Simplicial Families of Drawings

Lucas Kovar
University of Wisconsin-Madison
1210 W. Dayton
Madison, WI 53706
+1-608-265-2711
kovar@cs.wisc.edu

Michael Gleicher
University of Wisconsin-Madison
1210 W. Dayton
Madison, WI 53706
+1-608-263-2874
gleicher@cs.wisc.edu

ABSTRACT

In this paper we present a method for helping artists make artwork more accessible to casual users. We focus on the specific case of drawings, showing how a small number of drawings can be transformed into a richer object containing an entire family of similar drawings. This object is represented as a simplicial complex approximating a set of valid interpolations in configuration space. The artist does not interact directly with the simplicial complex. Instead, she guides its construction by answering a specially chosen set of yes/no questions. By combining the flexibility of a simplicial complex with direct human guidance, we are able to represent very general constraints on membership in a family. The constructed simplicial complex supports a variety of algorithms useful to an end user, including random sampling of the space of drawings, constrained interpolation between drawings, projection of another drawing into the family, and interactive exploration of the family.

KEYWORDS: Animation with constraints, geometric modeling

INTRODUCTION

Creating an artwork can be an expensive and labor-intensive process. If one doesn't have the requisite time and skill, one must rely on either pre-created examples from art libraries or automated generation techniques. Each approach has its drawbacks. The size of a given art library is bounded, limiting the amount of available variation. Generative approaches may not create the quality or specific properties of crafted examples. Either way, the user is likely to need to make adjustments to the drawing. This is problematic since, if the original quality is to be maintained, modifying an artwork can be nearly as challenging as creating it in the first place.

While ever larger libraries and more facile drawing editors may help, an alternate approach is to combine libraries with automated generation. The library is extended to store families of drawings related to the crafted examples, and generative methods are used to instantiate specific members of these families. Such an approach is attractive because it provides users with a collection of existing works of acceptable quality while simultaneously giving them control over the specific form of the final result. The central challenge of this approach is the representation of the families of drawings. The representation must afford the quality of drawings that the users demand, be rich enough to express complex families of similar drawings, support the generation of members of the family, and not be too difficult for the library authors to create.

Ngo et al [12] recently presented a topologically-flexible geometric construction called a *simplicial complex* and demonstrated its use in an interactive editing operation they termed "tugging". In their system a user supplied a set of seed drawings and then specified overlapping subsets (the *simplices*). A multi-target blending operation could be applied to the members of a subset in order to produce new drawings, forming the subset's *interpolation space*. The final simplicial complex was the union of the drawings produced in this manner by all of the subsets. Multiple simplicial complexes could be used in the event that a drawing had distinct parts. Once this structure was created, a user could interactively explore the family of drawings by dragging on an existing drawing. For example, if one had a drawing of a smiling face and wanted to turn it into a frown, one could drag on a single point on the mouth and have the entire face update appropriately.

This technique is promising for our goals, but it suffers a key limitation in that the structure of the simplicial complex must be specified by hand. This requires an artist to design a collection of drawings such that every interpolation is acceptable. This task is likely to be unintuitive, and it may even be impossible in some circumstances. On the other hand, though it is difficult to anticipate whether the interpolation algorithm will always succeed, it is comparatively simple for a human to identify whether a particular drawing is acceptable. We therefore employ user feedback to classify samples from the interpolation space of a set of example drawings. These classified examples are used to construct a simplicial complex representing the valid regions of the space, which is

in turn the desired family of drawings. We will also show that this simplicial complex supports useful algorithms outside of tugging:

Sampling: Randomly generate new legal drawings. For example, if one wanted to fill a field with flowers but only had a few actual drawings, one could build the simplicial complex and use it to populate the field with distinct flowers.

Morphing: Given two legal drawings, generate a sequence of drawings that smoothly transform the first into the second such that each of the inbetween drawings is legal.

Projection: Find the legal drawing "closest" to another drawing. For instance, if we had constructed a space of legal drawings for a cartoon character, we might then create that character in a particular pose by drawing a stick figure and projecting it into the legal space.

The remainder of the paper is divided into five sections. In Section 2, we discuss related work. In Section 3, we explain why a simplicial complex is an appropriate representation for a set of legal drawings. In Section 4, we present a method for constructing a simplicial complex based upon a user's classification of artificially-generated drawings. In Section 5, we describe how each of the preceding algorithms can be implemented in a simplicial complex and show sample results. In Section 6, we discuss potential areas for future work.

RELATED WORK

Ever since there have been interactive drawing systems, designers have grappled with the problem of creating related families of drawings. Ivan Sutherland's Sketchpad system [19] introduced the concept of an interactive drawing program, and it also introduced constraints as a mechanism for representing what changes were permitted to a drawing. These geometric constraints were considered part of the drawing itself. Since Sketchpad, other graphics systems have continued the concept of persistent constraints as a mechanism for defining the related family of drawings. Despite this, constraint-based drawing has been unpopular as constraints have proven to be difficult to specify, debug, and solve. A survey of the issues in constraint-based drawing was presented by Gleicher and Witkin in [4].

For representating families of drawings in libraries, constraints fail on several points. While Kurlander and Feiner [8, 7] explored the use of determining constraints based on multiple example drawings, in general it is difficult (or impossible) to find a complete set of restrictions sufficient for defining a valid family of drawings. While constraint inference mechanisms such as in Chimera [7] or Briar [4] make it easier to create constraints during the drawing process, augmentation of an existing artwork is difficult to execute, test, and debug. Finally, once a constraint-based representation is constructed, not all of the previously-outlined generation operations are realizable. For example, random sampling is difficult with most geometric constraint systems.

Generative modeling offers a different method for constructing families of drawings. This approach defines a family of drawings by a set of rules (a grammar) or a procedure that generates members of the family. While specific generative models have been created for various categories of artwork, such as flowers [13, 14], victorian houses [5], and floral ornamentation [21], defining a family this way is challenging enough that the models often are considered research results in their own right. Moreover, most generative models also do not support all of the desired generation operations.

Our strategy of building a representation of a set based on a series of sample classifications is well studied in the artificial intellegence community. Unfortunately, the established techniques (such as neural nets and decision trees) are designed for classification rather than generation. Hence they do not support the tasks we demand.

In the graphics community, systems guided by user classification of randomly generated examples have been discussed for procedurally defined images by Sims [18], and for a wide range of design tasks by Marks et al [11].

Our representation falls into a class of methods that represent a space of graphics objects by combining members of an example set. Most commonly radial basis functions are used to perform scattered data interpolation. This has, for example, been applied to character animation and skinning [15, 3, 9]. The work of Librande and Poggio [10] (the basis for a commercial software package produced by NFX) is the most similar to ours. They used radial basis functions to represent a space of drawings similar to an example set and supported sampling, morphing, and projection operations. However, there was no notion of systematically culling portions of the space deemed unacceptable by the user.

As noted in the introduction, our work is inspired by that of Ngo et al [12] and may be viewed as the next step toward using a simplicial complex to effectively model a set of desirable artwork. We have extended their work by providing a semi-automated method to faciliate construction of the simplicial complex and by demonstrating that simplicial complexes support operations other than tugging.

REPRESENTING A SET OF LEGAL DRAWINGS WITH A SIMPLICIAL COMPLEX

Our goal is to construct a representation of the set of "legal" drawings similar to the initial set. This requires defining a space of candidate drawings and deciding upon a representation for the legal subset of this space. In the following subsections we present *interpolation space* as a mechanism for concisely defining a space of drawings similar to the examples. We then argue that simplicial complexes are an appro-

priate representation for the legal subset of this space.

Interpolation space

A typical vector drawing may be parameterized by a set of numbers that includes the position of each point, the thickness of each curve, and the color of each region. These numbers are treated as a point in a high-dimensional space called the *image space*. We can now attempt to find other points in image space that are also legal drawings. This requires addressing two problems. First, in general two drawings will not have the same number of parameters and the same meaning ascribed to each parameter. To overcome this we assume that the initial set of drawings has been preprocessed such that each drawing has the same topology, i.e., the same number of curves and same number of points on each curve. We also assume that point correspondences are known. While this requires extra effort on the part of the artist, methods exist that help automate these processes [16]. For our examples we assumed that each drawing had the same number of curves and built an interface allowing the user to specify sparse point correspondences. We then evenly added points to the curves such that every set of corresponding segments contained the same number of points.

The second problem is that the image space is too large to work with in its entirety. Moreover, only a small region is likely to contain interesting drawings. It is hence necessary to identify a tractable subset. As a first step, we may describe the configuration in some more convenient coordinate system. For instance, we might encode rotations and scale parameters of rigid objects, rather than the positions of each point. We define *configuration space* (also called the *parameter space*) to be the space of these parameters. The mapping $\mathcal{F}$ from configuration space to image space was termed the "rendering map" by [12].

We can now define a third space existing only "in-between" the example drawings. Given a set of n drawings, $\mathbf{d_1}, \ldots, \mathbf{d_n}$, we define the *interpolation space* as the set of all convex linear combinations of these drawings. That is, a drawing (vector) $\mathbf{p}$ is in the interpolation space of the set of drawings $\mathbf{d_1}, \ldots, \mathbf{d_n}$ if and only if

$$\mathbf{p} = \sum_{i=1}^{n} \alpha_i \mathbf{d_i}$$

for convex weights $(\alpha_1, \ldots, \alpha_n)$ (that is, each α_i is non-negative and $\sum_i \alpha_i = 1$). Notice that although interpolation space is embedded in the high-dimensional configuration space, it is itself only of dimension $n-1$. Consequently a more compact representation of this space is possible by using only a vector of the α_i's.

Through the use of properly chosen rendering maps, a variety of methods for combining drawings can be realized using this framework. For instance, one of the examples in this paper transforms the drawings into a configuration space of angles and edge lengths, realizing the shape interpolation method described by Sederberg and Greenwood [17]. We believe that other interpolation schemes may also be applicable to our method, such as as-rigid-as-possible interpolation [1].

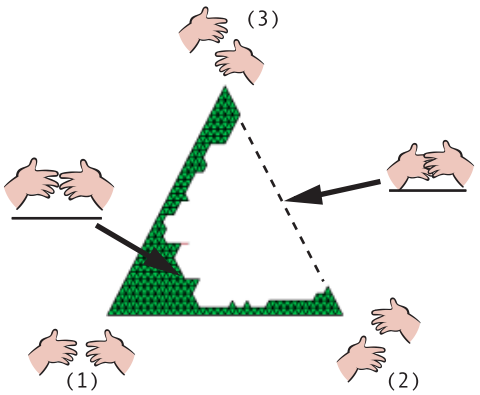

Figure 1: Three drawings of two hands (labelled 1 through 3) are used to construct a simplicial complex representation of the family of valid drawings. Here a "valid drawing" is defined as one wherein the hands don't intersect. The dotted line demarcates the boundary of the initial interpolation space. The underlined drawings are interpolations; the arrows indicate their locations in interpolation space.

Interpolation space is a $(n-1)$-dimension simplex in parameter space whose vertices correspond to the original drawings. A simplex in $k-1$ dimensions is defined by the convex hull of k points, where the vectors defining those points are linearly independent. In 1 dimension, a simplex is a line; in 2D it is a triangle and in 3D a tetrahedron. Any subset of the k points of a simplex forms another simplex, called a face of the original. A vector lies within a simplex if and only if it is a convex combination of the vertices.

In many cases the interpolation space will contain undesirable elements. Hence we identify the portions of interpolation space containing desirable ("legal") drawings. This amounts to sampling the simplex at specific points (by asking the user to classify drawings) and carving away regions containing illegal elements. Note that the definition of legality is entirely up to the user - she may reject drawings based upon unsatisfactory coloring, inadvertent curve intersections, or shapes that simply don't "look right" (see Figure 1).

For sampling to be feasible, the interpolation space must be of a reasonably small dimension, say, at most five or six. Since the initial drawings are presumed to be difficult to obtain, we simply assume that there will be few enough of them to make sampling practical. In the event that there are more drawings but only subsets of them can be meaningfully interpolated, we can treat each of these subsets independently and merge the results.

Representing the Legal Subset With a Simplicial Complex

A variety of methods may be used to reconstruct the legal space from the sampling. In general the samples will not be regularly spaced, and hence the problem is one of scattered data interpolation. Radial basis functions have been a popular method [15, 9, 10], but they are limited in the topologies they can represent (e.g., they can not contain holes). To avoid placing artificial restrictions on how the user can define the legal space, we use a more topologically general data structure: the simplicial complex. A simplicial complex is a set of simplices such that any simplices which intersect do so by sharing a face. The component simplices need not be of the same dimension and the simplicial complex may not be connected. The legal region in Figure 1 is represented as a simplicial complex.

In addition to the geometric interpretation, a simplicial complex may also be viewed as an embedded graph. In this case each vertex of a simplex is a node and each 1D face is an edge. Similarly, a simplicial complex can be treated as a hypergraph [2], which is similar to a graph except "edges" may involve more than two points. These edges correspond to faces at which simplices intersect.

Simplicial complexes facilitate many useful calculations. Due to the simple geometry of each simplex, it is easy to test whether a given point is inside the complex. Travel between adjacent simplices is straightforward since the connection is simple. The graph and hypergraph representations allow us to draw on algorithms developed for those data structures.

CONSTRUCTING THE SIMPLICIAL COMPLEX

We will now describe how to construct a simplicial complex representing the set of legal interpolations. This construction involves two phases: sampling and reconstruction. In the sampling phase we pick points from the interpolation space (henceforward called sample points) for the user to classify. Since the user may cease answering questions at any time, point selection must be guided by heuristics that attempt to maximize information content. During reconstruction these samples are interpolated to approximate the legal region of interpolation space. We emphasize that the only task required of the user is answering a series of yes/no questions; the underlying simplicial structure need never be presented.

Sample Point Selection

We use two heuristics to decide which drawings to present to the user. First, we would like to select sample points from the simplex in a hierarchical fashion, learning about the coarse structure of the space before filling in finer details. This is done by progressively dividing the simplex into smaller simplices and then selecting sample points from these subsimplices. The following algorithm implements the subdivision (Figure 2). First, sample points are placed at the midpoint of each edge in the initial n_d-dimension simplex (i.e., each line segment connecting two vertices). Edges are added between these sample points such that the simplex is subdivided into a set of non-overlapping sub-simplices of dimension n_d. In 2D, each new edge naturally partitions the relevant simplex; in higher dimensions, the sub-simplices are defined as maximal cliques of the resulting graph. Once all of the sample points have been classified, the process repeats: new sample points are placed at the midpoint of each edge of each simplex and then the simplices are subdivided.

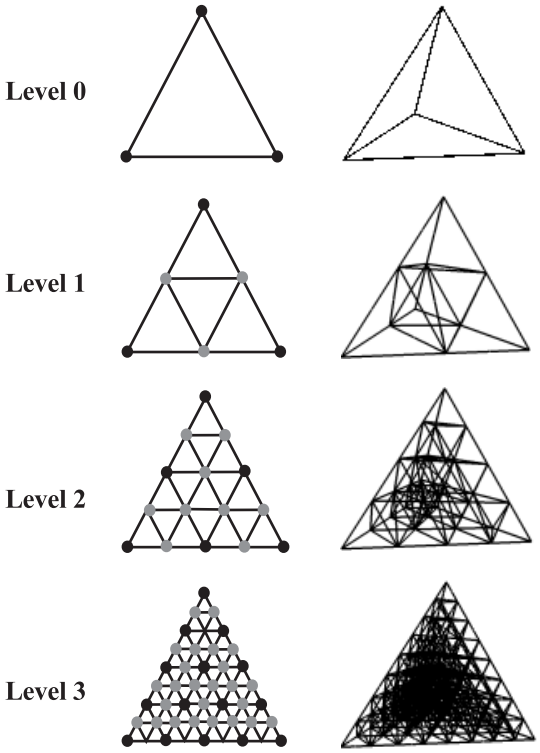

Figure 2: The left column illustrates the subdivision scheme for a 2D simplex. Classified and unclassified sample points are represented as, respectively, dark and light dots. The right column shows the subdivision scheme for a 3D simplex with sample points removed for clarity.

In two dimensions, there is only one way to attach the midpoints to perform the subdivision, but in higher dimensions care must be taken so the newly-added edges don't create overlapping sub-simplices. The appendix contains pseudocode for an algorithm that determines how to add edges to subdivide a simplex of arbitrary dimension. Note, however, that once a subdivision scheme is generated it can be saved for future use.

Deeper levels of resolution may contain many points, and therefore we should not expect the user to classify all of them. This requires us to prioritize the order in which the samples are presented to the user. One reasonable heuristic is to bias the sampling toward learning the borders of the space. This can be achieved by preferring sample points near an equal mix of legal and illegal neighbors to those near, say, only legal neighbors.

The simplest algorithm would count the number of legal and

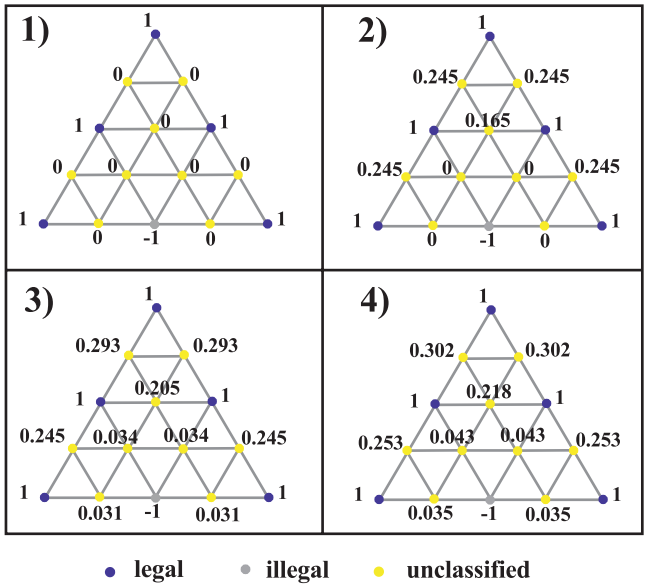

Figure 3: The diffusion process iteratively estimates the uncertainty at each sample point at the current subdivision level, allowing them to be prioritized for their presentation to the user. Above are uncertainty estimates for the first four iterations on an example configuration.

illegal neighbors of each unclassified point, but this doesn't address the fact that edges may connect to other unclassified points. For this reason we use an iterative algorithm for assigning interest values to each unclassified point. First, all legal points are set to 1, all illegal points are set to -1, and all unclassified points are set to 0. Then every unclassified point $\mathbf{p}$ with neighbors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is assigned the value

$$v(\mathbf{p}) = tanh(\frac{1}{n}\sum_{k=1}^{n} v(\mathbf{x}_k))) \quad (1)$$

where $v(\mathbf{x}_k)$ is the value of $\mathbf{x}_k$ at the beginning of the current iteration. Note that Equation 1 has the largest absolute value when $\mathbf{p}$ has either all legal or all illegal neighbors, and it equals 0 if $\mathbf{p}$ has an equal number of legal and illegal neighbors. After iterating a small number of times, the sample point with the smallest absolute value is selected. Ties are broken randomly. Figure 3 shows the results of applying this algorithm to a particular sample point configuration.

Final Construction

Once the user has indicated that she no longer wishes to classify drawings, we must interpolate the sample points to form the final simplicial complex. First the value-assigning algorithm from above is run on any remaining unclassified sample points. Those points are deemed legal or illegal according to the sign of the calculated value. Then any edge containing an illegal vertex is removed and the maximal cliques of the resulting graph become the simplices of the final simplicial complex.

The process of identifying the maximal cliques is greatly simplified by observing that the points of any maximal clique must be fully contained in one of the simplices generated at the deepest level of subdivision, which we refer to as leaf simplices. We first locate all leaf simplices with n_d positive points. Each such set of n_d points is a simplex in the final simplicial complex. We then search the remaining leaf simplices for those with $n_{d-1}, n_{d-2}, \ldots, 1$ positive points, each time making sure that we do not attempt to add a simplex that is subsumed by one we have added previously (this ensures we only extract the maximal cliques).

One can imagine other algorithms for constructing the final simplicial complex. For example, one could partition the interpolation space into cells via a generalized Voronoi diagram, remove cells whose interiors contain negative sample points, and triangulate the remaining cells to ensure the resulting structure is a simplicial complex. However, this would not change the qualitative structure of the final simplicial complex, and as the reconstruction is imprecise to begin with, we feel our simpler scheme is more appropriate.

USING THE SIMPLICIAL COMPLEX

In this section we describe how each of the algorithms mentioned in the introduction may be implemented using a simplicial complex of legal drawings. Figure 4 shows the three examples used in this paper and their corresponding simplicial complexes.

Sampling

Random legal drawings can be drawn from the simplicial complex by choosing a random simplex and then selecting a random point therein. The latter is implemented as follows: if the parameter vectors corresponding to the vertices of the chosen simplex are $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k$, then we generate a convex set of random numbers $\lambda_1, \lambda_2, \ldots, \lambda_k$ and calculate $\sum_i (\lambda_i \mathbf{p}_i)$.

If we want uniform sampling, then selecting the initial simplex is a subtler problem as we must decide how to weight the simplices. One obvious solution is to make a simplex's weight proportional to its volume. Various formulas exist for this calculation, such as the Gram determinant and the Cayley-Menger determinant[6]. However, in general the simplicial complex will contain simplices of differing dimensionality, making direct volume comparisons meaningless. We chose to handle this problem by allowing a user-defined "sampling density" ρ and multiplying the volume of every k-vertex simplex by ρ^{k-1}. Regardless, in our examples we found that doing "correct" sampling didn't produce appreciably different results.

Figure 6 shows some results for sampling. We constructed a simplicial complex for the islands and then sampled both by randomly assigning weights throughout interpolation space and by confining sampling to the simplicial complex. The scale of the drawings obscures some of the subtler self-

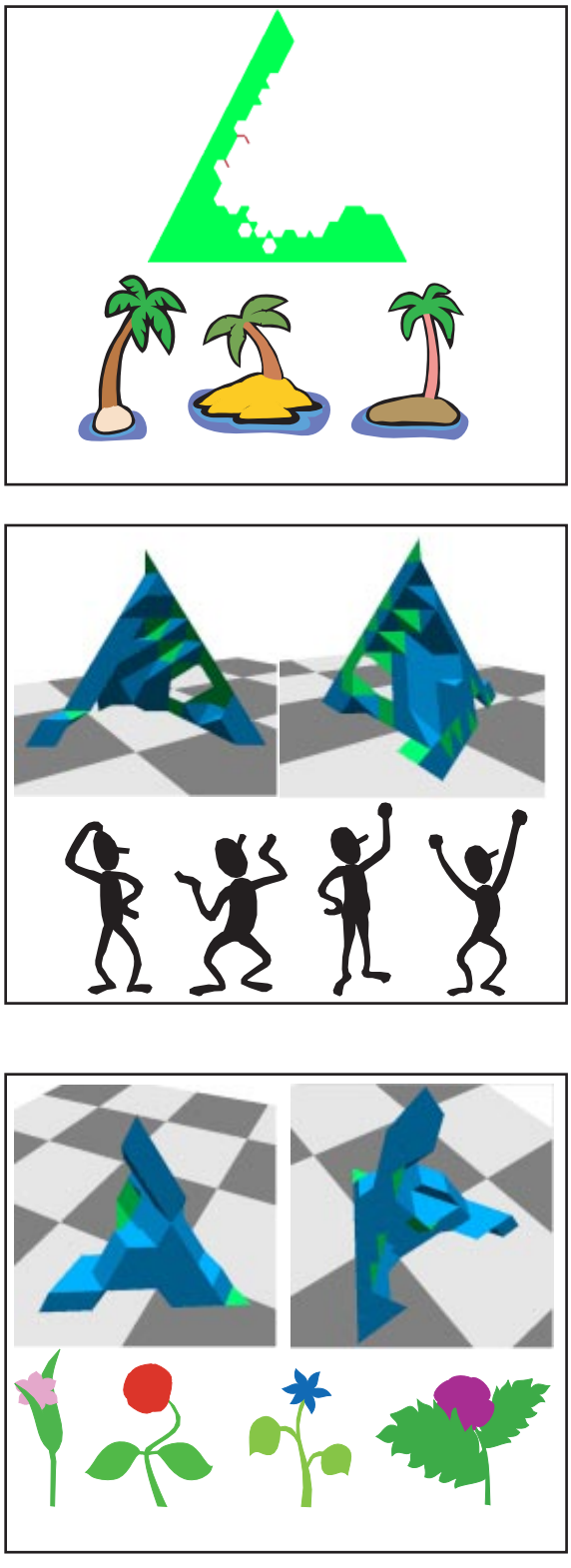

Figure 4: The three examples used in this paper. The top shows 3 islands, the middle shows 4 cartoon figures, and the bottom shows 4 flowers. In the latter two cases we present two views of the 3D simplicial complex of valid interpolations. 3D simplices (tetrahedra) are in blue, 2D simplices (triangles) are in green, and 1D simplices are in red. All drawings are modifications of drawings found on a web-based clip-art library.

intersection problems, but at least a few of the drawings in the unconstrained sampling are clearly incorrect.

An obvious use for sampling is crowd generation (such as for a wallpaper scheme), as it provides a reliable way of quickly generating distinct drawings. If one wanted new drawings "like" a particular drawing, then one could preferentially weight the sampling to be near the corresponding point in the simplicial complex.

Morphing

A legal morph can be generated between a source drawing and destination drawing by finding a path in the simplicial complex that connects the two corresponding points. First, if either the source or the destination is not one of the vertices of a simplex, we create a path connecting it to such a vertex. We can then treat the simplicial complex as a graph and use Dijkstra's algorithm to find a lowest-cost path connecting the two verties. A given edge is weighted by the L^2 distance between the endpoints in *image space*, not parameter space. That is, we define the distance between drawings as the sum of the squared distance between corresponding points. The advantage of this is that distance is meaningfully defined regardless of how drawings are parameterized.

The path produced by this algorithm is piecewise linear, and so the resulting animation may appear jerky. This can be resolved by fitting a spline to the path's vertices. While the spline may extend beyond the boundaries of the legal space, it ought not be noticeable if these deviations are small. Still, if such deviations are a problem then spline tension may be used to constrain the path tighter to the simplicial complex.

This algorithm is not optimal in the sense that it doesn't find the shortest possible path, which requires traversing the interior of individual simplices. While in practice the shortest path is unlikely to be appreciably different from the path our method produces, it is of intellectual interest to consider how to calculate it. One approach is to use an active contour algorithm of the sort found in the computer vision community [20]. The starting path would be obtained as above and the energy terms would consist of the total arc length of the path plus a penalty term for extending past the boundaries of the simplicial complex.

This constrained morphing algorithm is particularly useful in two cases. First, it can create reasonable morphs in cases where direct interpolations fail, as in Fig. 6. Second, if one has to make a large number of morphs within a particular set of drawings (say, if one is animating a cartoon character), then it is convenient to be able to specify two poses and automatically generate a correct morph.

Projection

A drawing $\mathbf{K}$ can be projected into the legal space in the sense that we can find the drawing that is geometrically the most similar. This can be done by finding the point in the simplicial complex that is closest to $\mathbf{K}$ in the L^2 sense. As

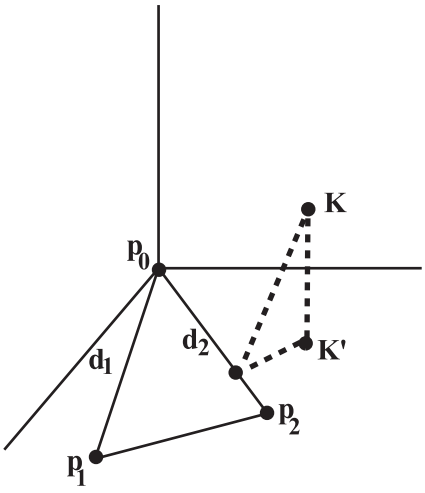

Figure 5: The point $\mathbf{K}$ is projected into the simplex $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$ at the point $\mathbf{K}'$

with morphing, we measure distance in image space.

For the moment we will assume that the function $\mathcal{F}$ mapping the image space to the parameter space is linear. If this is true, then every simplex in parameter space has a corresponding simplex in image space. We will first consider the case where we want to project $\mathbf{K}$ into a single simplex S with image space vertices $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_r$. If $r = 0$, then S is a single point and the solution is trivial. Otherwise, consider the barycentric coordinates of S with origin $\mathbf{p}_0$ and spanning vectors $\mathbf{d}_1$, ..., $\mathbf{d}_r$, where $\mathbf{d}_i = \mathbf{p}_i - \mathbf{p}_0$ (see Figure 5). A point is inside S iff it can be expressed as $\sum_{i=1}^{r}(\alpha_i \mathbf{d}_i)$ such that the α_i's are convex. Now, the point in S closest to $\mathbf{K}$ will also be closest to the projection $\mathbf{K}'$ of $\mathbf{K}$ into the linear subspace spanned by the $\mathbf{d_i}$'s. $\mathbf{K}'$ can be expressed as a weighted combination of $\mathbf{d}_1, ..., \mathbf{d}_r$ by finding the least-squares solution of the following linear system

$$\mathbf{D}\alpha = \mathbf{K} - \mathbf{p}_0, \tag{2}$$

where the i^{th} column of $\mathbf{D}$ is $\mathbf{d_i}$. A least-squares solution to equation 2 can be found by solving the normal equations,

$$\mathbf{D}^T\mathbf{D}\alpha = \mathbf{D}^T(\mathbf{K} - \mathbf{p}_0) \tag{3}$$

Note that the resulting system is quite small: at most $n_d - 1$ by $n_d - 1$ for n_d drawings. In our implementation we solved this system via the singular value decomposition, which was chosen for its stability.

The matrix multiplies in Equation 3 are potentially expensive if the drawings contain a large number of parameters. However, we can greatly reduce the necessary computation by recognizing that every column of $\mathbf{D}$ is simply a linear combination of the original drawings $\mathbf{P}_1, \ldots, \mathbf{P}_{n_d}$. Hence we have

$$d_i \cdot d_j = \sum_{m=1}^{n_d}\sum_{n=1}^{n_d} \beta_m \gamma_n (\mathbf{P}_m \cdot \mathbf{P}_n), \tag{4}$$

where $\sum(\beta_m \mathbf{P}_m) = \mathbf{d}_i$ and $\sum(\gamma_n \mathbf{P}_n) = \mathbf{d}_j$. If we precompute and store the dot product of every pair of drawings, then only n_d^2 calculations are necessary for each entry of $\mathbf{D}^T\mathbf{D}$.

Once we have found α, there are three possible scenarios. First, say that one of the elements α_i is negative. This means that any attempt to travel along $\mathbf{d}_i$ toward $\mathbf{K}'$ will take us outside of the simplex. Hence the solution we seek is in the hyperface that excludes $\mathbf{p}_i$. So, if there are negative elements in α, we remove the corresponding vertices and repeat the process on the smaller simplex. The second scenario is where the elements of α are all positive but sum to a value greater than 1. This means the solution must lie on the hyperface that excludes $\mathbf{p}_0$, and so we remove $\mathbf{p}_0$ from the simplex and iterate. The third and final scenario is when the elements of α are positive and sum to a value less than or equal to 1. Then $\mathbf{K}'$ is inside the simplex and is the closest point to $\mathbf{K}$.

The projection of $\mathbf{K}$ into the simplicial complex can now be found by projecting it into each simplex and selecting the projection that minimizes the L^2 error. Since $\mathbf{K}$ will be projected multiple times, we use the same trick as in Equation 4 and precompute $\mathbf{P}_i \cdot \mathbf{K}$. If speed is still a problem, we can project into fewer simplices by using a branch-and-bound algorithm. Recall that the original interpolation space was partitioned into a tree-like hierarchy of volumes while we were constructing the simplicial complex. Starting at the root node (original simplex), we project into each child simplex, calculate the distance to the projection, and store the simplex-value pairs in a priority queue. We then remove the smallest-valued element of the queue and check whether it's a leaf node. If it is, we are done. If not, we repeat the process. Since the projection into a leaf simplex will never be closer than the projection into its parent, we are guaranteed to find the closest projection.

If $\mathcal{F}$ is not linear, in general a simplex in parameter space will not correspond with a simplex in image space. In this case projection is a nonlinear optimization problem with no easy general solution. However, if a simplex in parameter space is small, then its transformation into image space may be approximated as a simplex. Thus, we can implement projection in the nonlinear case by 1) subdividing each simplex until it is sufficiently small, 2) transforming the vertices of each simplex S into image space and treating them as corners of a simplex S', and 3) projecting $\mathbf{K}$ into S' as outlined above. This will take longer than in the linear case, since in general neither Equation 4 nor branch-and-bound are applicable.

Figure 6 shows an example of projection. Projection is useful as a way of automatically filling in details given a general shape. A potential application is conversion of a set of drawings (such as of a cartoon character) into another set of similar topology but different detail (such as a different character). In our current implementation, it is still necessary to generate point correspondences between one of the origi-

nal drawings and the drawing to be projected. However, if a group of drawings are all registered to one another, then only one would have to be registered with the original set.

Tugging

Ngo et al [12] have shown in earlier work how to implement tugging with a simplicial complex; in fact, they handle the more general case where the interpolation space is the Cartesian product of multiple simplicial complices. We will briefly summarize their algorithm for the case of a single simplicial complex. Say the current configuration of the drawing places it in the simplex $\mathcal{S}$ and the user attempts to drag a particular point on the drawing. This is equivalent to attempting to change the original image-space vector $\mathbf{x}$ to a new vector $\mathbf{x}'$. The desired direction of change is $\mathbf{C}_x = \mathbf{x}' - \mathbf{x}$. This can be converted to a direction in parameter space via the equation

$$\mathbf{J}\mathbf{C}_\alpha = \mathbf{C}_x. \tag{5}$$

Here $\mathbf{C}_\alpha$ is the projection of $\mathbf{C}_x$ into the linear space defined by the barycentric coordinates of $\mathcal{S}$ and $\mathbf{J}$ is the Jacobian of $\mathbf{x}$ with respect to α. This system can be solved safely via the singular value decomposition. By starting at the original location in $\mathcal{S}$ and travelling along $\mathbf{C}_\alpha$, we match the user's attempted modification as much as possible. If we encounter a boundary (hyperface) of $\mathcal{S}$ before fully traversing $\mathbf{C}_\alpha$, then there are two cases. If the boundary leads to adjoining simplices, then we continue travelling in the simplex that has the closest projection to $\mathbf{C}_x$. If there is no adjoining simplex, then we recalculate Equation 5 replacing the original simplex with the hyperface.

Figure 6 shows snapshots of the tugging process.

CONCLUSION AND FUTURE WORK

In this paper we have presented a technique for converting a set of drawings into a more useful form. This was accomplished by sampling the set of interpolations in order to reconstruct the subset that produces acceptable results. This subset was in turn represented as a simplicial complex, which facilitated implementation of algorithms for manipulating and generating drawings similar to the initial set.

The key property of our work is that the representation for related families of drawings is both easy to construct and capable of supporting generative tasks. The authoring of the simplicial complex from a set of registered drawings requires a user only to answer a series of yes/no questions evaluating generated drawings. Users need answer only as many questions as they have patience for, although answering more questions yields a more accurate representation.

In practice, the difficult step is obtaining registered sets of related drawings. Without the development of automatic tools for determining correspondences, it requires non-trivial effort to prepare "found" drawings for usage in our system, despite the fact that it reads a standard file format (postscript files as generated by Adobe Illustrator). Given the challenge of such automation, we expect that our tool would be most useful in cases where the drawings are specifically authored to work with our system.

We believe that there are no conceptual difficulties in applying our approach to other data, such as 3D models. The only place where our system makes any interpretation of the vectors that represent drawings is in the user interface. Extending our system to more sophisticated interpolation schemes, such as [1], mainly requires devising multi-target versions of them.

The limitation to a small number of drawings is inherent in our approach: a higher dimensional space is too large to sample effectively. In practice, this limitation is not very restrictive as we rarely encounter large sets of sufficiently similar drawings. In the event that we do, we expect that the best approach will be to manually divide the examples into smaller subsets and construct simplicial complexes for each subset independently. The union of these simplicial complexes would then form the simplicial complex for the original ensemble. Such an approach even more closely resembles that of Ngo et al [12]. Indeed, one could imagine applying our subdivision process to add detail to their manually constructed simplices.

ACKNOWLEDGMENTS

This research is supported by an NSF Career Award "Motion Transformations for Computer Animation" CCR-9984506, support from Microsoft, equipment donations from IBM and Intel, and software donations from Alias/Wavefront, Microsoft, Intel, SoftImage, Autodesk, and Pixar. We also thank the reviewers for their many useful and constructive suggestions

REFERENCES

1. Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. *Computer Graphics (Proceedings of SIGGRAPH 00)*, pages 157–164, July 2000.

2. Claude Berge. *Graphs and Hypergraphs*. North-Holland Publishing, Amsterdam, 1973. translated by E. Minieka.

3. Michael Cohen. Example-based everything. In Hyeong-Seok Ko and Normal Badler, editors, *Proceedings of the Symposium on Human Modeling and Animation*, June 2000.

4. Michael Gleicher and Andrew Witkin. Drawing with constraints. *The Visual Computer*, 11(1), November 1994.

5. Jeff Heisserman and Robert Woodbury. Generating languages of solid models. *SMA '93: Proceedings of the Second Symposium on Solid Modeling and Applications*, pages 103–112, May 1993.

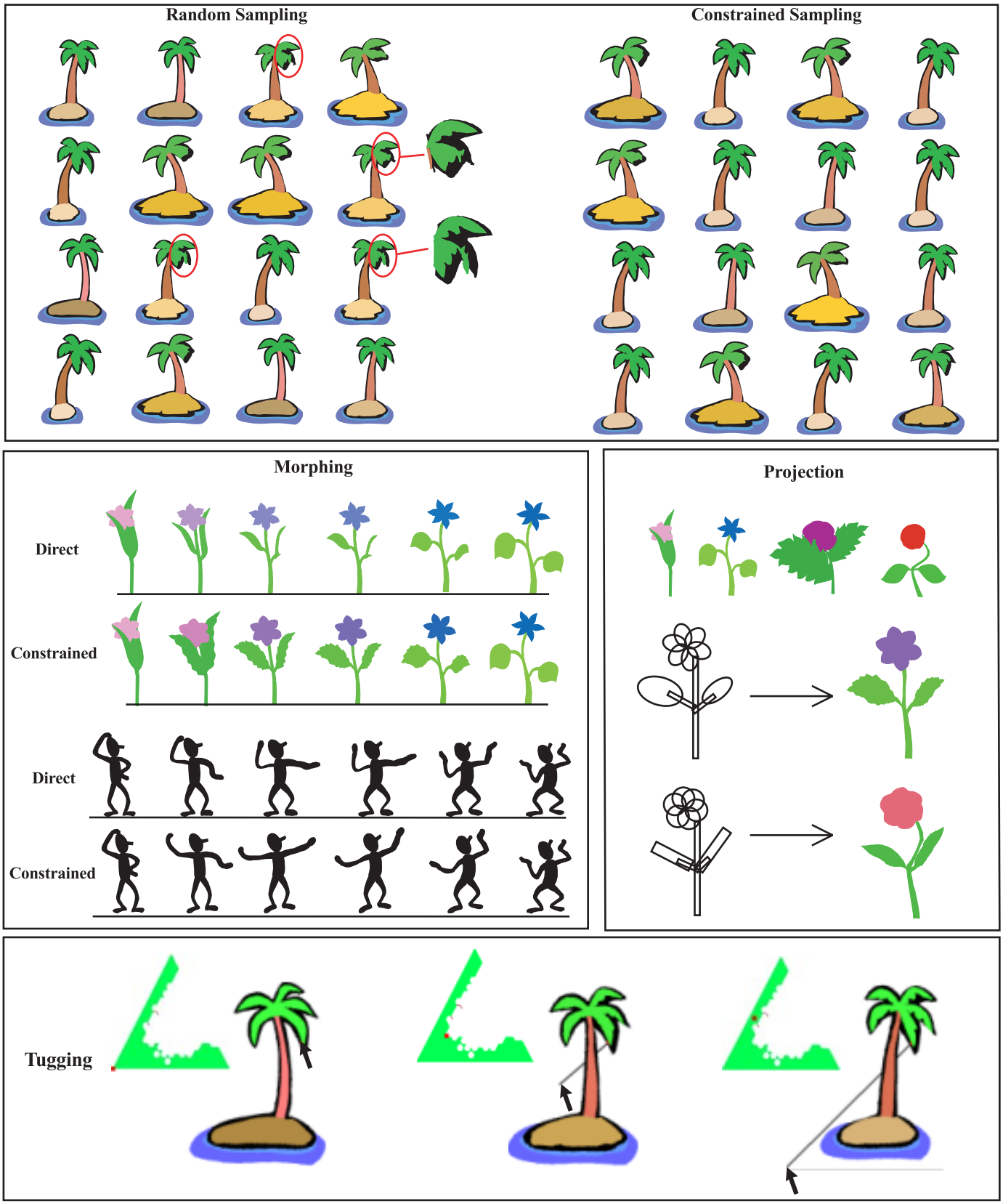

Figure 6: Examples of sampling, morphing, projection, and tugging. The top figure compares random sampling of interpolation space with sampling constrained to the simplicial complex. Note that unconstrained sampling produces undesirable drawings wherein the bottom-right branch of the palm tree ends in a self-intersecting scribble. The middle left figure compares direct linear interpolation with a constrained morph. The middle right figure shows two sketched flowers and their projections into the simplicial complex. Note that the projections are distinct from the original examples (shown on top). The bottom figure shows three snapshots of a typical tugging operation: a user clicks on the bottom right branch of the tree and drags diagonally down and to the left. The corresponding points in the simplicial complex are shown.

6. Christopher Hillman. Regarding: finding volume of tetrahedron with one vertex at origin. Posting on sci.math, January 1997.

7. David Kurlander. *Graphical Editing by Example*. PhD thesis, Columbia University, 1993.

8. David Kurlander and Steven Feiner. Inferring constraints from multiple snapshots. *ACM Transactions on Computer Graphics*, 12(4), October 1993.

9. J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformations: A unified approach to shape interpolation a nd skeleton-driven deformation. *Proceedings of SIGGRAPH 2000*, pages 165–172, July 2000.

10. Steve E. Librande. Example-based character drawing. Master's thesis, MIT, 1992.

11. J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. *Proceedings of SIGGRAPH 97*, pages 389–400, August 1997.

12. Tom Ngo, Doug Cutrell, Jenny Dana, Bruce Donald, Lorie Loeb, and Shunhui Zhu. Accessible animation and customizable graphics via simplicial configuration modeling. *Computer Graphics (Proceedings of SIGGRAPH 00)*, pages 403–410, July 2000.

13. P. Prusinkiewicz and A. Lindemayer. *The Algorithmic Beauty of Plants*. Springer Verlag, New York, 1st ed., 1990; 2nd ed 1996.

14. Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental models of herbaceous plants for computer imagery purposes. *Computer Graphics (Proceedings of SIGGRAPH 88)*, 22(4):141–150, August 1988.

15. Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications*, 18(5), September – October 1998.

16. Thomas W. Sederberg, P. Gao, G. Wang, and H. Mu. 2-d shape blending: An intrinsic solution to the vertex-path problem. *Computer Graphics (Proceedings of SIGGRAPH 93)*, 27(2):15–18, July 1993.

17. Thomas W. Sederberg and Eugene Greenwood. A physically based approach to 2d shape blending. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):25–34, July 1992.

18. Karl Sims. Artificial evolution for computer graphics. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):319–328, July 1991.

19. Ivan Sutherland. *Sketchpad: A Man Machine Graphical Communication System*. PhD thesis, Massachusetts Institute of Technology, January 1963.

20. D. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. *CVGIP*, 55:14–26, 1992.

21. Michael T. Wong, Douglas E. Zongker, and David H. Salesin. Computer-generated floral ornament. *Proceedings of SIGGRAPH 98*, pages 423–434, July 1998.

APPENDIX: SIMPLEX SUBDIVISION

We present here pseudocode for subdividing a simplex S with vertices $V = \{v_1, v_2, \ldots, v_n\}$ such that the sub-simplices don't overlap, as discussed in section 4.1. Here a hyperface refers to a simplex S' generated from any $n-1$ vertices of S and an anchor refers to the point from S not contained in S'.

```
Subdivide( simplex S )
    h = an empty stack of hyperfaces;
    b = an empty queue of sub-simplices;
    q = an empty queue of points;

    // S_new is a "corner" of S
    S_new = {v_1 and the midpoint of every edge containing v_1};

    b.add( S_new );
    h.push( every hyperface of S_new);

    while !h.empty()
        face = h.pop();
        q.clear();
        for every point p_i in S // vertices and midpoints
            isValid = true;
            for every hyperface h_new of the simplex
                face.vertices(), p_i
                if h_new has already been encountered and
                does not separate p_i and h_new.anchor
                        isValid = false;
                        break;
            if isValid
                q.add( p_i );
        if !q.empty()
            v = the point in q closest to face
            S_new = v and every vertex in face
            b.add( S_new )
            h.push( every hyperface of S_new )

    return all new edges added in forming the simplices in b
```

Since every hyperface connects to at most two points and these points are on opposite sides, no sub-simplices intersect. Since we attempt to attach every hyperface to two points, the sub-simplices entirely fill the original volume.