Accessible Animation and Customizable Graphics via Simplicial Configuration Modeling

Research paper

Tom Ngo,[*] Doug Cutrell,[†] Jenny Dana, Bruce Donald,[‡] Lorie Loeb,[§] and Shunhui Zhu[#]
Interval Research Corporation, 1801 Page Mill Rd., Palo Alto, CA 94304-1216
{ngo,jdana}@interval.com

Abstract

Our goal is to embed free-form constraints into a graphical model. With such constraints a graphic can maintain its visual integrity—and break rules tastefully—while being manipulated by a casual user. A typical parameterized graphic does not meet these needs because its configuration space contains nonsense images in much higher proportion than desirable images, and the casual user is apt to ruin the graphic on any attempt to modify or animate it.

We therefore model the small subset of a given graphic's configuration space that maps to desirable images. In our solution, the basic building block is a simplicial complex—the most practical data structure able to accommodate the variety of topologies that can arise. The configuration-space model can be built from a cross product of such complexes. We describe how to define the mapping from this space to the image space. We show how to invert that mapping, allowing the user to manipulate the image without understanding the structure of the configuration-space model. We also show how to extend the mapping when the original parameterization contains hierarchy, coordinate transformations, and other nonlinearities.

Our software implementation applies simplicial configuration modeling to 2D vector graphics.

Keywords: Animation with constraints, geometric modeling, weird math, WWW applications.

1 Introduction

1.1 Goal

Constraints often provide coherence and creative freedom. That principle has been exploited pervasively in computer graphics since the days of Ivan Sutherland's Sketchpad system [18], in forms ranging from user-specified constraints such as those found in CAD systems [3] and dataflow systems [9] to constraint-inference engines [10] Even bitmap-based tools such as Painter [14] empower the user's creativity by constraining the modifications she can make to an image.

We are interested in how to re-parameterize graphical models so that they are constrained enough to hold their visual form when modified or animated by a casual user. Target applications include creativity software, dynamic clip art, cartoons, performance-based avatars, and charts. Most existing parameterized representations (e.g., 2D vector graphics, NURBS surfaces, and CSG) are not constraining enough to meet our requirements. Consider the spline-based example in Figure 1. In that diagram, the blue cube represents the configuration space of the spline. Each configuration is a set of numerical values for the parameters of the spline (the coordinates of its control points). Most configurations correspond to meaningless scribbles. A rare few generate humanoid shapes.

Our task, then, is *configuration modeling*: representing a subset of the configuration space of an existing parameterized graphic. Because this subset can have arbitrarily topology, our modeling scheme must not impose any a priori restrictions on topology. Moreover, the resulting re-parameterization might not map in any convenient way to a user's "mental parameterization." We therefore seek to shield the user from knowing the structure of the parameterization by allowing for direct manipulation of the graphical object. We refer to such direct manipulation as *tugging*.

Configuration modeling is a fundamental open problem in computer graphics; the need to represent a subset of a graphical object's configuration space is pervasive.[1] Although our test implementation is designed for 2D spline-based graphics, we expect simplicial configuration modeling to be applicable to other forms of parameterized graphics, including morphs, physically based models, and so forth.

1.2 Structure of paper

Section 2 provides a relatively qualitative description of a modeling pipeline appropriate for configuration modeling. Quantitative and other algorithmic details are deferred for the appendices. Section 3 discusses how the re-parameterization used in configuration modeling can be used in concert with domain-specific techniques, primarily to incorporate hierarchy. Section 4 briefly describes selected pieces of content made using

[*] Corresponding author.
[†] Present address: NetLens, 10975 N. Wolfe Rd., Cupertino, CA 95014, doug@netlens.com.
[‡] Present address: Dartmouth Computer Science Dept., 6211 Sudikoff Lab., Rm. 113, Hanover, NH 03755, brd@cs.dartmouth.edu.
[§] Present address: Stanford Computer Science Dept., Stanford, CA 94304, lorie@graphics.stanford.edu.
[#] Present address: Integral Development Corp., 156 University Ave., Palo Alto, CA 94301, shunhui.zhu@integral.com.

[1] The technique might also be applied to objects that are not graphical, such as parameterized sounds.

our authoring system. Section 5 compares our technique with related work, employing the modeling abstractions introduced in earlier sections. Section 6 recapitulates our main contributions and lists possible avenues of future work.

2 Qualitative description

2.1 Configuration-modeling pipeline

Any configuration-modeling technique can be described in terms of the modeling pipeline depicted in Figure 2. The rightmost component in that pipeline, the *image space*, represents the space of all possible images that can be rendered on the display device. The middle component, the *configuration space*, represents the freedom in the given parameterized image: a configuration is a set of parameter values, one for each parameter. Its dimensions are therefore called *configuration parameters*, or simply *parameters*.

The mapping from the configuration space to the image space is the *rendering map*. It determines how the parameter values in a configuration are interpreted to generate an image on the display device; thus, it subsumes all processes normally associated with modeling, rendering, and display. Together, these two spaces and the intervening map define a parameterized graphic—an object whose re-parameterization is the goal of our work.

The leftmost component of the pipeline is the *state space*. The mapping from it to the configuration space is the *re-parameterization map*. The goal in configuration modeling is to define a state space and re-parameterization map so that the range of the map is a set of desirable configurations.

Thus, a state determines[2] a configuration under the re-parameterization map, and a configuration determines an image under the rendering map. Most computer graphics research, including work on modeling, rendering, and display, is concerned with the rendering map. This paper describes an approach to defining the re-parameterization map.

2.2 Nature of the configuration parameters

As we have described it, configuration-modeling tasks usually involve continuous configuration parameters: geometry (e.g., 2D or 3D coordinates of control points or lengths of line segments), colors, and even physical parameters such as forces and masses in a relaxed mesh. In most such cases, it is appropriate to define a re-parameterization map that is a function.

A significant twist on configuration modeling can arise when continuous configuration parameters represent coordinate transformations. Consider a gedanken experiment in which the original parameterized image is a closed 2D spline. The goal in this experiment is to build a configuration model that represents a complete human walk cycle in profile, a complete crawl cycle, and any gait between walking and crawling. The state space is isomorphic to a square: one dimension represents progress in the gait cycle; the other, interpolation between walking and crawling. Assuming that a walk cycle covers more horizontal distance than a crawl cycle, the path through the state space depicted in Figure 3 produces a net horizontal translation. Thus, the addition of a global horizontal translation is required to avoid representing configurations that differ from each other only by translation. Moreover, the horizontal-translation variable is not a function of state: it experiences hysteresis.

When a discrete parameter is present, the re-parameterization map can map patches of the state space to values of the discrete parameter.

2.3 Structure of the state space

In our solution, the state space is represented by a data structure that is topologically general and computationally practical: the simplicial complex [15]. For convenience in expressing certain common relationships, we allow for a cross product of simplicial complexes.

Combinatorially, a *simplicial complex* S can be specified[3] by a set V of symbols, and subsets of V chosen so that none is a subset of any other. Each of those subsets is called a *simplex*: a point, a line segment, a triangle, a tetrahedron, or a higher-order simplex, depending on the number of symbols in the subset. The standard embedding of that combinatorial object is then the subset Λ of R^V such that every λ in Λ lies in a simplex.[4] A vector λ lies in a simplex s if each of its coordinates lies in the unit interval [0,1], the coordinates sum to unity, and all coordinates corresponding to symbols outside the simplex are zero. The coordinates corresponding to symbols inside the simplex are called *barycentric coordinates*. We use the combinatorial representation of a simplex as shorthand for its standard embedding.

Suppose, for example, that the simplicial complex S_1 comprises the set of symbols {o,p,q,r} and simplices {{o,p,q},{q,r}}. It consists of a triangle {o,p,q} and a line segment {q,r}. Using the symbol ordering (o,p,q,r), the points $(^3/_4,{}^1/_8,{}^1/_8,0)$ and $(0,0,{}^1/_3,{}^2/_3)$ both lie in the standard embedding of this simplicial complex because $(^3/_4,{}^1/_8,{}^1/_8)$ lies in {o,p,q}, and so forth. Thus, a simplicial complex can be used as a data structure for generating interpolation weights; the simplices restrict certain groups of weights from being simultaneously non-zero.

We define the state space Ψ to be a cross product of k simplicial complexes: $\Psi = S_1 \times S_2 \times \ldots \times S_k$. We use the term *state* to denote an element of the state space. Thus, a state ψ lies in the state space Ψ if it can be expressed as a tuple $(\lambda_1;\lambda_2;\ldots;\lambda_k)$ such that each λ_i lies in the corresponding simplicial complex S_i. We use the term *vertex* to refer to a "corner" of the state space, i.e., a state $\psi=(\lambda_1;\lambda_2;\ldots;\lambda_k)$ in which each λ_i has exactly one coordinate equal to 1.

Suppose, for example, that a state space is equal to $S_1 \times S_2$, where S_1 is as defined above, and S_2 comprises the set of symbols {u,v,w} and the simplices {{u,v},{v,w}}. Using the symbol ordering (o,p,q,r; u,v,w), the points $(^1/_4,{}^1/_2,{}^1/_4,0;\ {}^3/_7,{}^2/_7,{}^2/_7)$ and (0,0,1,0; 0,1,0) both lie in the state space. The latter is a vertex.

[2] We will later introduce hysteresis into the re-parameterization map, so the word "determine" is used loosely.

[3] Only top-level simplices—simplices that are not sub-faces of other simplices—are specified by the user and explicitly represented in the implementation.

[4] The notation R^V denotes a Euclidean space whose |V| dimensions are named according to the symbols in V.

We use the term *zone* to refer to a linear region of the state space, i.e., a tuple of simplices, with one simplex taken from each factor in the cross product of simplicial complexes. Thus, the state space $S_1 \times S_2$ contains four zones.

2.4 Re-parameterization map

We use re-parameterization maps of various levels of complexity, depending on the nature of the configuration parameters. In its simplest form, a re-parameterization map on a k-factor state space is specified by an arbitrary configuration for each vertex in the state space, and is k-linear within each zone (linear within each factor simplex). Thus, a re-parameterization map for the state space $S_1 \times S_2$ would be specified by twelve independent configurations: one for each symbol pair in the cross product $\{o,p,q,r\} \times \{u,v,w\}$. Configurations admitted by the map would be of the form

$$x' = \Sigma_{a \in \{o,p,q,r\}} \Sigma_{b \in \{u,v,w\}} \lambda_1[a]\, \lambda_2[b]\, x[a][b]$$

where square brackets denote indexing by the symbols in the set $\{o,p,q,r\} \cup \{u,v,w\}$, $x[a][b]$ denotes the configuration associated with symbols a and b, and the state $\psi = (\lambda_1; \lambda_2)$ lies in the state space $S_1 \times S_2$. In our current software implementation, those twelve configurations are supplied by an artist with a drawing tool. In other systems they might be automatically generated.

A re-parameterization map can also take more complicated forms if the configuration space contains some discrete parameters, or some parameters control coordinate transformations.

In the case of a discrete parameter, the state space can be tessellated into regions, each of which maps to a different value of the parameter. We have implemented a form of such a discrete map in which the regions are zones and the discrete parameter controlled the front-to-back permutation ordering of the primitives in a 2D vector graphic.

Appendix A.2 describes the case of a continuous parameter that represents a coordinate transformation and therefore can undergo hysteresis.

2.5 Forward process: "driving"

We use the term *driving* to describe any process in which a sequence of states, generated either automatically or interactively, is used to compute a sequence of configurations, and hence an animation. Driving can be used in many ways.

Clip motions. A sequence of states can be used to animate any simplicial configuration model whose state space is compatible[5] with it, and therefore can be used as a clip motion. The term "clip motion" has been used previously different way. In Litwinowicz' Inkwell system [13], for example, a clip motion is a collection of animated Coons patches that can be textured differently to generate different characters. By contrast, two simplicial configuration models can use the same clip motion if they have isomorphic state spaces, even if they bear no geometric resemblance.

Factor synchronization. If two configuration models have state spaces whose cross products have one or more factors in common (e.g., $S_1 \times S_2$ and $S_2 \times S_3$), then the barycentric coordinates associated with the two instances of the shared factors (e.g., S_2) can be synchronized. This is useful when two models contain some logical dependency; for example, the position of a drawn shadow can be synchronized with the position of a light source.

Factor sharing is also critical in abating the exponential explosion that can occur when a model contains many degrees of freedom. In a model with hierarchy, it is typical for different graphical elements to depend on different—but overlapping—sets of hierarchical levels. Consider a human figure: trousers could depend on the knee and hip degrees of freedom; a T-shirt, on the hip and shoulders; and a scarf, on the shoulders and neck. Without factor sharing, each element would need to be re-parameterized with a state space that includes factors associated with every joint. With it, each element's state space needs to include only the factors on which it depends.

Algorithmic behavior. Some or all of the factors in a model can be driven algorithmically. We have implemented, for instance, a behavior that executes cartoon-like squash and stretch, taking into account the magnitude and direction of flight, and impacts with the ground. This behavior can be applied to any model that contains a squash-and-stretch factor in its state space that is isomorphic to the one expected by the behavior. Because it interfaces with the model at the level of state—not geometry—it is sufficiently general to be applied to both a soft, round beach ball and a rigid block of wood. Moreover, the same behavior code can be used either interactively (for performance or play) or offline (for authoring).

2.6 Inverse process: "tugging"

Tugging means inverting the re-parameterization map so that a user or some exogenous process can manipulate the model through some of its configuration parameters, rather than its state. The system answers each requested change in configuration with a change in state that matches the request as closely as possible. The configuration parameters being manipulated are often an (x,y) pair;[6] we refer to them collectively as the *tug point*.

The algorithms for tugging, described in Appendix A.1, address two principal challenges.

1. The re-parameterization map on a zone is often ill-conditioned. We invert it safely using the Moore-Penrose inverse [7].
2. At boundaries between zones, the re-parameterization map contains discontinuities in the first derivative. Our

[5] For example, if two simplicial configuration models can use the same clip motion if they have state spaces that are isomorphic to each other, even if their appearance is radically different. The condition for compatibility is weaker than isomorphism: the clip motion can also be used for a model whose state space is a superset of the aforementioned isomorphic state spaces.

[6] The number of configuration parameters in the tug point could be more or less than two. For example, a mixing station or armature could control many parameters simultaneously. In a 3D environment, the tug point might often be a (x,y,z) point. Non-geometric parameters such as colors can also be tugged.

algorithms handle these discontinuities seamlessly, so that inter-zone boundaries—and therefore the structure of the state space—can be transparent to the user.

Because configuration parameters are generally more intuitive to use than states, tugging is the primary mechanism by which both users and software interact with the graphic. Like driving, tugging can be used in many ways.

Direct manipulation. The user specifies desired configuration changes by dragging the tug point with a pointing device. This style of interaction obviates the need for (but still permits the creation of) separate graphical user-interface elements. It is also well suited for performance-driven animation.

Combination with factor sharing. When factor sharing makes objects in a scene mutually dependent, tugging can be used to control any of the objects.

Simple software control. The graphic is regarded as a software object whose tug point is its interface. This technique permits an algorithmic behavior to be designed for reusability. We have implemented, for instance, a dynamic behavior that simulates a mass in a viscous medium, attached to a user-translatable anchor point through a damped spring. It can be used for any graphic with an (x,y) tug point and a 2D translation, regardless of state-space topology.

Factor locking. Barycentric coordinates associated with one or more of the factors in a state space can be constrained not to change during a tug. In our system, the author can arrange for certain factors to be locked whenever a given point is tugged.

Hybrid driving and tugging. One or more factors can be driven algorithmically while others are tugged. This is useful in arranging for a combination of interactivity and autonomous behavior.

3 Hybridizing with domain-specific techniques

Configuration modeling is best suited for describing relatively free-form interdependencies that are difficult to express algebraically. When constraints are more easily described by domain-specific techniques such as articulated-figure kinematics, the domain-specific techniques are preferable. In addition, a mechanism to make one model depend on another improves model reusability: for example, a model of an eye might be made once and used with many faces.

One approach to addressing these needs is to use configuration modeling and an existing domain-specific technique independently. For example, to develop a human character in a 2D vector-based system, one might use standard forward and inverse kinematics for articulated figures to define a complete skeleton, and express fragments of clothing using independent re-parameterized models, each in the local coordinate system of a different rod in the skeleton. In this approach, clothing would not automatically deform in response to skeletal movements.

Another approach is incorporate curvilinear interpolation into the re-parameterization map, which is linear on each simplex as we have presented it. For instance, to mitigate the foreshortening effects characteristic of linear interpolation in Cartesian coordinates, one might make the re-parameterization map polynomial or transcendental on each simplex. We have considered employing simplicial splines with differential constraints at boundaries. Librande and Poggio have successfully developed a curvilinear re-parameterization technique that employs radial basis functions on a hypercube [12].

Instead of either approach, we place parameterized, domain-specific coordinate transformations in the original model. Thus, the coordinate transformations are executed in the rendering map, but because they are parameterized, their behavior can be influenced during re-parameterization. Structural hierarchy is permitted in several ways: control points, entire models, and coordinate transformations can depend on other coordinate transformations. This technique simultaneously addresses the needs for reusability and for domain-specific constraints. In addition, we find that it permits the use of re-parameterization to break rules imposed by the domain-specific constraints.

To amplify the last point, consider a 2D vector-based arm drawn as one spline around two rotational joints. Parts of the spline are represented in a forearm coordinate system; others are in an upper-arm coordinate system; yet others are in absolute coordinates. Rotating the two joints without re-parameterizing any spline coordinates causes the curve to move roughly as the outline of an arm, but with artifacts: the spline folds incorrectly at the elbow.

Re-parameterization allows the artifact to be removed. In addition, it permits the artist to arrange for the shoulder to dislocate artfully by adjusting translational joint parameters[7] in the extreme poses. Specifically, one might create a state space with topology {{elbow1,elbow2}, {elbow2,elbow3}} × {{shoulder1,shoulder2}, {shoulder2,shoulder3}}: simple bilinear interpolation in each of four zones generated from a 3×3 grid of example configurations. By arranging for the nine configurations to differ only in their shoulder- and elbow-joint parameters, one would obtain the coarse movement described above. The artifact repair and joint dislocation would be brought about by adjusting the spline coordinates in each of the nine configurations.

4 Results

We implemented an authoring system for the re-parameterization of 2D vector graphics via simplicial configuration modeling. We also implemented a number of smaller applications in which novice users could manipulate models created in the authoring system. The authoring system and applications were written for the Win32 operating system and the Microsoft Foundation Classes. For smooth real-time animation, a Pentium-class processor with a clock speed of at least 266 MHz is required. Rendering—not tugging—is the slow step.

Figure 4 shows selected content authored in our system.

[7] Our domain-specific coordinate transformation has four parameters. It represents a translatable rubber sheet that stretches along a preferred axis whose orientation is variable. When these transformations are chained, the origin of each rubber sheet is expressed in the coordinate system of the preceding transformation but the orientation and stretch are in absolute coordinates.

The fern exemplifies free-form constraints. It was authored using twenty-one drawings of the fern in different positions. Tugging its tip elicits graceful undulations. A flattened diagram of its single-factor state space is in Figure 5.

The beach scene demonstrates factor synchronization. It has five factors: two for the sun position, one for the shoreline shape, one for the castle shape, and one for the motion of the waves. The first four factors are linear chains of three, two, three, and four symbols each; the fifth is a single triangle. Each object in the scene depends on a different subset of the factors. They are interrelated in several ways; for example, the position of the sun, its reflection, and the castle's shadow are mutually dependent and all can be tugged.

The Trapeze Guy illustrates how a simple behavior (the mass-spring dynamical system described in Section 1.6) can be reused. The Trapeze Guy's state space is a ring of eight symbols (which represents rotation about the trapeze), crossed with a linear chain of five symbols (which represents, roughly, the distance of the character's feet from the trapeze). The user-translatable anchor point of the mass-spring system is attached to the trapeze; the dynamically controlled mass, to a tug point near the character's feet. His body swings about as the user moves the trapeze. In the spirit of free-form constraints, an open red mouth and beads of sweat added to some of the forty authored configurations add to the comedy.

The model of Bart Simpson,[8] our most complex piece, contains extensive factor synchronization and hybridization using the modified form of articulated-figure kinematics described in Section 3. Its fourteen separate simplicial configuration models represent body parts such as arms, legs, eyes, and eyelids. They share subsets of the twelve factors, with the greatest amount of sharing used for axial rotation as well as left-right and part-to-part coordination of the eyes. One of the more interesting factors is the one that governs the shape of the mouth: it contains four triangles and two tetrahedra, assembled into a kite-like structure.

Articulation in each limb and the attachment of facial parts to the head are both accomplished using the hybrid scheme described in Section 3. In both cases, the visual integrity of the cartoon depended critically on the re-parameterization component of the hybrid scheme, without which the coordination of body parts is only approximate. In addition to having visual integrity, the image of Bart stays on model.

5 Related work

In computer graphics, simplicial complexes have been used principally for modeling 3D geometry. Hoppe et al.. [8] exploited the well-understood topological properties of simplicial complexes to regulate changes to mesh representations of surfaces. Edelsbrunner [5] used a multi-resolution approach (α-shapes) to track scale-dependent topological changes in the Delaunay triangulation of a multi-scale point-data set.

We use simplicial complexes to meet a goal much closer to that of Librande and Poggio [12], whose work may be described in the language of configuration modeling. Viewed from that perspective, their technique prescribes a state space that is a hypercube, and therefore cannot represent topological holes. Representational power resides in the re-parameterization map, whose form is a superposition of radial basis functions, whereas ours is only piecewise linear. Thus, their system is geometrically flexible but—in comparison to ours—topologically restricted. (Holes are important, for example, when rotational degrees of freedom are involved. Indeed, recent work by Rademacher on view-dependent deformations [17] essentially employed simplicial configuration modeling.)

Gleicher and Witkin [6] influenced us to ensure that any simplicial configuration model is, without any special effort on the part of the artist, responsive to direct manipulation. Pai [16], among others, has applied similar concepts to robot control; his methods are based much more on half-space penalty functions.

Also related are constraint-based systems that require the user to specify constraints explicitly, either algorithmically or through a more graphical interface. Such systems are ideal when the desired constraints are relatively easy to state, as in industrial CAD [3], diagramming tools [1], and 3D animation tools that permit entry of algebraic constraints [2]. By reducing cognitive burden on the user, the work of Kurlander and Feiner on constraint inference [10] challenges our distinction between explicit and free-form constraints.

Techniques for multi-target interpolation [4,11] are complementary to simplicial configuration modeling in the sense that each represents a new class of parameterized models to which the structured interpolation implied by simplicial configuration modeling may be applied. In fact, the present work is part of a growing interest in structured interpolation between examples, either digitally captured or created by an artist.

6 Epilogue

We have identified the challenge of configuration modeling, which we believe to be an important open problem in computer graphics. To address this challenge, we have proposed to use a modeling primitive based on the simplicial complex. This choice leads to topological generality.

We have shown how to run the maps from state space to image space both forward (by driving) and in reverse (by tugging). We have identified a number of ways in which driving and tugging lead to economies related to reusability of code and content. We have shown why the need for hysteresis arises in configuration modeling and have proposed techniques for obtaining it. We have demonstrated how domain-specific coordinate transformations can be used in harmony with configuration modeling in a manner that exploits the strengths of both.

Today's simplicial configuration models are characterized by low simplex counts and labor-intensive authoring—much like the first polygonal models in 3D graphics. If configuration modeling addresses a genuine need, one might expect to see further developments along lines analogous to the ones that have permitted polygonal models to grow in complexity by orders of

[8] We are in the process of obtaining permission to publish this likeness of Bart Simpson and the Corn Flakes mascot, Cornelius. Until we are able to obtain such permission, this review submission should not be redistributed.

magnitude. These might include semi-automated authoring using high-level primitives and capture of configuration models from video sources.

Acknowledgments

We thank Subutai Ahmad, Neal Bhadkamkar, Michele Covell, Frank Crow, Yan-Bin Jia, Scott Klemmer, Andrew Kunz, Bud Lassiter, Golan Levin, and Meg Withgott for their thoughtful contributions to this work.

A Appendices

A.1 Tugging algorithms

This appendix describes how to invert the re-parameterization map. Given a requested change in one or more configuration parameters, the goal is to compute a state change that satisfies the request as closely as possible. We describe cases in order of increasing complexity. Following Gleicher and Witkin [6], we use local solutions instead of global ones to provide temporal continuity in animation.

One factor, one zone. In a single-factor model that contains one simplex, the re-parameterization map is linear. Let x be a vector whose coordinates are the configuration parameters to be changed, and let Δx be the desired change. A state is said to *improve upon* the current state if it maps to a configuration x' for which $(x'-x)\cdot\Delta x > 0$.

Let λ be the state in barycentric coordinates. To constrain the barycentric coordinates to sum to unity, we use the redundant coordinate system ρ, defined by $\lambda_i = \rho_i + (1-R)/k$, where k is the number of symbols in the simplex and $R = \Sigma_i\rho_i$, summed over all symbols in the simplex.

The state change is then $\Delta\rho = J^{\ddagger}\,\Delta x$, where the components of the Jacobian J are the partial derivatives of x with respect to ρ, and the symbol $\ddagger$ denotes Moore-Penrose inversion, which handles rank-deficient matrices by giving special treatment to singularities [7].

If the requested state change $\Delta\rho$ would cause the state to exit the simplex, travel is halted at the simplex boundary. At the simplex boundary, one or more barycentric coordinates are zero. The Jacobian computation is repeated with the corresponding symbols in the simplex omitted from the computation, i.e., constraining the state to lie in the subface. If a boundary of that subface is encountered, the procedure continues in subfaces of decreasing dimensionality until a subface of dimensionality zero is encountered.

Multiple factors, one zone. When k factors are involved, the re-parameterization map is k-linear. Let x and Δx be defined as above, but let each ρ_i in the tuple $(\rho_1; \rho_2; \ldots; \rho_k)$ be the state coordinates from factor i. The state change is then $[\Delta\rho_1^T|\Delta\rho_2^T|\ldots|\Delta\rho_k^T]^T = [J_1|J_2|\ldots|J_k]^{\ddagger}\,\Delta x$, where each Jacobian J_i is defined as above with respect to the corresponding ρ_i while keeping ρ_j fixed for all $j\neq i$. In a multilinear map, rectilinear movements produce linear effects, but diagonal movements can produce polynomial effects because each Jacobian J_i depends on every ρ_j for all $j\neq i$. Therefore, local optima exist.

Factor locking. To lock factor i, i.e., prevent ρ_i from changing during tugging, we merely omit ρ_i and J_i from the equation given above. Multiple omissions lock multiple factors. When one factor is driven while another is tugged, we interleave steps of driving and tugging.

One factor, multiple zones. This is the piecewise linear case. In contrast with the linear case, a simplex boundary can be the portal to one or more neighboring simplices. When the state arrives at a simplex boundary, we first identify neighboring simplices, i.e., ones that share the subface in which the state resides. We remove from consideration each neighboring simplex[9] that contains no states that improve upon the current state. (Because the re-parameterization map is linear on the neighboring simplex, it is sufficient to test each vertex in the neighboring simplex that lies outside the current subface.) If zero neighboring simplices remain, tugging proceeds in the subface, as in the linear case. If one neighboring simplex remains, tugging proceeds in that simplex.[10]

The subcase in which multiple neighboring simplices remain is one for which we have discussed and implemented various heuristics, but that we have not yet encountered in practice. We leave as an open problem the development of a universally acceptable way either to choose an appropriate neighboring simplex, or to control the topology of the state space so that this subcase cannot arise.

Multiple factors, multiple zones. This is the piecewise multilinear case. It raises additional issues that are also open problems. As in the piecewise linear case, the only challenges that are not present in the purely multilinear case arise at boundaries between zones. If the state encounters a zone boundary that is a simplex boundary in only one of the factors, the decision reduces to the one-factor, multiple-zone case. At a zone boundary that is a simplex boundary in more than one factor—this case does arise in practice—the decision is more complicated. The remainder of this section discusses that decision.

Recall that in a state space formed from the cross product of k simplicial complexes, a zone is a k-tuple of simplices, with one simplex taken from each factor. In the situation under consideration, the state lies on a subface in more than one of those simplices. Another zone is considered to be a neighbor if at least one of the simplices is replaced by a neighboring simplex that shares one of those subfaces.

One difficulty arises because the re-parameterization map associated with a zone can be polynomial for moves that are not rectilinear. The simple test used in the piecewise linear case to determine whether a neighboring simplex should be eliminated does not necessarily work. A neighboring zone can contain states that improve upon the current state even if none of its vertices do. We have therefore replaced the global test over vertices by a local test: we compute the anticipated state change from Δx in each neighboring zone and eliminate any neighboring zone from which the state would immediately exit.

The second difficulty is that the test given above may admit multiple neighboring zones. Again, the situation does not occur in practice but a universally acceptable solution (or way to guarantee that the situation is never encountered) would be desirable for the sake of completeness.

A.2 Re-parameterization map with hysteresis

This appendix shows how to define a re-parameterization map with

[9] Moving into such a simplex cannot produce any movement in the direction of the requested configuration change.

[10] Infinite looping is prevented by eliminating a simplex from consideration for the duration of a single tugging step once it has been exited.

hysteresis. We have developed re-parameterization maps with three differing levels of hysteresis.

A *conservative* map is one for which traversing a closed loop in the state space always produces a closed loop in the configuration space. Outside of this appendix, all re-parameterization maps described in this paper are conservative.

A *semi-conservative* map is one for which traversing a closed loop in the state space can be guaranteed to produce a closed loop in the configuration space only if the loop in state space crosses no zone boundaries.

A *non-conservative* map is one for which traversing a closed loop in the state space can never be guaranteed to produce a closed loop in the configuration space.

We describe semi-conservative and non-conservative maps. Each of these developments requires to departures from the mechanisms put forth outside this appendix.

First, we modify the definition of a simplicial complex. A simplicial complex is normally defined as a union of simplices in which each simplex is an open set. Instead, we define each top-level simplex as a closed set, deliberately creating redundancy at shared subfaces. Under this definition, every state on a shared subface has one or more siblings that have the same barycentric coordinates but are associated with different top-level simplices.

Second, we distinguish between the *relative* values of a configuration parameter (which are present in the range of the map) and the *actual* values (which are supplied to the rendering map and may differ from the relative values).

Semi-conservative map. Suppose x is a continuous configuration parameter for which we wish to define a semi-conservative map. In each zone, we permit the author to specify an arbitrary, relative value of x at every vertex. Thus, a vertex can have a different relative value of x in every zone of which it is a member. We define the re-parameterization map to be multi-linear within each zone, just as in the conservative case, but possibly discontinuous at zone boundaries. As the state moves continuously within a zone, the infinitesimal changes in the relative value of x are accumulated into the actual value of x; but when it moves from one zone to another, the non-infinitesimal changes in the relative value of x are ignored.

A semi-conservative map could be used, for example, to control the rotation of a 2D wheel on or near a high-friction 1D surface. The parameters of the wheel are (x,y,θ), where x and y are the position of the wheel's center and θ is its rotation angle. For x and y, the re-parameterization map is conservative and defined so that a zone boundary maps to a horizontal line just above the surface. For θ, the map is semi-conservative. It is defined so that the relative value of θ is constant in the zone above the surface, and linearly related to x in the zone on the surface. The wheel exhibits the physically correct hysteresis that results from rotating only when moved horizontally while in contact with the surface.

Non-conservative map. Suppose, now, that x is a continuous configuration parameter for which we wish to define a non-conservative map. In each zone, we permit the author to specify an arbitrary relative-value difference Δx at every edge (ordered vertex pair). When the state moves within a zone from barycentric-coordinate vector λ to barycentric-coordinate vector λ', we accumulate into the actual value of x the following quantity:

$$\Delta X(\lambda,\lambda') = \Sigma_v \Sigma_{v'} \lambda(v) \Delta x(v,v') \lambda'(v'),$$

where each sum is over all vertices in the zone, each $\Delta x(v,v')$ is an relative-value difference supplied by the author, and $\lambda(v)$ is a scalar quantity extracted from the vector λ by multiplying together all components of λ associated with symbols in the symbol tuple v. As with the semi-conservative case, we do not change the actual value of x when crossing a zone boundary.

This procedure has a number of attractive properties. First, it produces hysteresis. Second, $\Delta X(\lambda,\lambda')$ is a smooth function of λ and λ'. Third, when moving from one vector (v) to another (v') it produces a change in x equal to the one specified by the author, i.e., $\Delta x(v,v')$. Fourth, under some circumstances[11] we have been able to show that a linear path through the state space can be executed in any number of smaller steps without affecting the total change in x. Fifth, when the values of $\Delta x(v,v')$ are conservative (i.e., $\Delta x(v,v') + \Delta x(v',v'') = \Delta x(v,v'')$ for any v, v', and v''), the procedure reduces to the semi-conservative case.

A non-conservative map could be used to provide the hysteresis called for in Figure 3.

References

[1] Aldus Corp. IntelliDraw. Computer Program (1992).

[2] Alias|Wavefront Corp. PowerAnimator 9. Computer Program (1998).

[3] Autodesk Inc. AutoCAD 2000. Computer Program (2000).

[4] S.E. Chen, L. Williams. "View interpolation for image synthesis". In Proc. SIGGRAPH 1993, 279-288 (1993).

[5] H. Edelsbrunner, E. P. Mücke. Three-dimensional alpha shapes. ACM Transactions on Graphics, 13(1):43-72 (1994).

[6] M. Gleicher, A. Witkin. "Through-the-lens camera control". In Proc. SIGGRAPH 1992, 331-340 (1992).

[7] G.H. Golub, C.F. Van Loan. "Matrix Computations", p. 243. Johns Hopkins University Press, Baltimore, MD (1989).

[8] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, W. In Proc. SIGGRAPH 1993, 19-26 (1993).

[9] M. Kass. CONDOR: Constraint-Based Dataflow. In Proc. SIGGRAPH 1992, 321-330 (1992).

[10] D. Kurlander, S. Feiner. "Inferring Constraints from Multiple Snapshots." ACM Transactions on Graphics 12(4):277-304 (1993). pp. 277-304.].

[11] S. Lee, G. Wolberg, S.Y. Shin. "Polymorph: Morphing Among Multiple Images". IEEE Computer Graphics and Applications 58 (Jan/Feb 1998).

[12] S.E. Librande. "Example-Based Character Drawing". Master's thesis, Media Arts and Science, MIT (1992).

[13] P.C. Litwinowicz. "Inkwell: a 2 1/2-D Amination System". In Proc. SIGGRAPH 1991, 113-122 (1991).

[14] MetaCreations Corp. Painter 6. Computer Program (1999).

[15] J.R. Munkres. "Elements of Algebraic Topology". Addison-Wesley, Reading, MA (1984).

[16] D.K. Pai. "Least Constraint: A Framework for the Control of Complex Mechanical Systems". In Proc. American Control Conf., Boston, MA, 1615--1621 (June 1991).

[17] P. Rademacher. View-Dependent Geometry. In Proc. SIGGRAPH 1999, 439-446 (1999).

[18] I.E. Sutherland. "Sketchpad: A man-machine graphical communication system." Ph.D. thesis, Dept. of Electrical Engineering, MIT (1963).

[11]We have shown this property for the case of linear movement in one factor.

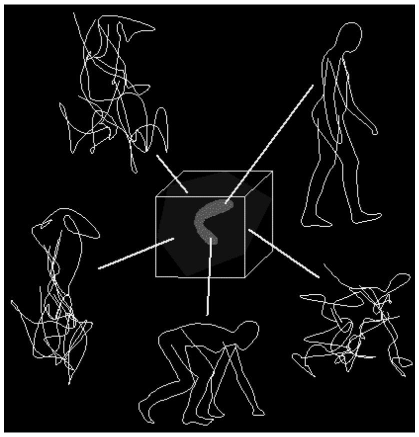


Figure 1. Configuration space

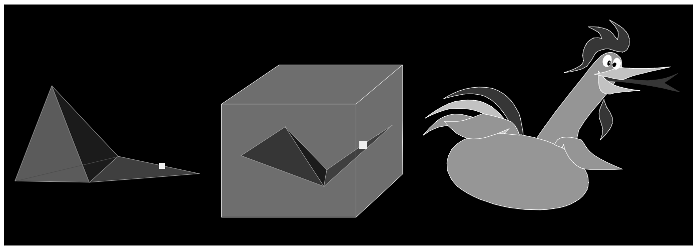


Figure 2. Modeling pipeline: State space → configuration space → image space

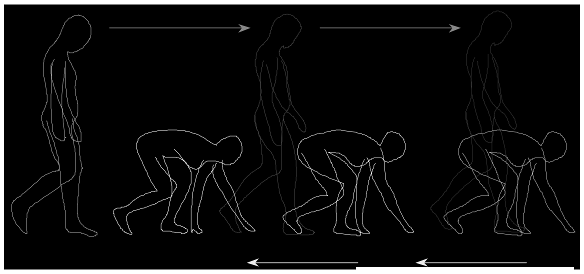
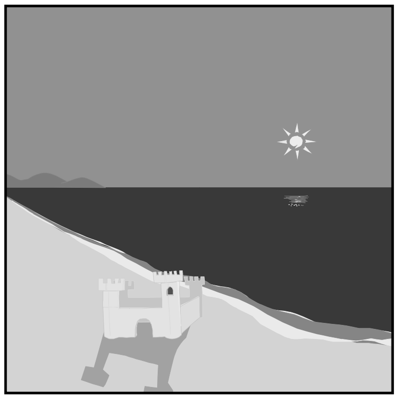
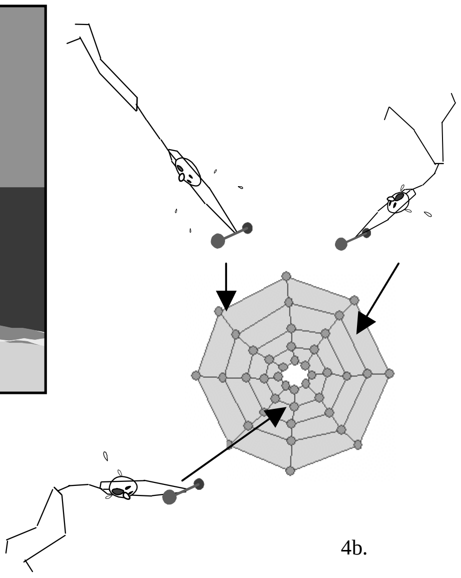


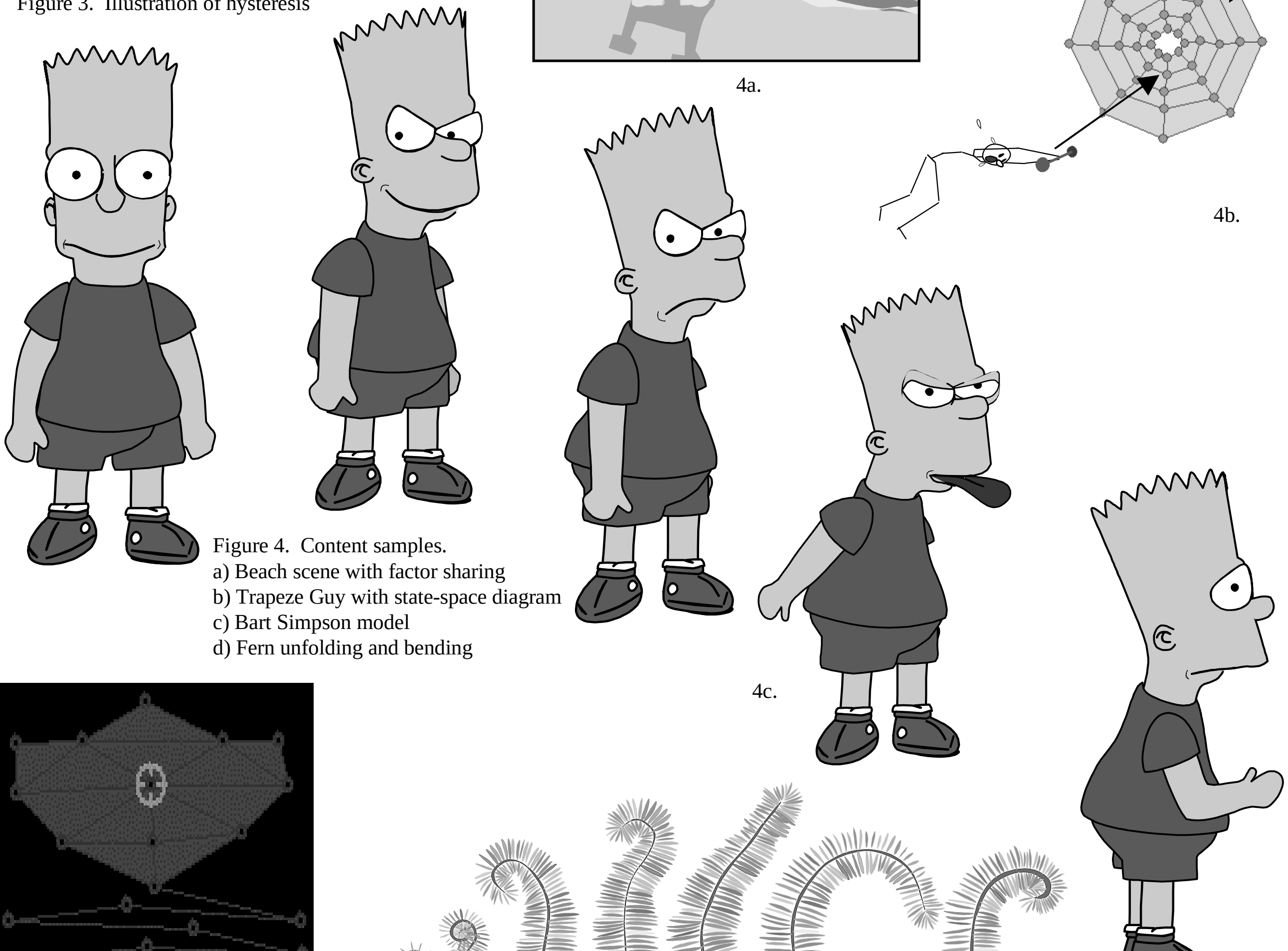
Figure 3. Illustration of hysteresis



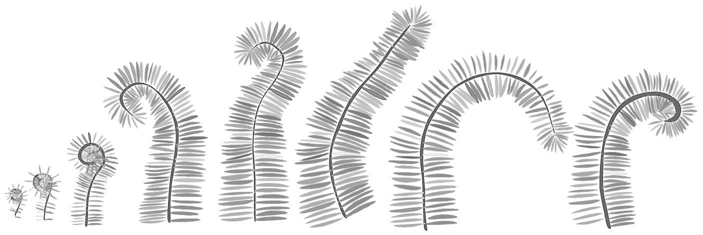
4a.



4b.



4c.



4d.

Figure 4. Content samples.
a) Beach scene with factor sharing
b) Trapeze Guy with state-space diagram
c) Bart Simpson model
d) Fern unfolding and bending

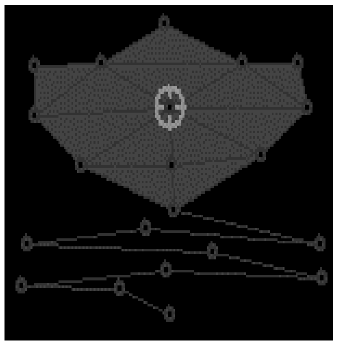


Figure 5. State-space diagram for fern