only degree of freedom that this shading technique offers. Here we rotate the virtual projector around the scene.

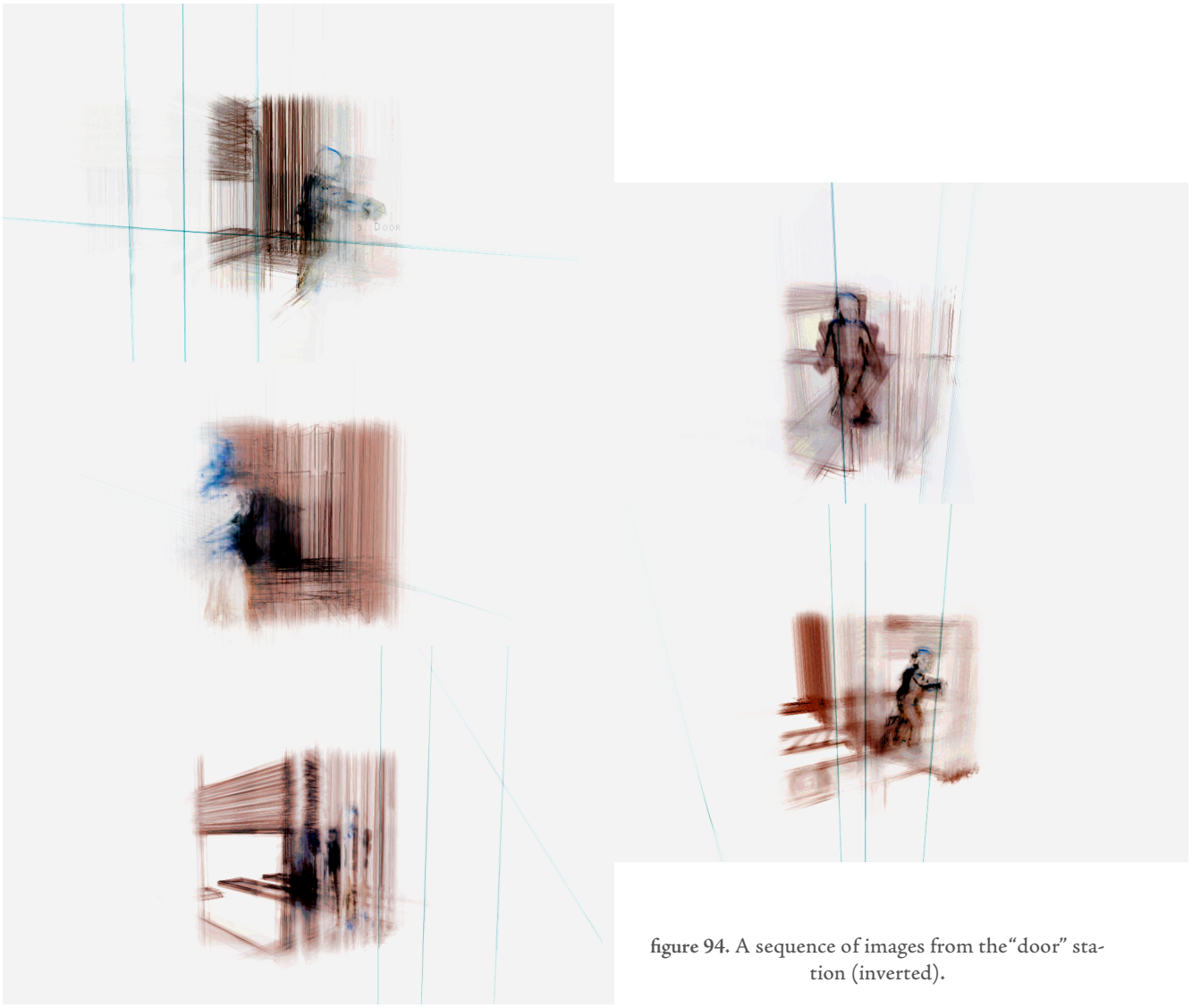


figure 94. A sequence of images from the "door" station (inverted).

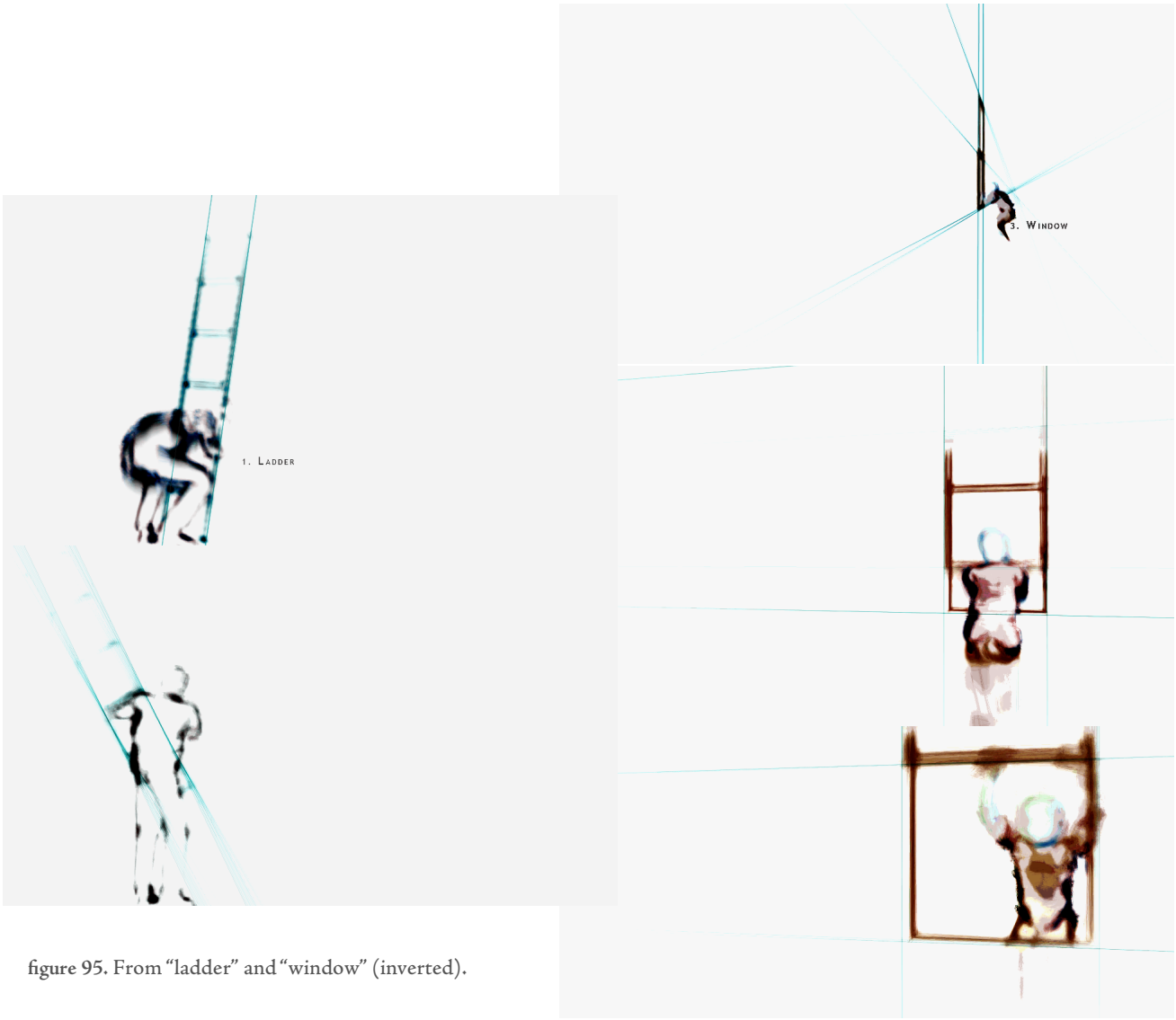


figure 95. From "ladder" and "window" (inverted).

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 the 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 for 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.

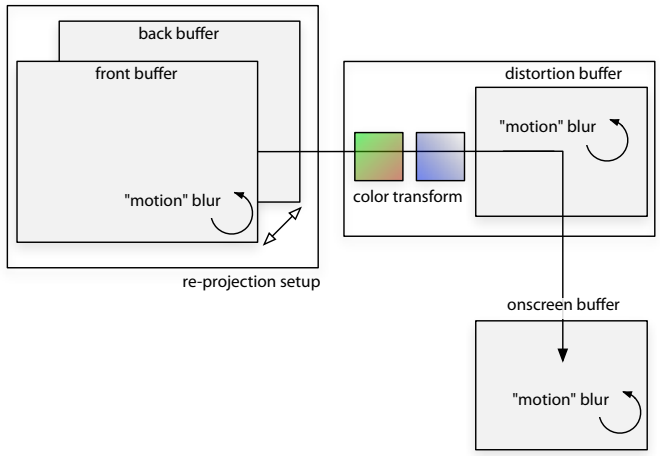


figure 96. The full re-projection setup adds a buffer used to perturb the drawing of the main buffer onto the screen and an additional two texture lookups.

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 a 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 thwarted as properties of the color-space are set to be recomposed as needed by the artist during the work.

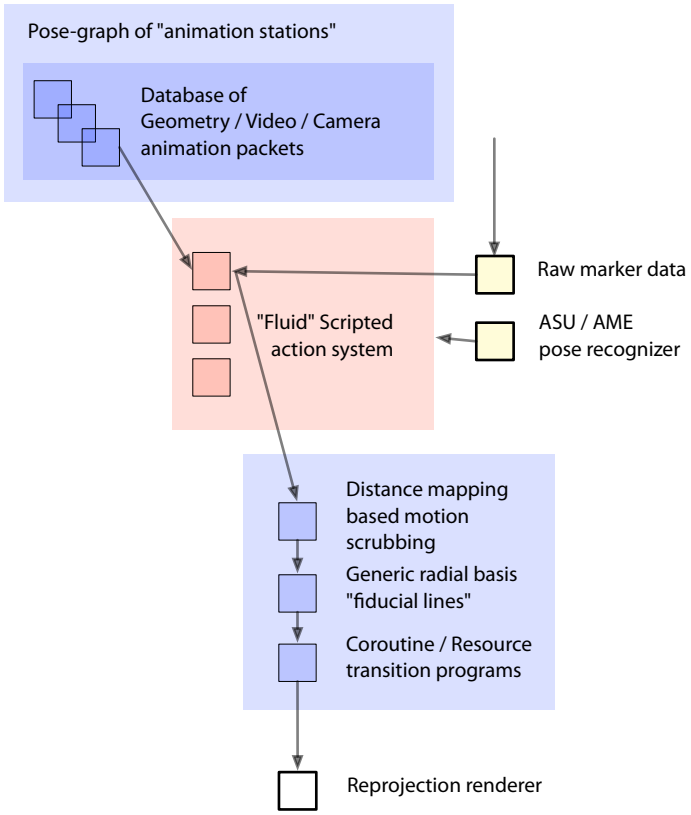


figure 97. The *22* agent system diagram.

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 140*, 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, which requires the simultaneous use of two high resolution video textures. However, to cross-fade between multiple stations each station we 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, as well as the 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 186*.

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, like its predecessor *21*, licenses the creation of dance works with multiple, simultaneous, streams of media that are deliberately crafted to create new figurative and narrative meaning (unlike a classic Cunningham / Cage / Rauschenberg work). 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 that 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 that will, for the reader of this complete thesis, seem familiar — choreographic trackers based on the b-tracker frame-

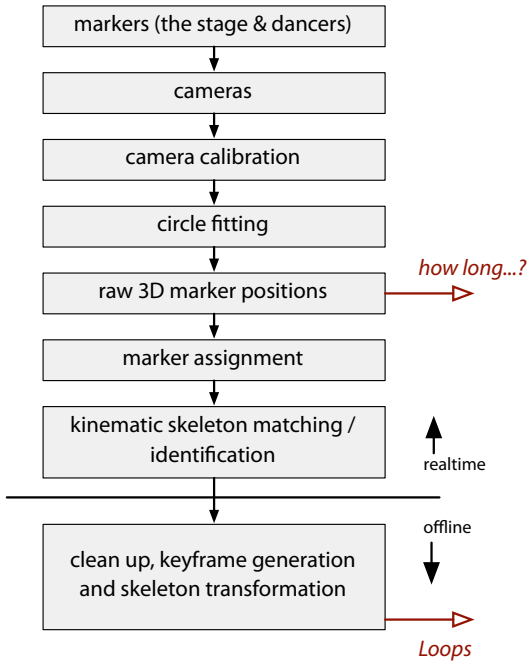


figure 98. Motion-capture is a series of complex data-processing tasks. Real-time motion capture systems must offer data at a variety of levels of processing if they are to be used for live digital artworks. ***how long...*** takes most of its data at a far lower level than other "real-time" applications. *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.

work, 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 enter a *body representation* allows for perceptual mechanisms to be reused as proprioceptual mechanisms and allows the agents to perceive other agent's bodies in the same manner as they perceive the live dancers. As we shall see below, it takes some delicate work to turn these untracked points of motion into data suitable for coupled visual imagery.

Unlike *Loops*, *The Music Creatures* or even *22*, the other interactive work for dance that premiered on the same evening, *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 first instantiated and then 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 that 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. We begin with the treatment of the exciting, but troublesome, realities of the currently available real-time motion capture data; progress to the general framework for constructing the bodies of this work's agents; and then reach 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.

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 perceptual structures of the agents, rather than passively consuming the output of a proprietary black box, should come as no surprise.

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.

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 conception of the work would have 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.

J. C. Bezdek, Fuzzy Mathematics in Pattern Classification. Ithaca, NY: Cornell University; 1973.

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 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.

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 simul-

taneously 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.

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.

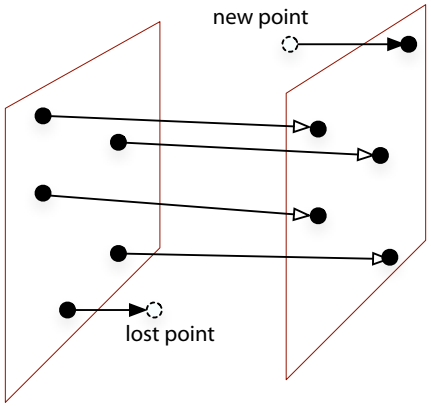


figure 99. The point matching algorithm tries to assign this frame's set of points to the previous, predicted, ongoing model positions. Occasionally points are dropped and new points enter the scene.

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_σ 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 Voronoi partitioning of space into k areas (clusters), each cluster being represented by a center, which is 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 99*. 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 parameterized, the specifics of the actions taken, so such "mapping"-like filter networks will have their place, but there are a few general principles that the use of these measurements adhere to to make the working practice around these filter networks tolerable, or even tenable within a rehearsal or improvisatory 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

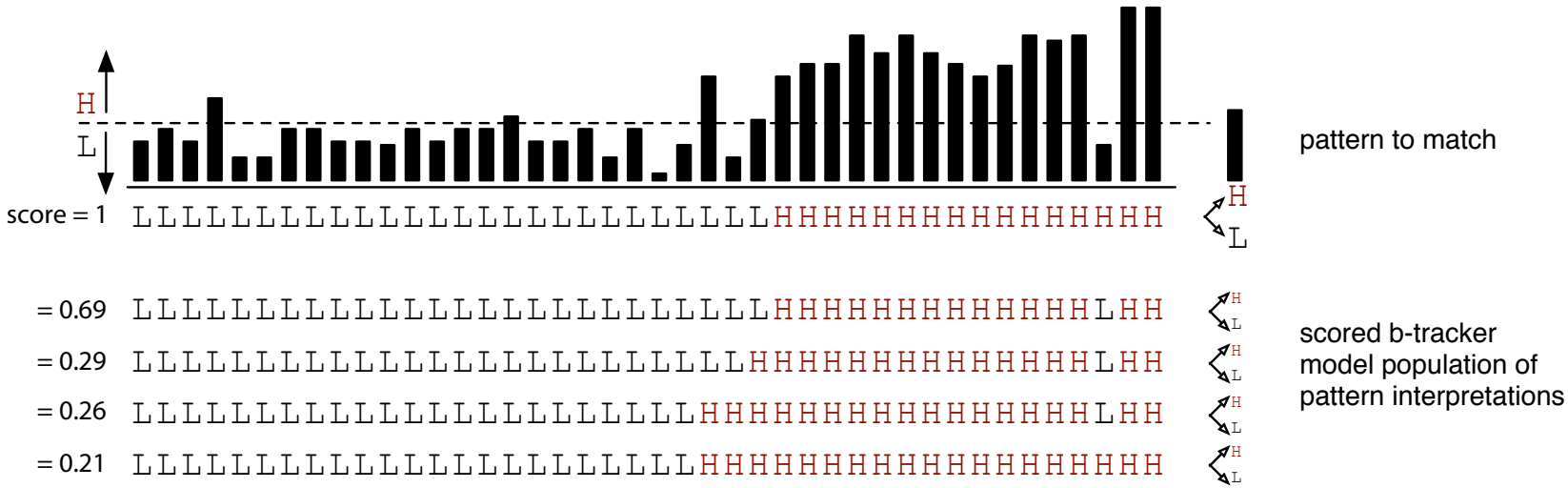


figure 100. A b-tracker population of models trying to parse a sequence of average heights into low (L) and high (H).

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 premade 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 to not 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 more 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 might be 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 than 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 136*; **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.

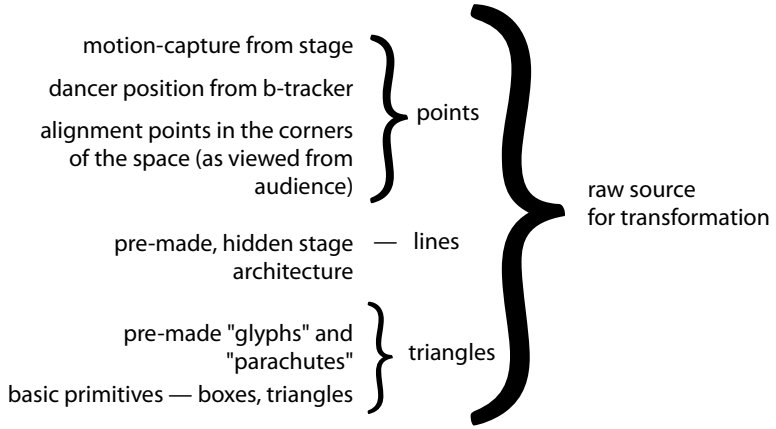


figure 101. The raw material in each of the three graphic languages. Of course, the motion capture from the stage is by far the most present.

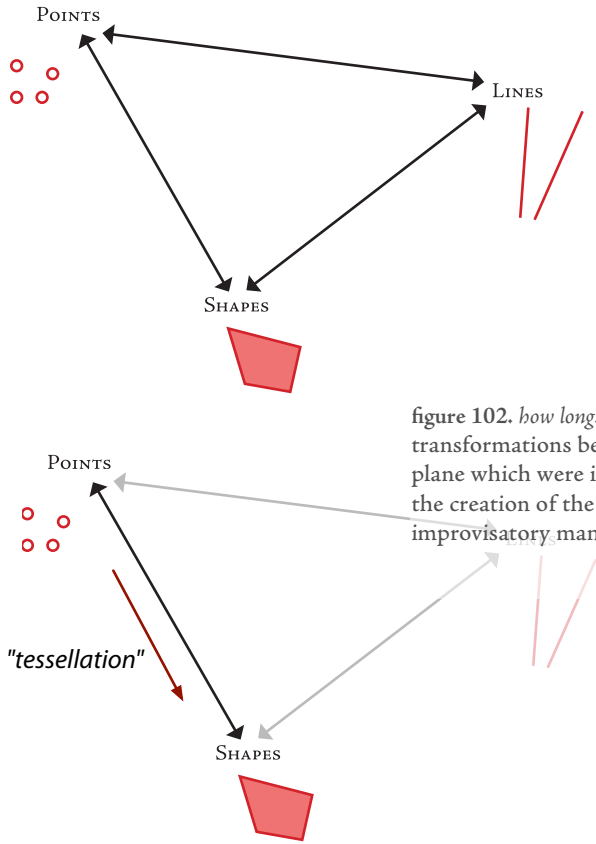


figure 102. *how long...* builds a vocabulary of transformations between point, line and plane which were in the initial stages of the creation of the work, deployed in an improvisatory manner with live dancers.

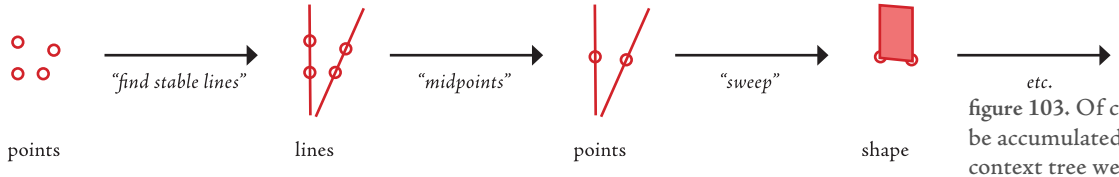
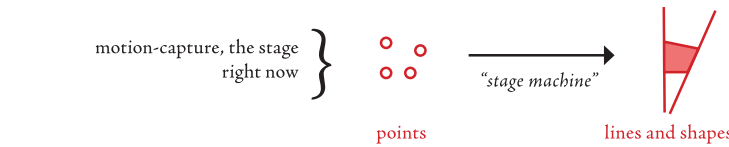
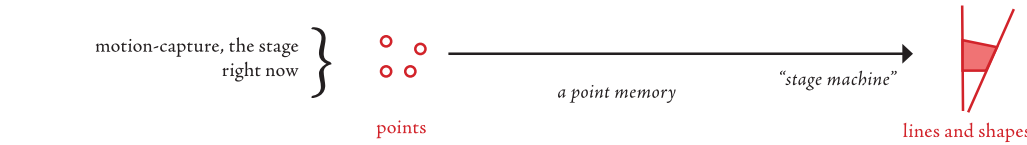


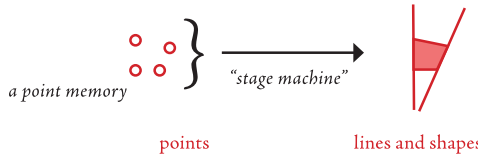
figure 103. Of course, these processes can be accumulated. By using the tricks of the context tree we can stack processes on top of each other while loosely coupling them and the agents that instantiate them.



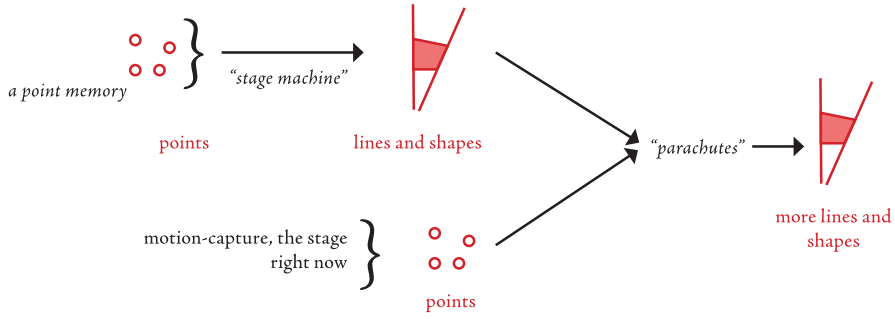
The points on the stage are illustrated by a particular agent, here, the motor system of "stage machine"



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 104. An example ordering of transfer processes — taken from documents shared between the collaborators.

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/

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 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 195*. We are now in a position to form programs of filtering elements stored in context-tree-local lists, *pages 204*. 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 begin 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).

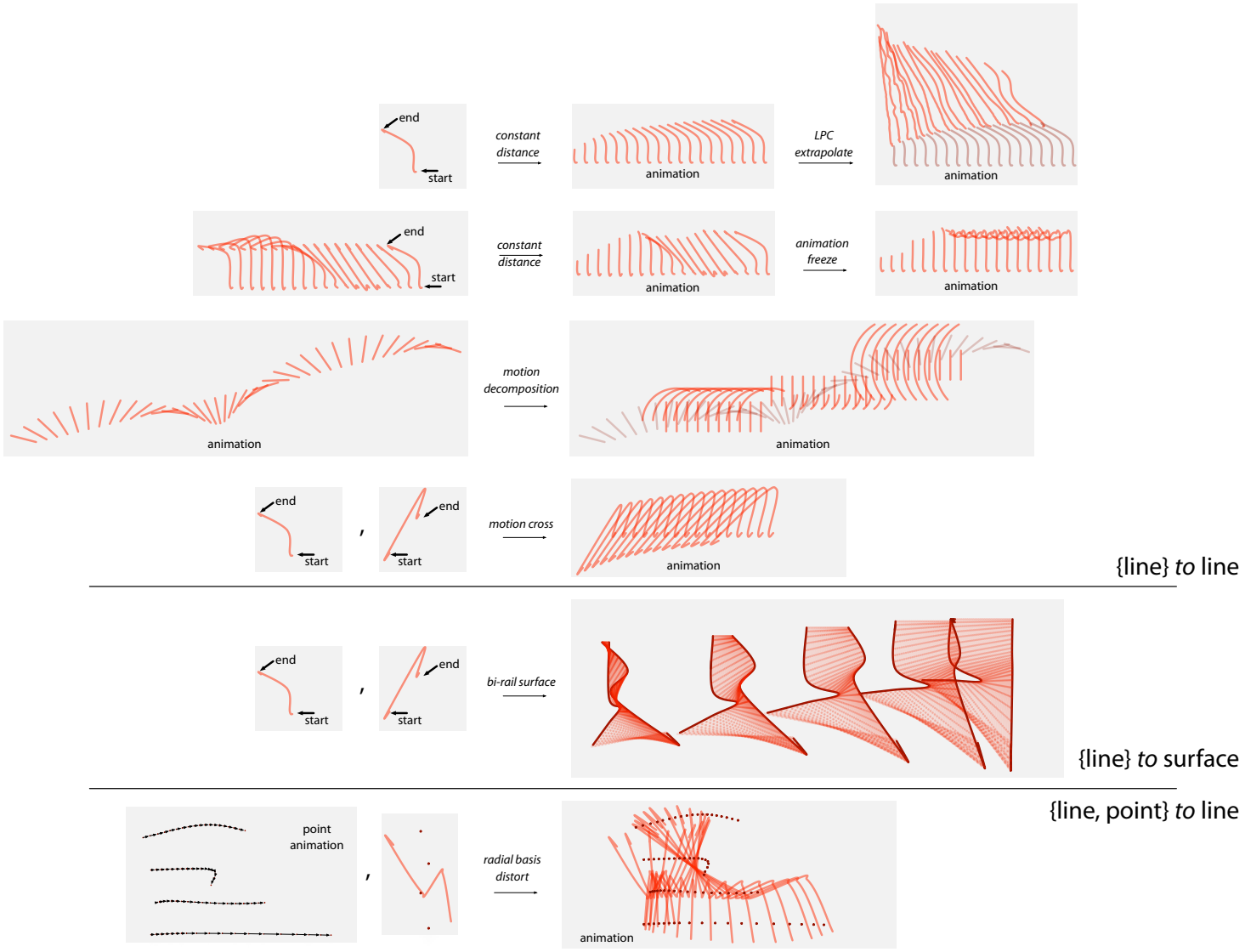


figure 105. Line acceptor primitives convert betweeen (potentially animated) lines to (potentially animated) lines. Other conversions link points and surfaces to linear, gestural forms.

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 324*, and the *parachute / accumulation* agent, *page 329*.

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.

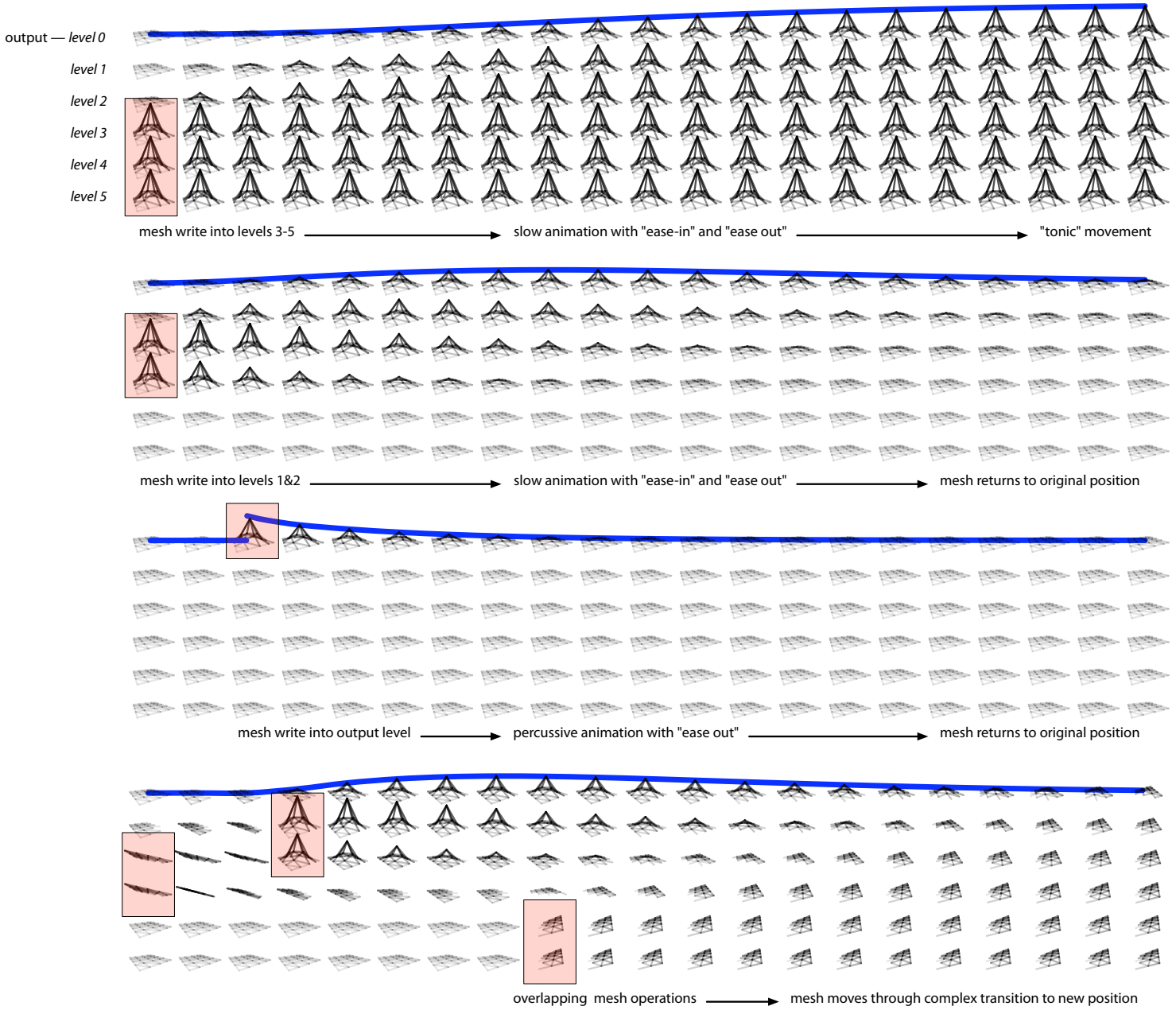


figure 106. By writing into the stack of vertex buffers at various places, smooth, complex or percussive transitions can be blended and overlapped without interaction between the processes that cause them.

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. 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 code (in Python) 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 *homogenous* 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:

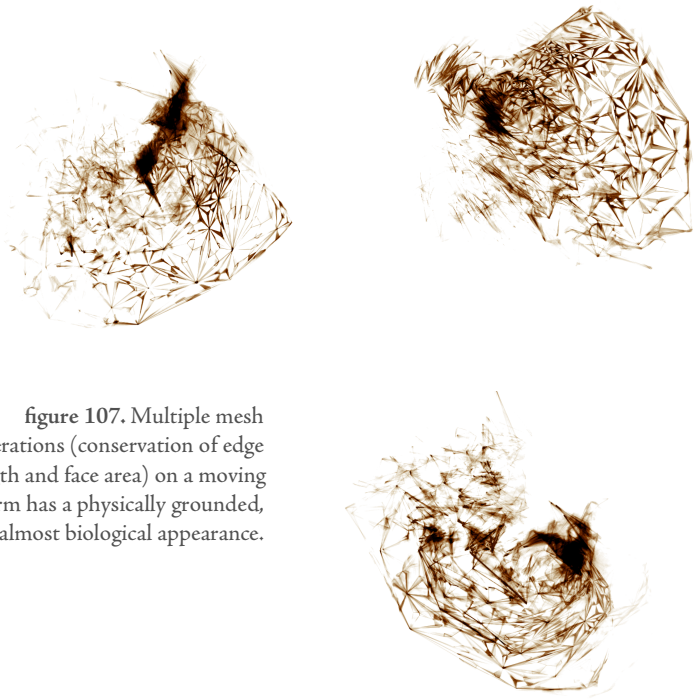


figure 107. Multiple mesh operations (conservation of edge length and face area) on a moving form has a physically grounded, almost biological appearance.

$$\alpha_e = \alpha' \alpha_{\text{mul}} + \alpha_{\text{add}}$$

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 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 through 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

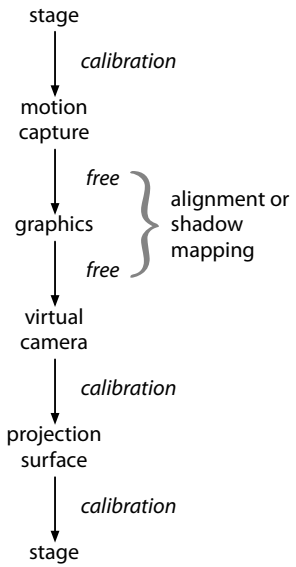


figure 108. The complete stack of coordinate system transformations that are possible when using real-time motion capture in theater. In order to complete this loop we need to know the corners of the projection screen, and the coordinate system of the motion capture hardware in the same frame of reference 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 is 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.

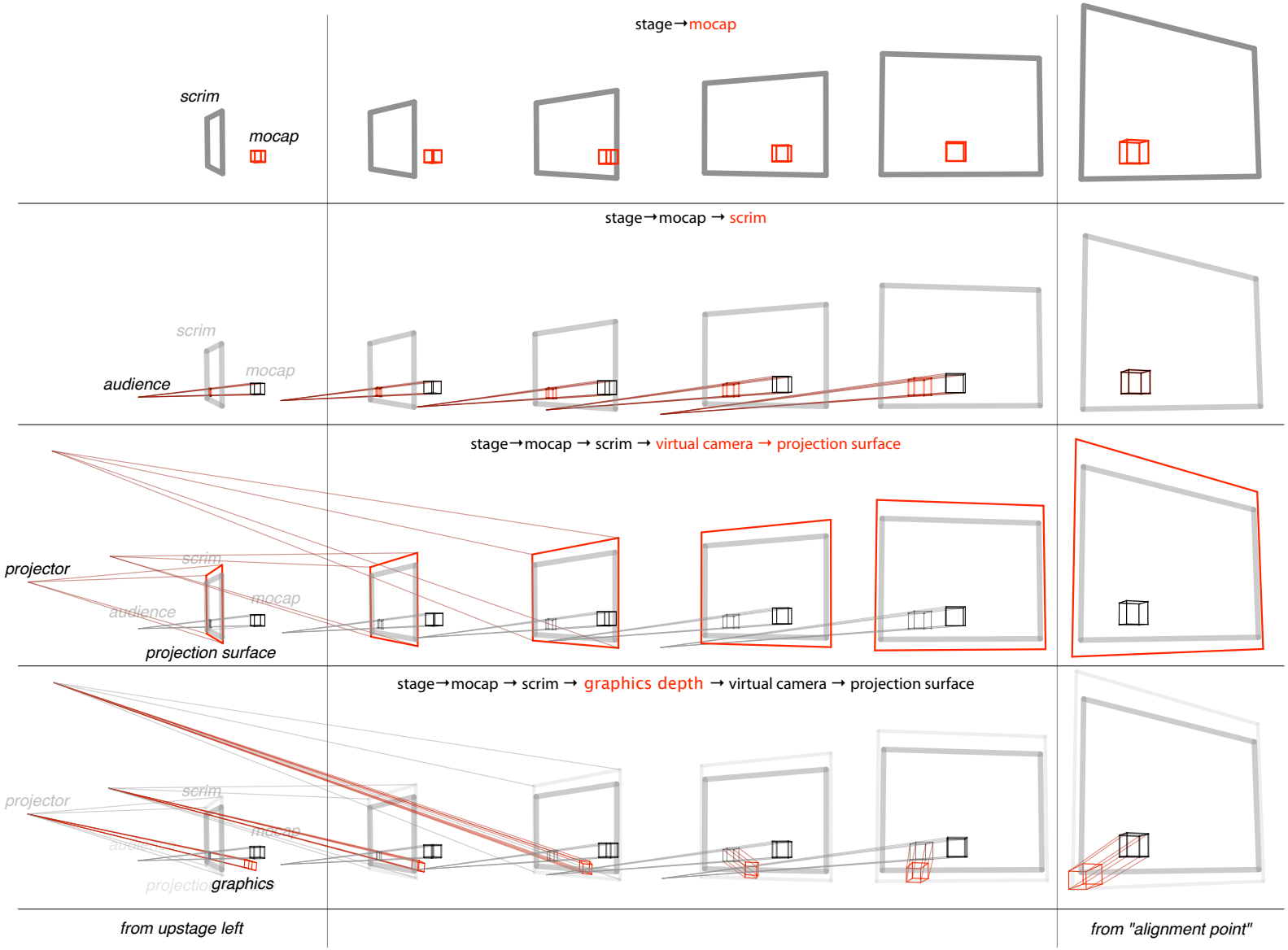


figure 109. A camera orbit of an object on stage with imagry projected onto a scrim in the front of the stage. From the alignment point the imagery point aligns exactly, from the point of view of the graphics world, the object has an oddly distorted perspective.

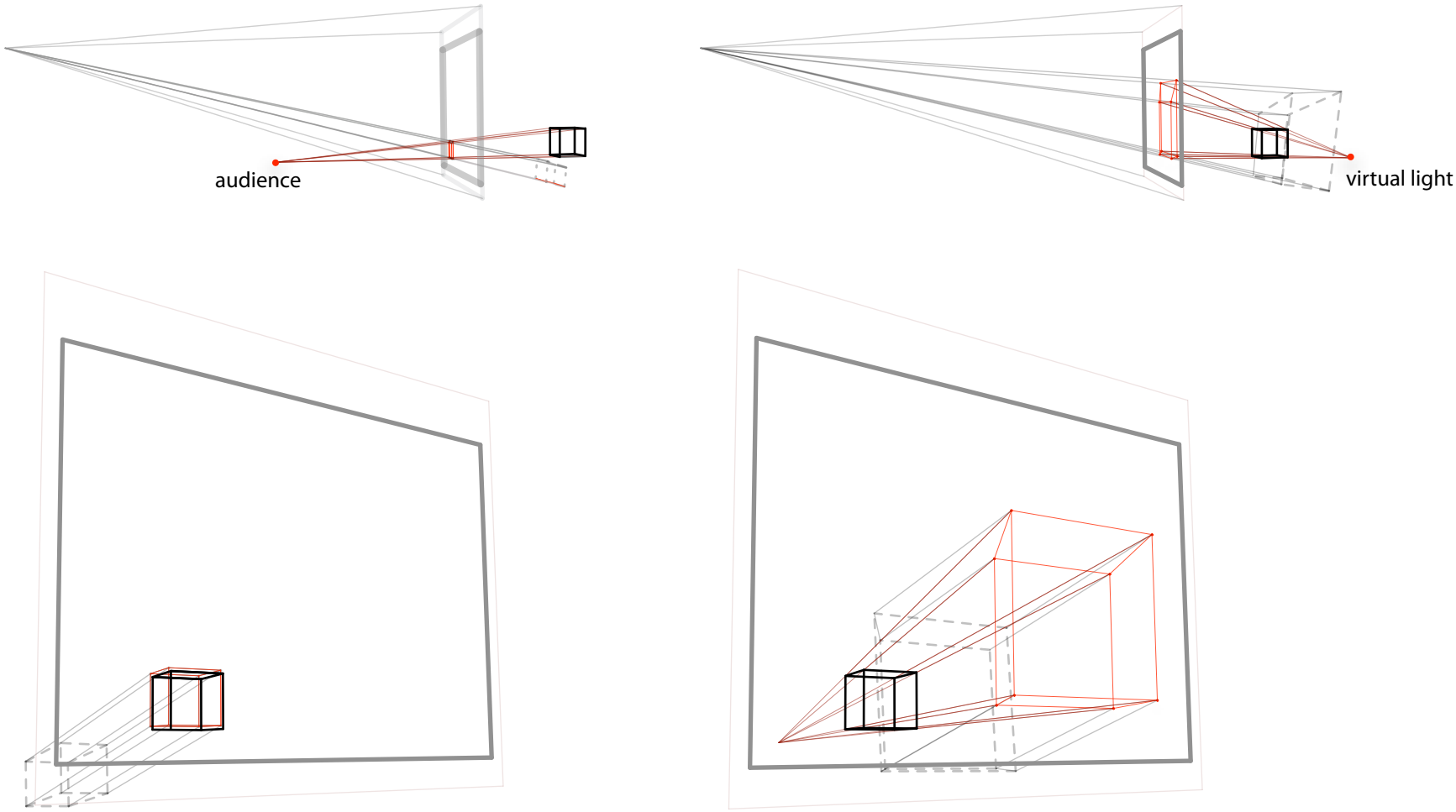


figure 110. The audience-aligned and shadow-projection mappings compared. An identical geometric construction yields dramatically different results. In the latter, objects and figures loom large and transiently on the scrim.

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.

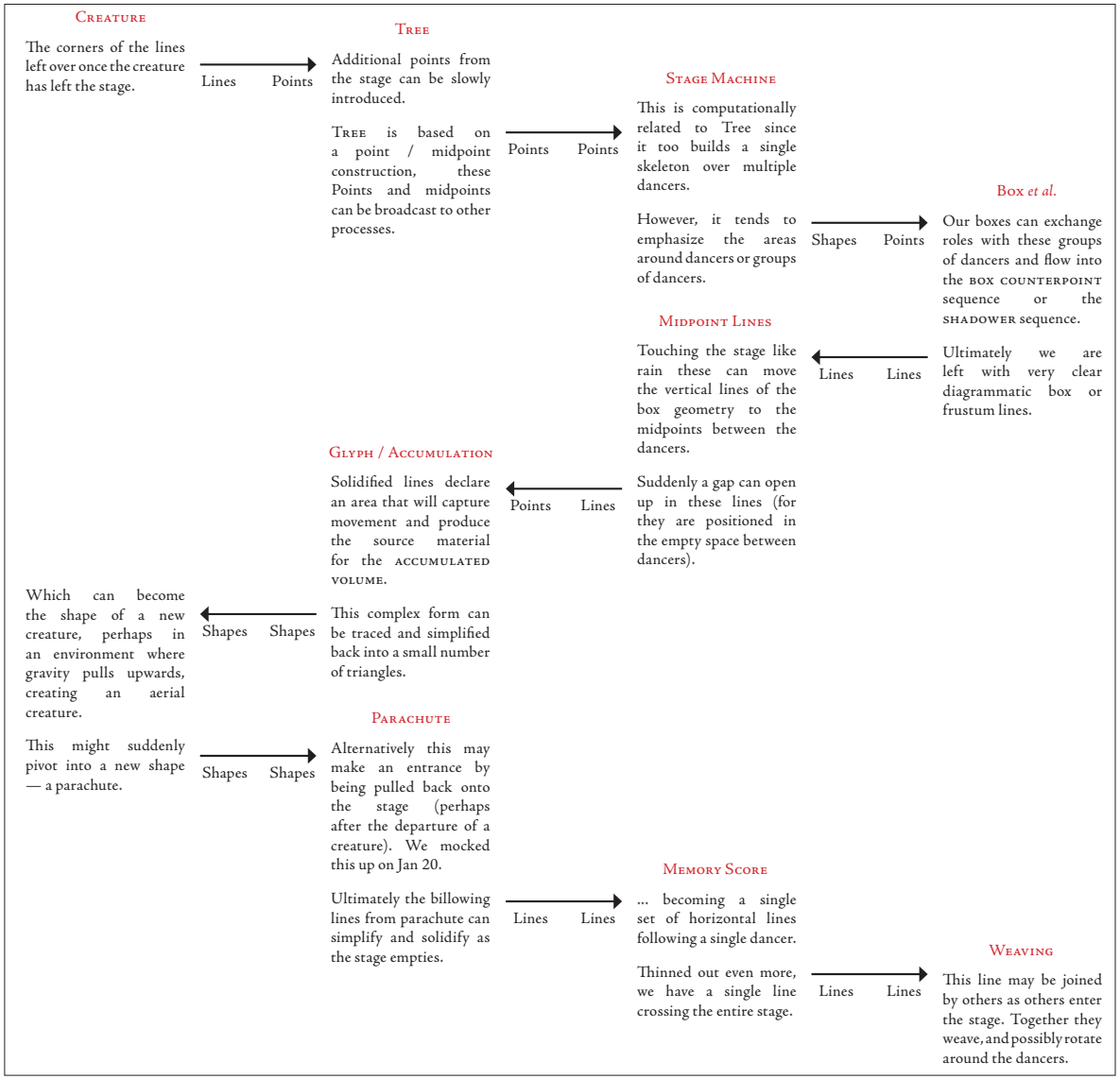


figure 111. An example "flow" through the *how long...* agent framework. This was the state of the piece for a series of workshops. The finished work adds a final return to the triangle here labeled "creature" — taken from documents shared between the collaborators.

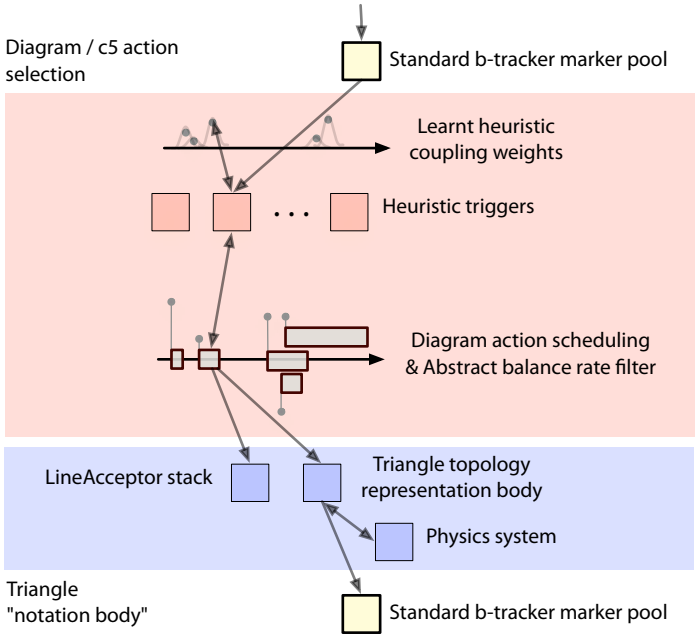


figure 112. The *triangle* agent diagram.

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.

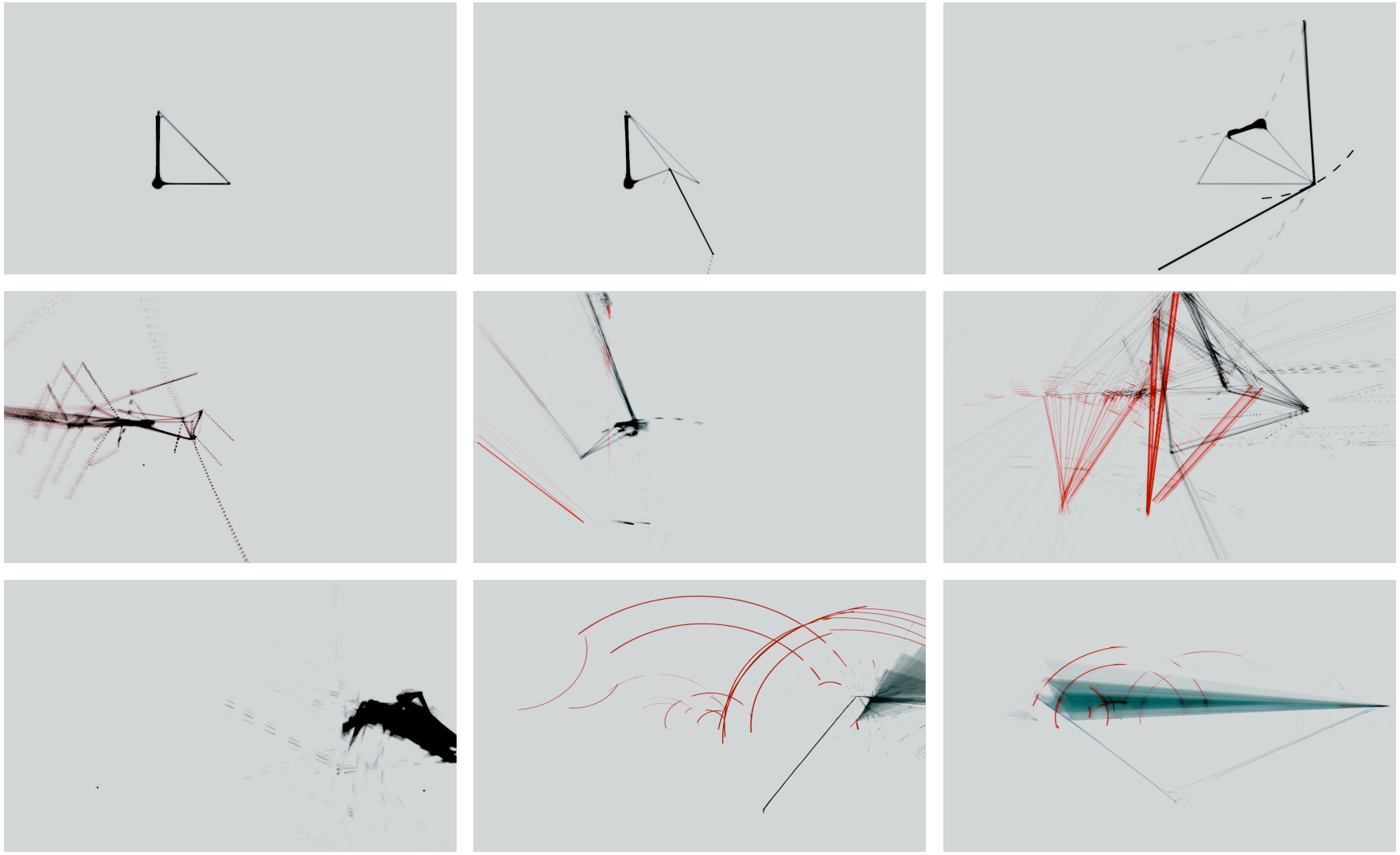


figure 113. The *triangle*. The first seven images are from various openings of the piece. A re-projection rendered line element picks out one path through the triangular framework. The last two images are from the "reprise" of triangle. (inverted).

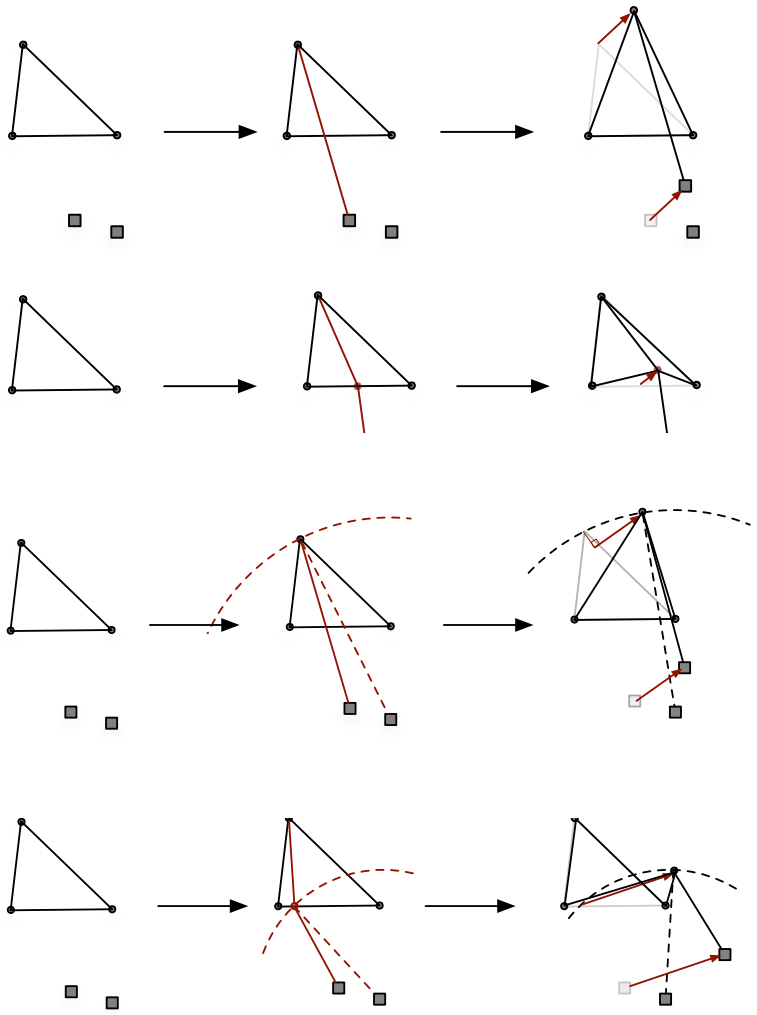


figure 114. The four classes of actions that *triangle* can take on one or two points: pull vertex, split-and-pull vertex, rotate vertex (constrained to be on a circle from another point) and split-and-rotate vertex.

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 that operation took place that 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 235* — 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

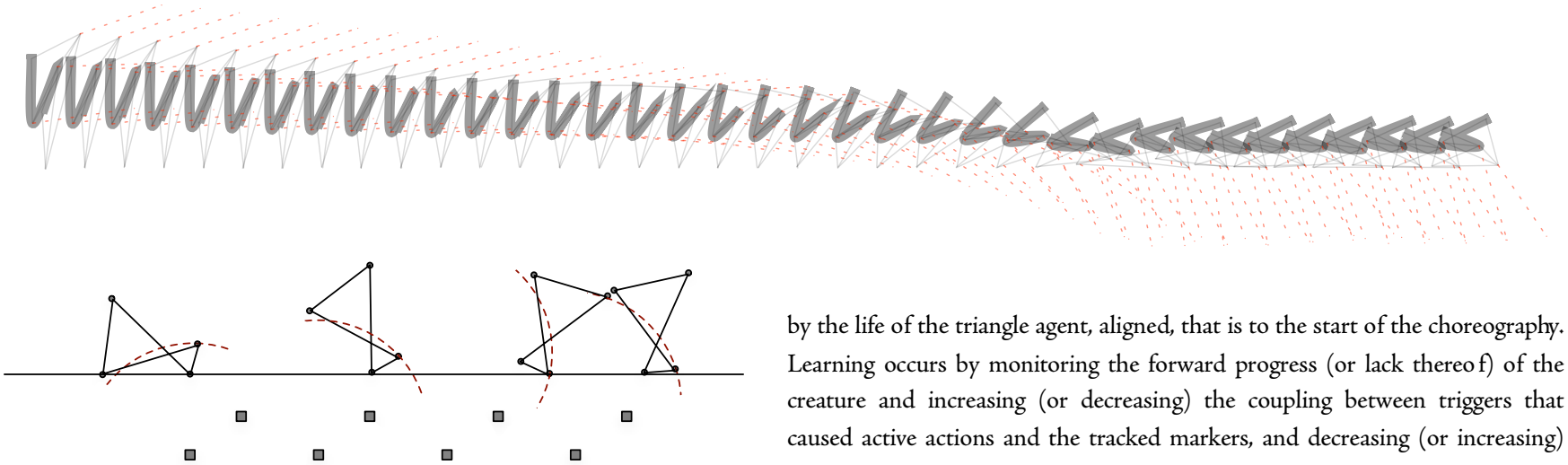


figure 115. Once actions have completed, traces of the operation remain connected to the body of *triangle*. This body is located in a simple physics simulation, thus, while it attempts to conserve edge length and face area it is simultaneously falling towards a hidden ground-plane.

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 an *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 235*, the context-tree's power for allowing duplication-with-modification of agents, *page 206*, and *Fluid's* utility in allowing complex systems to be tested and tuned live, *page 367*.

Indeed 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.

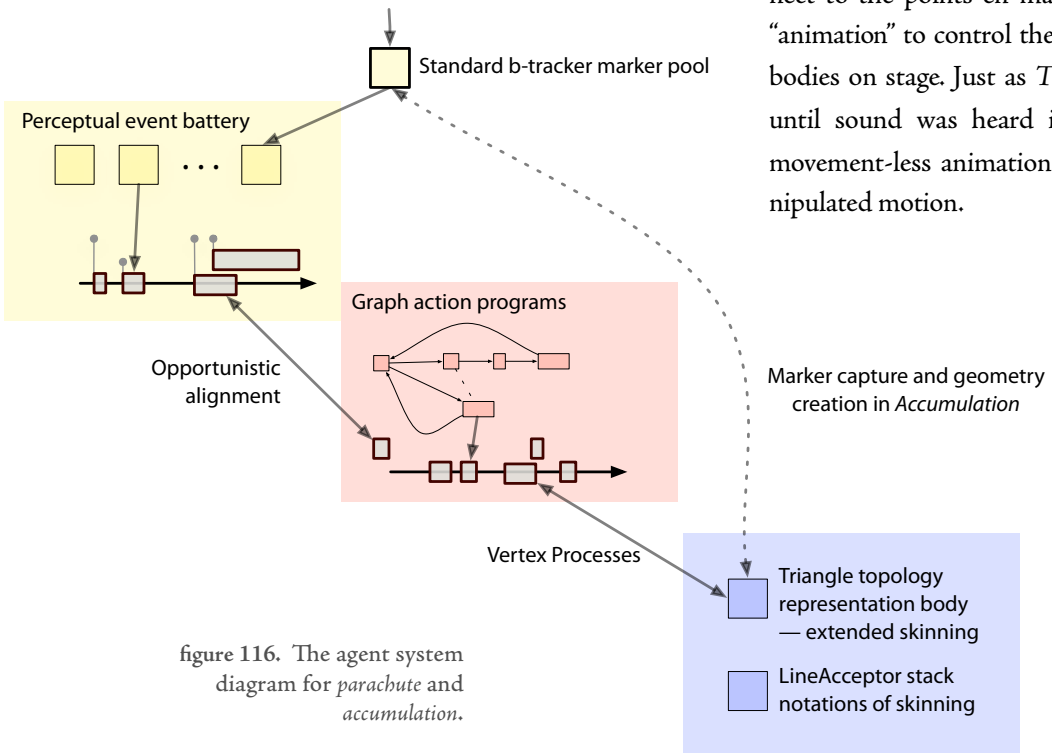


figure 116. The agent system diagram for *parachute* and *accumulation*.

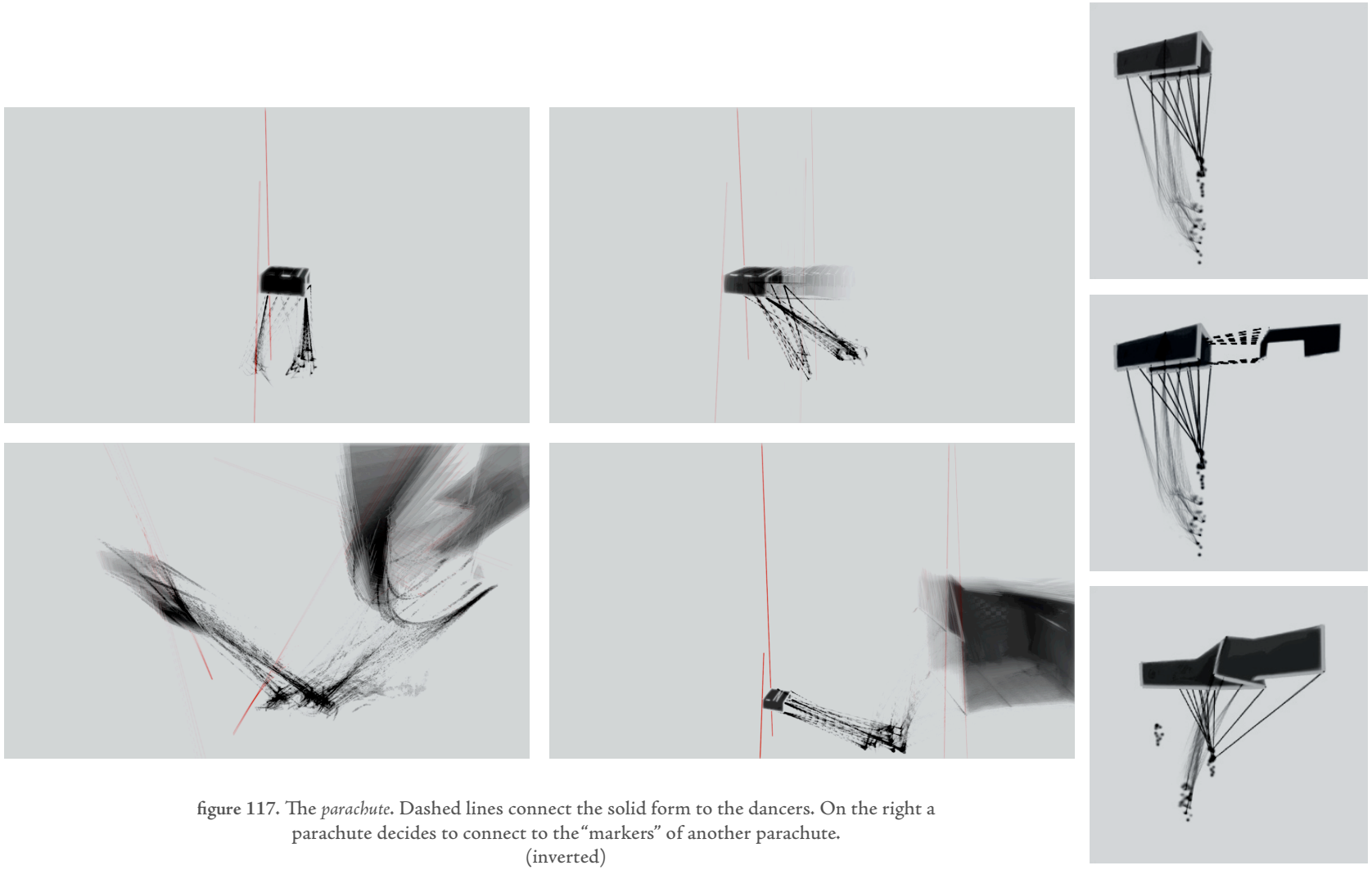


figure 117. The *parachute*. Dashed lines connect the solid form to the dancers. On the right a parachute decides to connect to the "markers" of another parachute. (inverted)

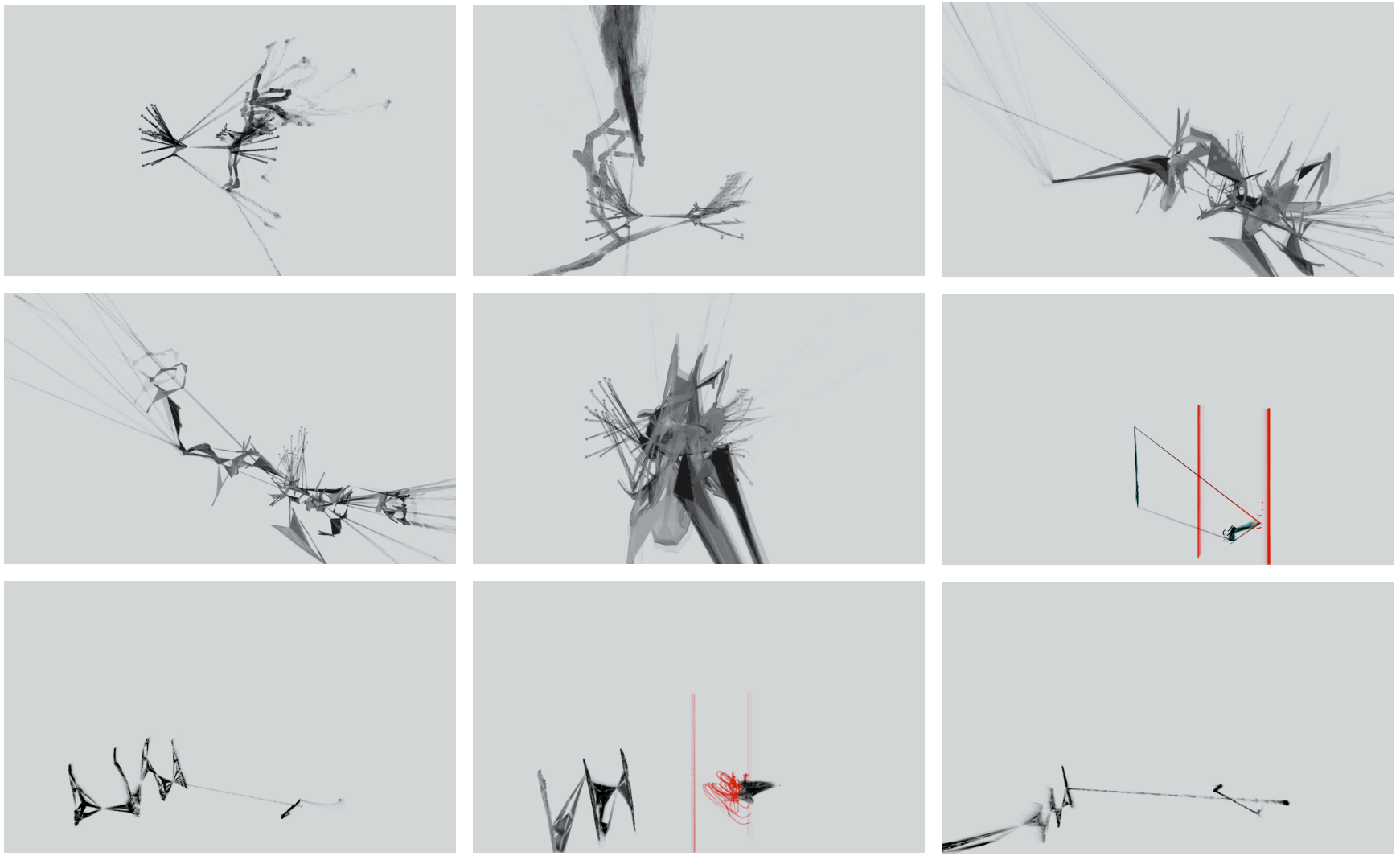


figure 118. The *accumulation*. The red lines indicate the sampling of movement material, and the creation of geometry, from the stage. (inverted)

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 marker 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:

$$v'_i \leftarrow \sum_j w_{i\to j} \left\lfloor v_i + (v_i - P'_j)\left(|v_i - P_j| \,/\, |v_i - P'_j| - 1\right)\right\rfloor / \sum_j w_{i\to j}$$

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:

$$v'_i \leftarrow \sum_j w_{i\to j} \left\lfloor P'_j + (v_i - P'_j) \cdot Q'_{i\to j} \cdot (v_i - P'_j) \right\rfloor / \sum_j w_{i\to j}$$

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, 145*. 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.

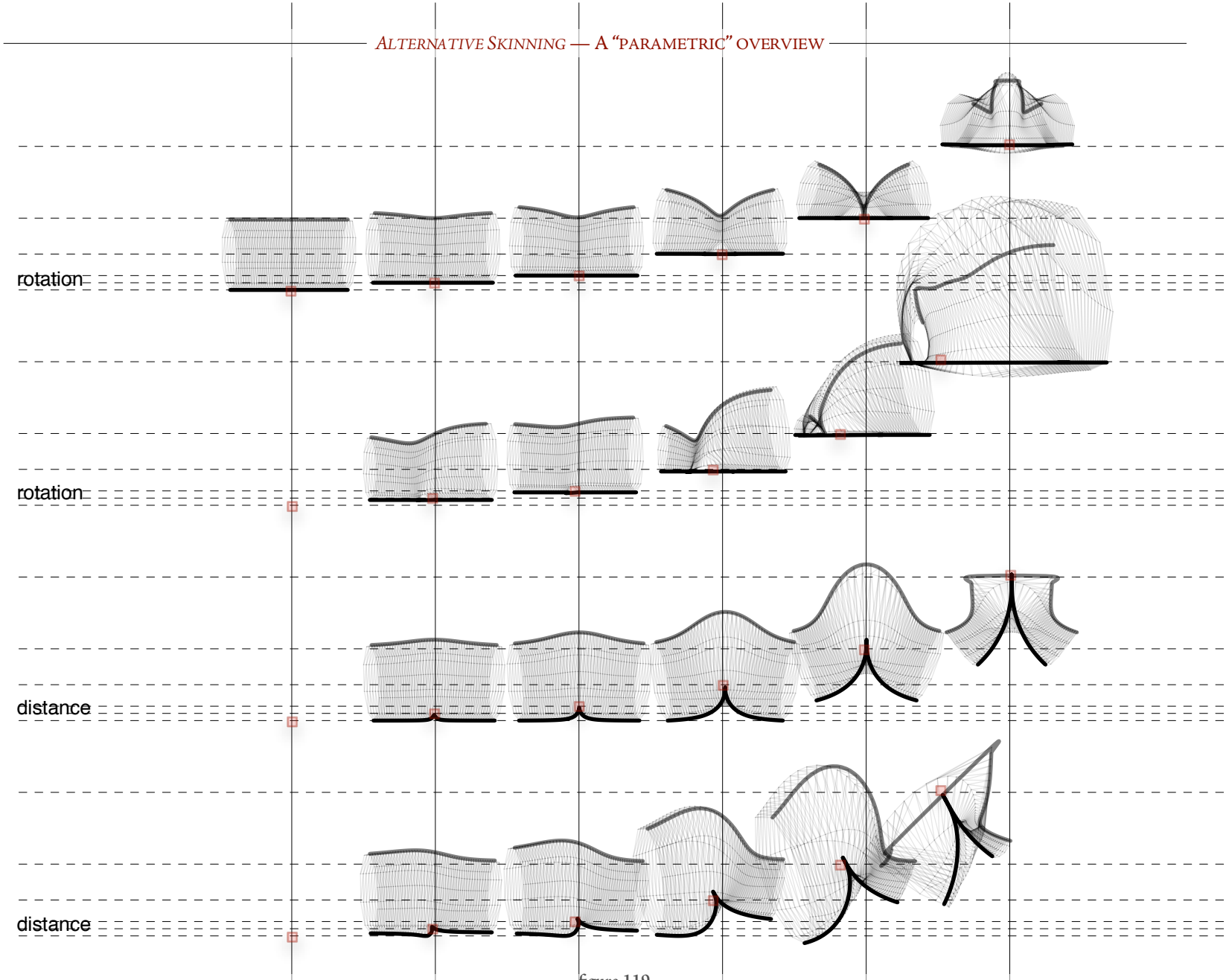


figure 119.
A single"skinning" control point applied to an initially undeformed cylinder — two classes of constraint are shown, in the upper level of each the control point —shown in red — is moved vertically upwards, in the lower, the control point is moved diagonally.

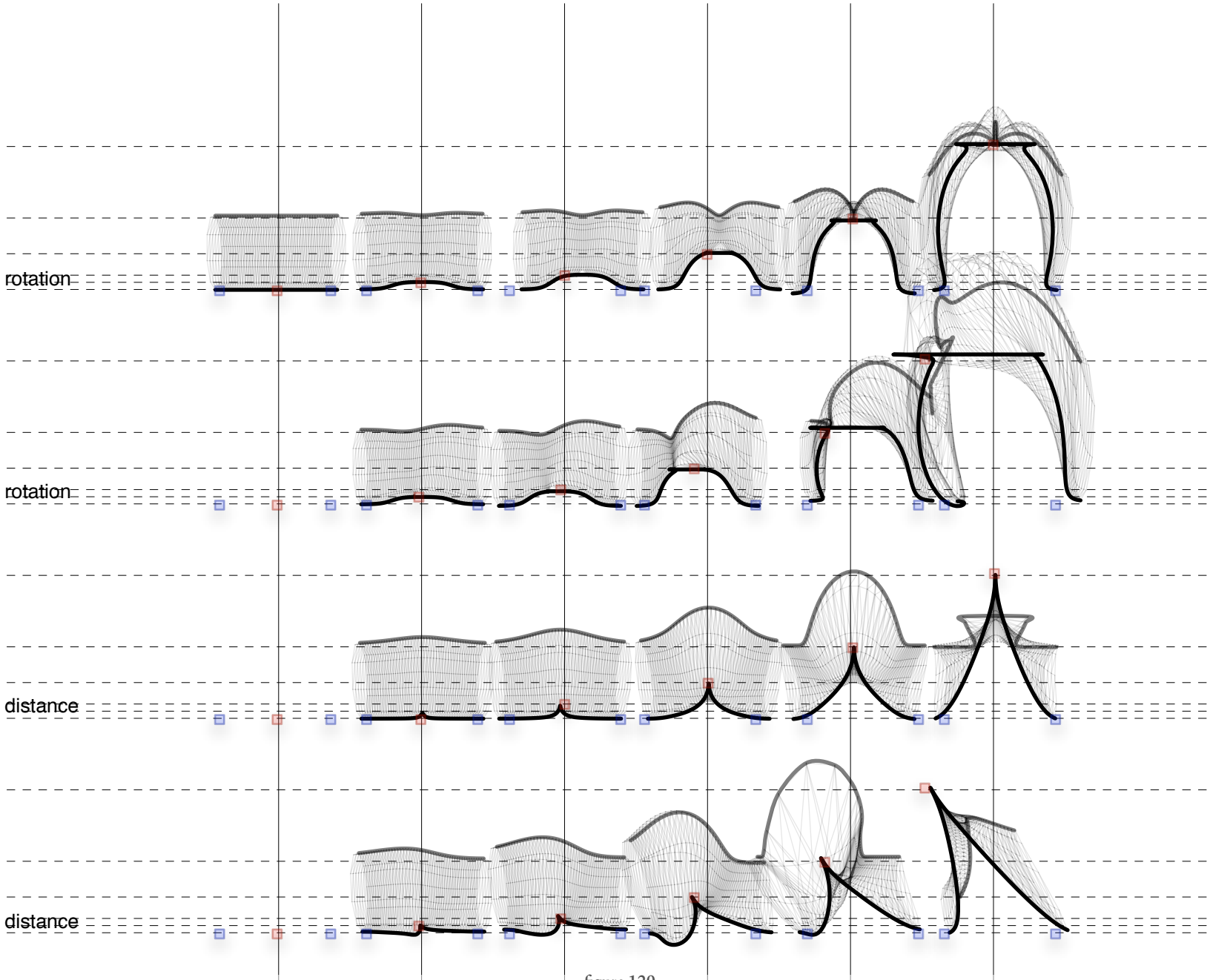


figure 120.
Similar to the previous figure, however here two additional "translation" constraints are added to each side of the original constraint. These act to keep nearby parts of the cylinder "undeformed". Of course, for more exotic constraints what undeformed implies can be quite complex.

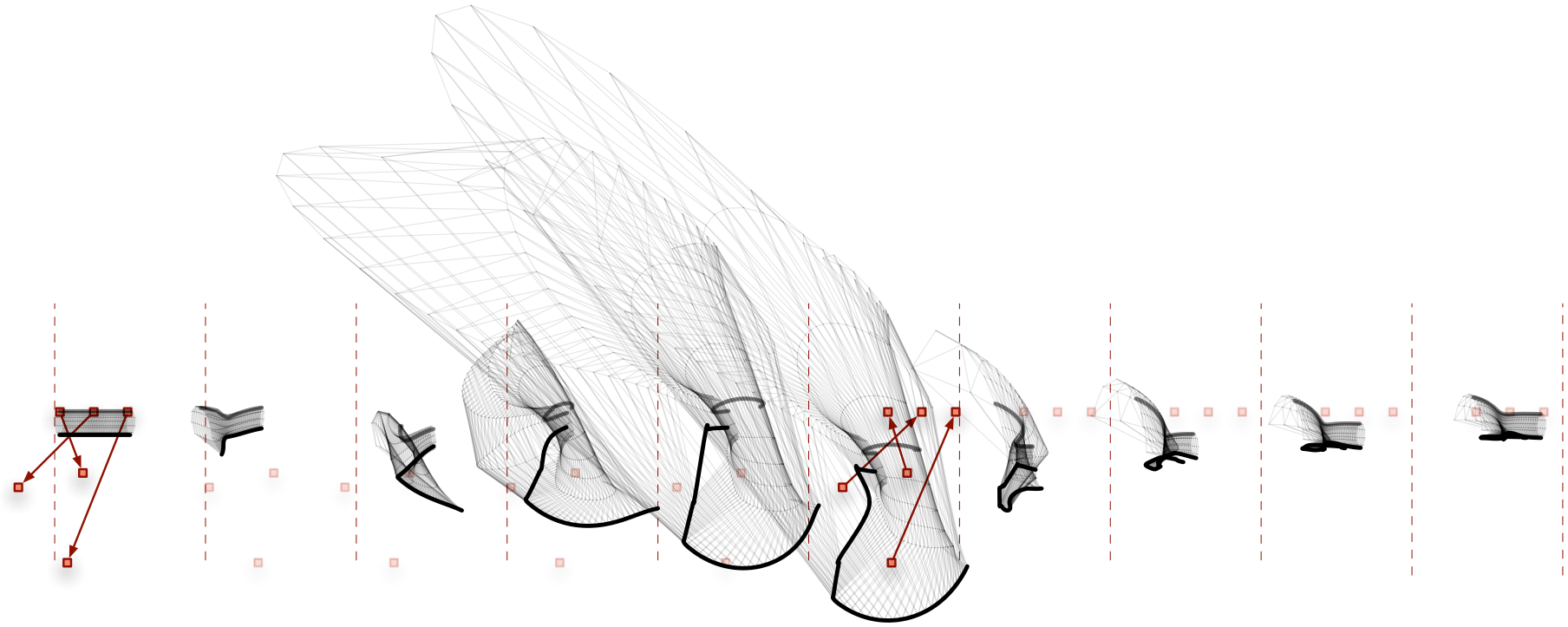


figure 121.
Because these skinning constraints are processes not operations, they necessarily produce animations not new static forms. The above figure shows an animation created by two successive, instantaneous movements of the control points. Because of the nested capture() and release() calls used 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 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 merge — allow the internal state of the constraining object to be duplicated and at some later point blended with another constraint.

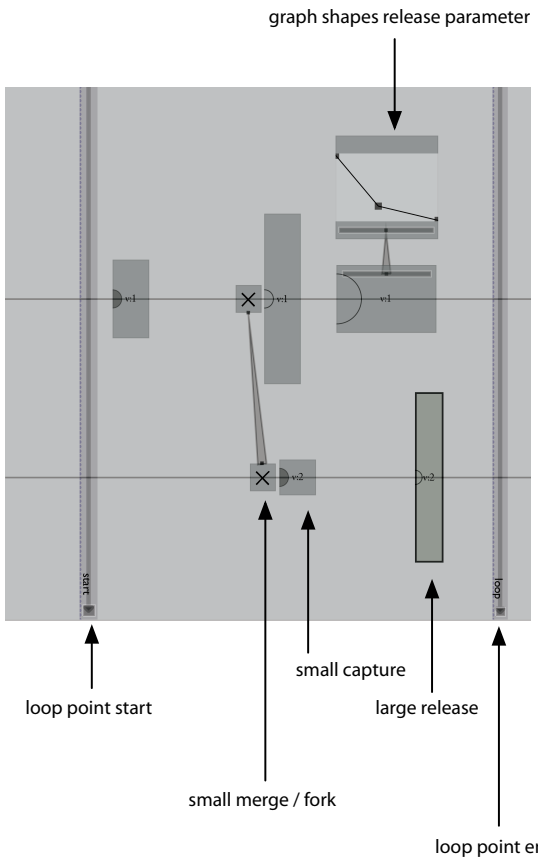


figure 122. The graphical environment *Fluid* (discussed in the next chapter) can be used to create these small, rhythmic pattern generators, scripting the application of the skinning and other mesh operations. These scores can be "unrolled" into diagram channels and opportunistically aligned with events gathered from the stage.

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 particular choreography, or losing the sense of the cell's identity? Is there there

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 127*, (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, 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).

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.

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.

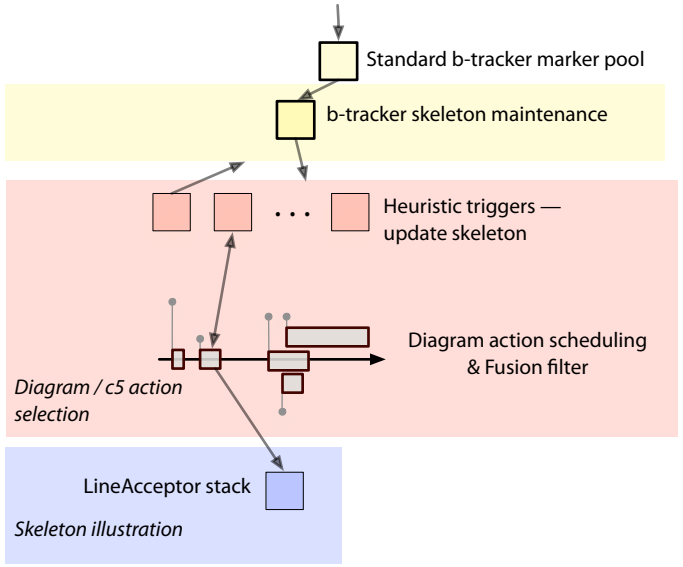


figure 123. The general agent system diagram for *tree* and *stage machine* and *forest fire*. Only the skeleton representation and the line acceptor stacks differ significantly — the agents share a common "base class".

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.

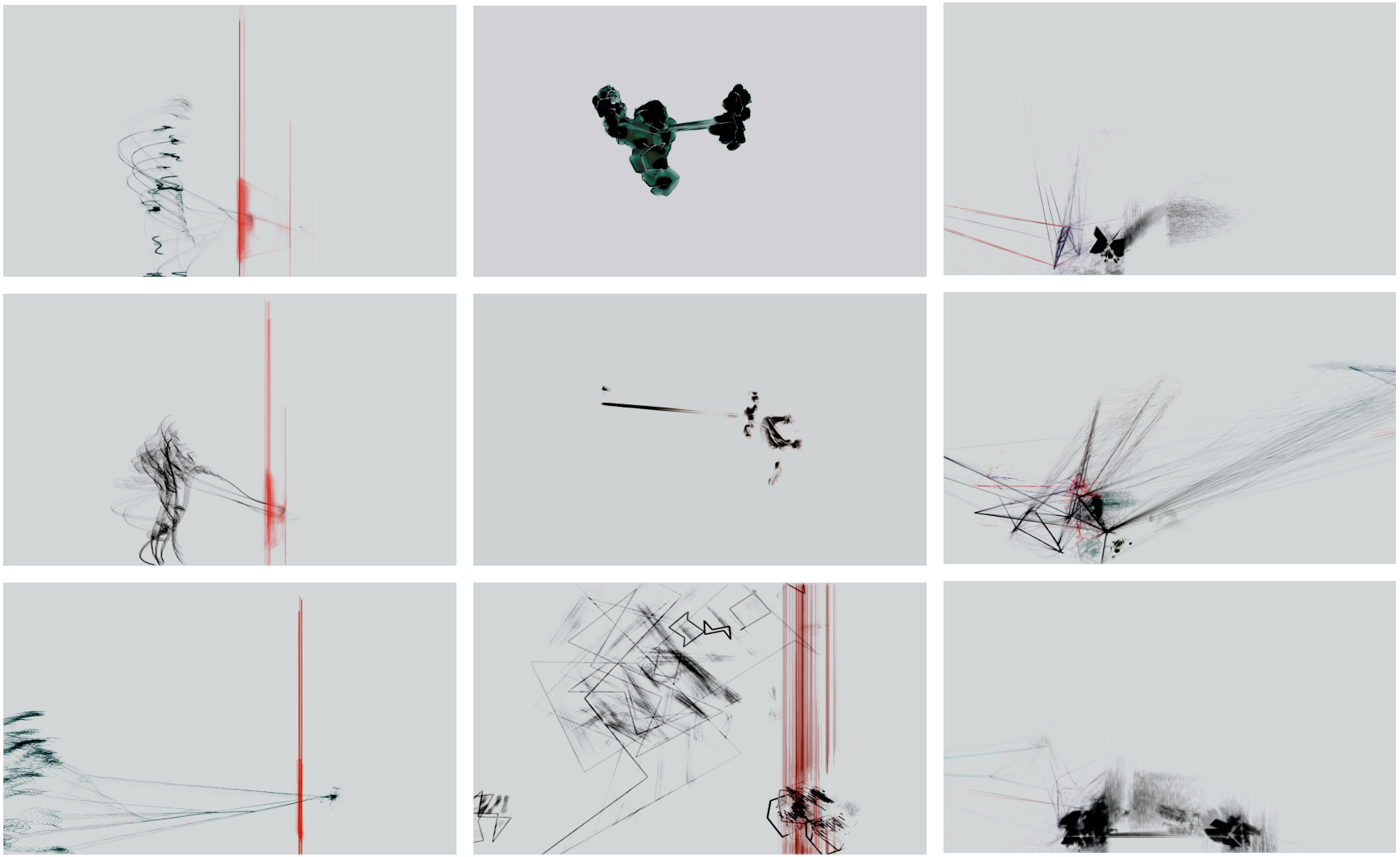
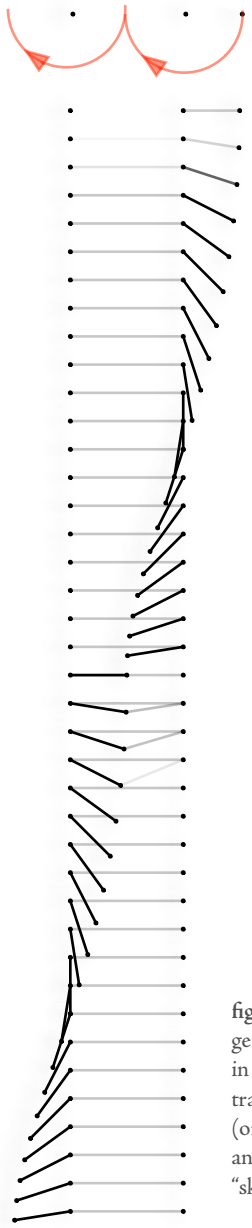


figure 124.
Left: *tree* reconnecting with material from the stage; middle: *stage machine* illustrated two different ways; right: *forest fire* and *stage machine* overlapping. (inverted).



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.

figure 125. 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.

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 :

$$a_{ij,t_0} = \frac{\left|m_{i,t_0} - m_{j,t_0}\right| \cdot \left(\left|m_{i,t_o} - m_{i,t_1}\right| - \left|m_{j,t_o} - m_{j,t_1}\right|\right)}{\left(\left|m_{i,t_o} - m_{i,t_1}\right| + \left|m_{j,t_o} - m_{j,t_1}\right|\right)^2}$$

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

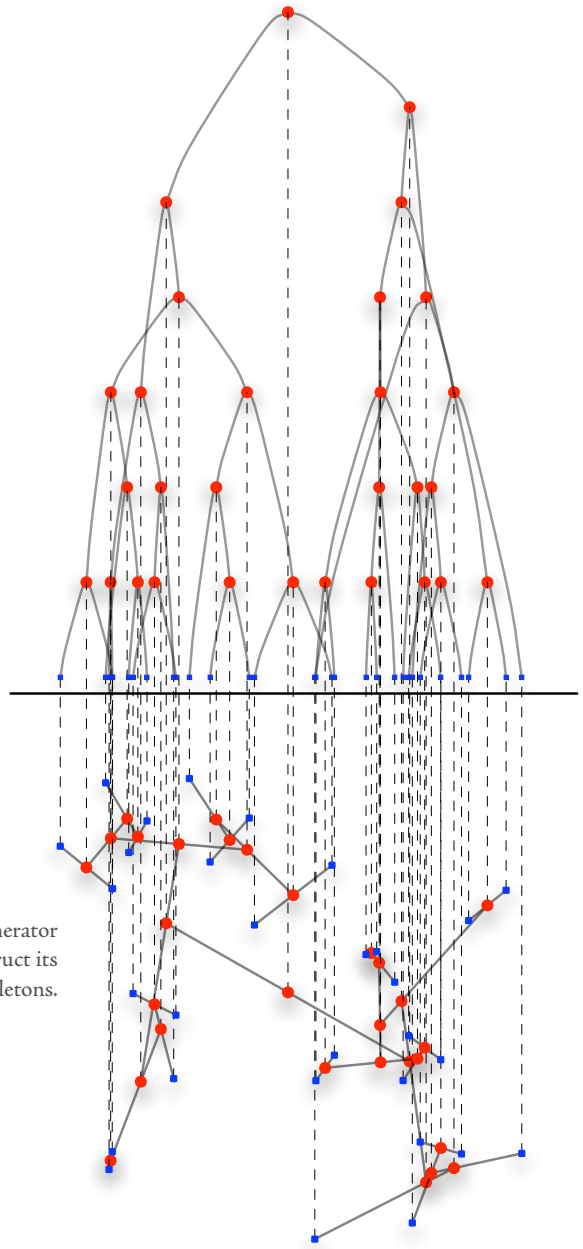


figure 126. The dendogram generator that *Tree* uses to construct its hypothetical skeletons.

the markers and eventually individual markers themselves might loose all reasonably 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 rehierarchi-

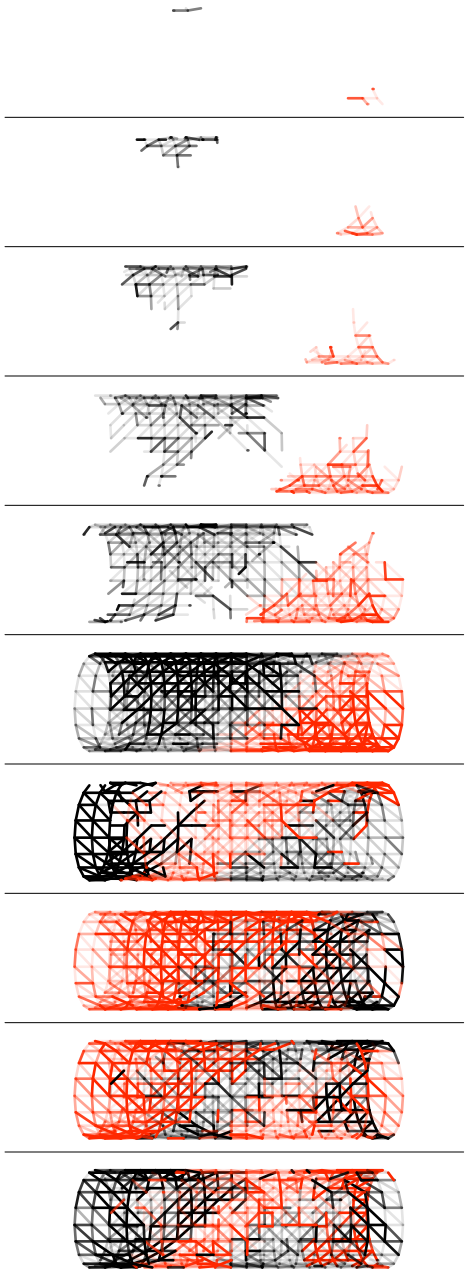


figure 127. The forest fire process,as applied to a static mesh. Two processes (fires), red and black compete for vertices (trees). By altering the propagation probability, the burn time and the regeneration rate a range of dynamics and interchanges can be created.

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.

calization of its perception system, *tree* also interprets its current skeleton as a filtering network on the underlying motion-capture data.

Each node bar 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 processes

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.

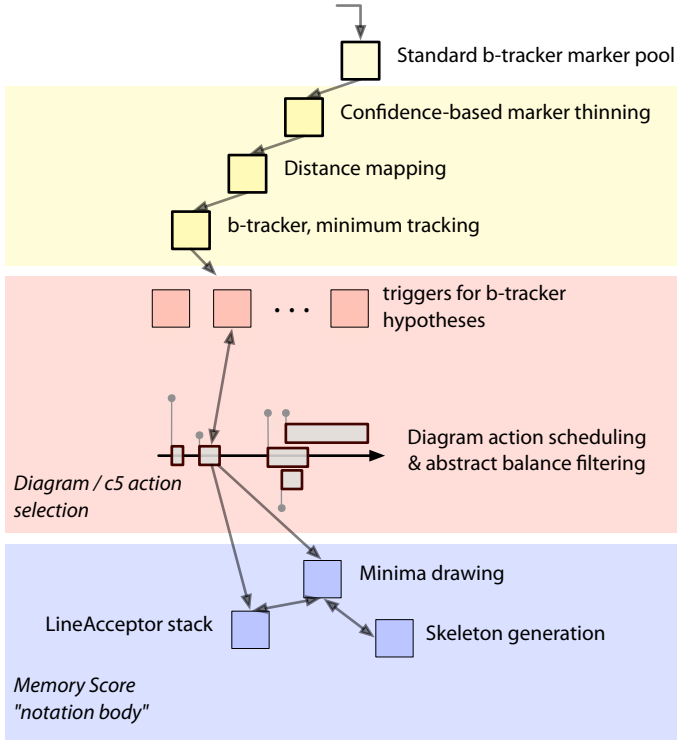


figure 128. The agent system diagram for *memory score*.

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.

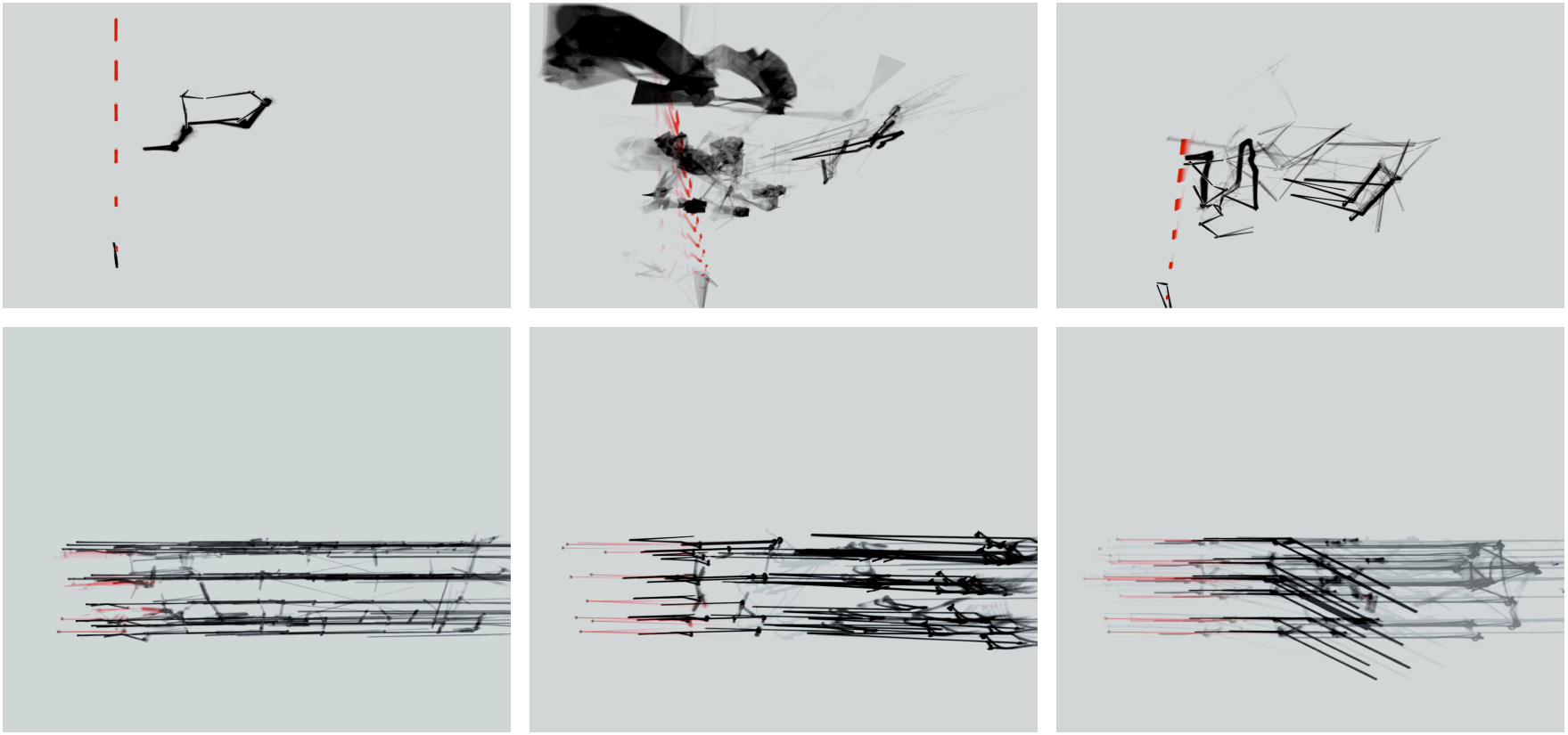


figure 129.
Memory Score, above: taken over the "tangle solo"; below, a more complete marker set. (inverted).

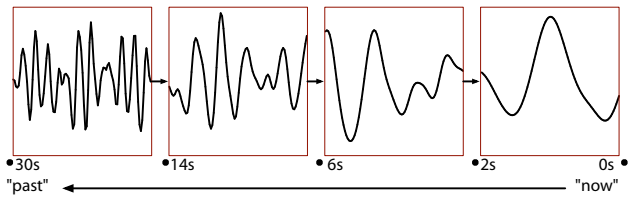


Figure 131. A cascaded, variable resolution signal for the distance mapping algorithm.

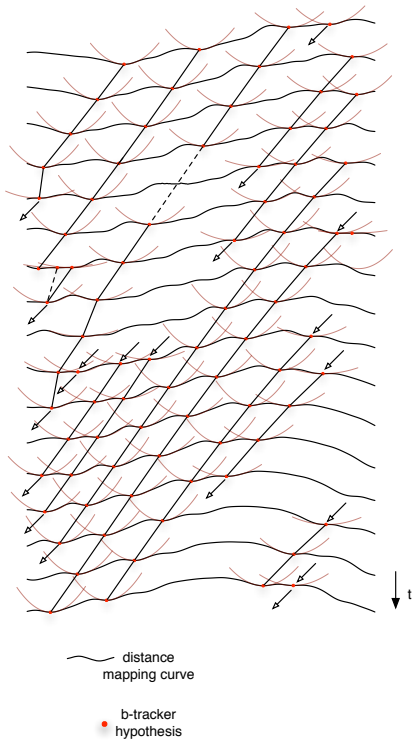


Figure 130. Tracked minima roll backwards through the output of the distance-mapping algrorithm.

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.

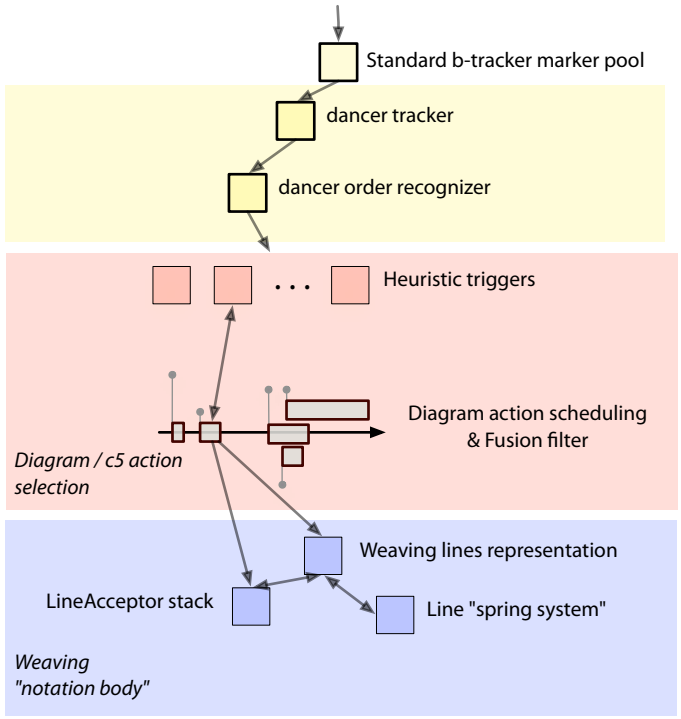


Figure 132. The agent system diagram for *Weaving*.

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 un-crossings 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.

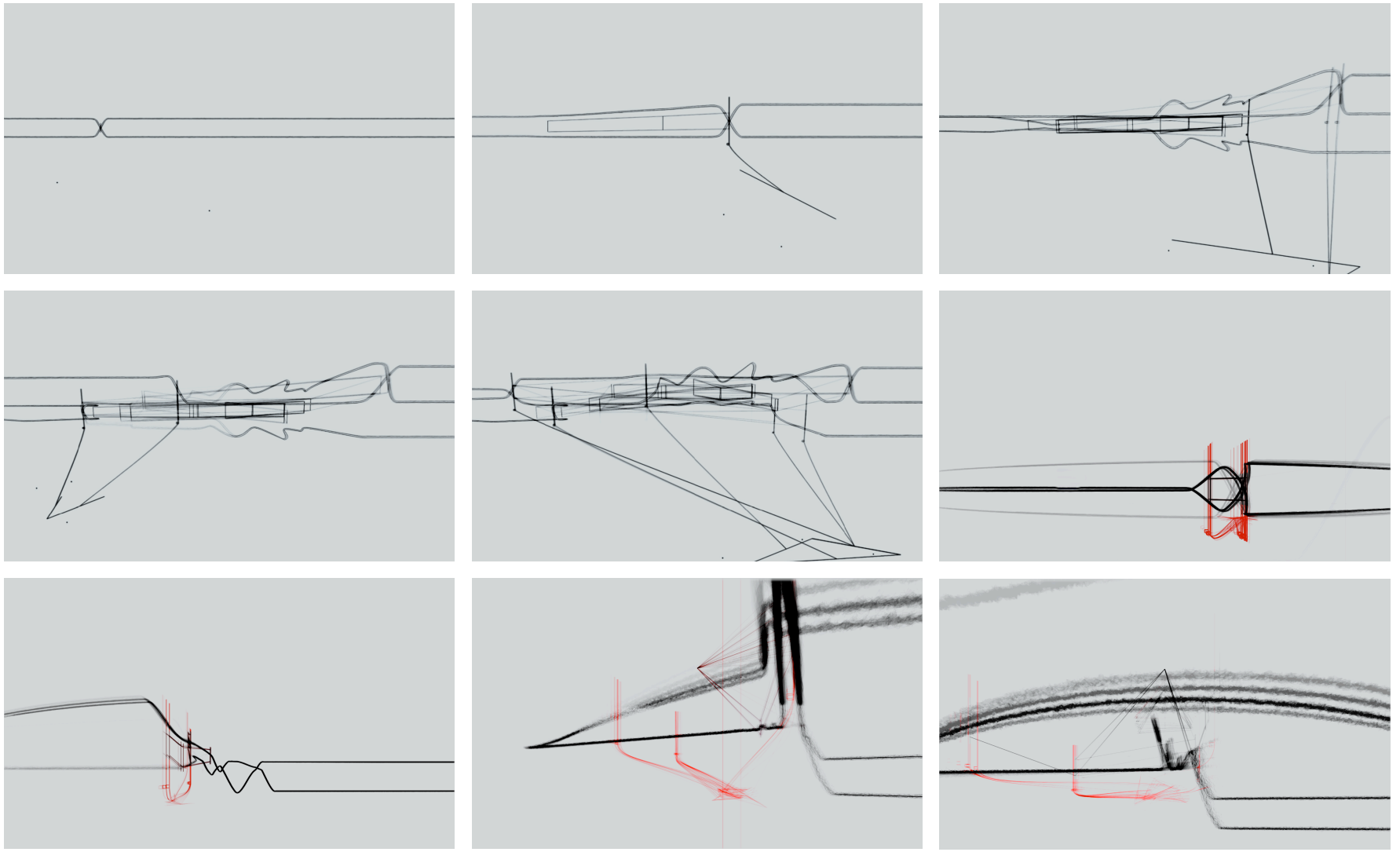


figure 133. *Weaving.* (inverted).

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. In doing so, there is 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, that 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 of a transformation of movement or instruction to move, their working practices accrete transformation, layering and burying 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. A goal 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 to not only with the dancers, but with 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 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 occultation 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".