# SmoothSketch: 3D free-form shapes from complex sketches

Olga A. Karpenko* Brown University

John F. Hughes† Brown University

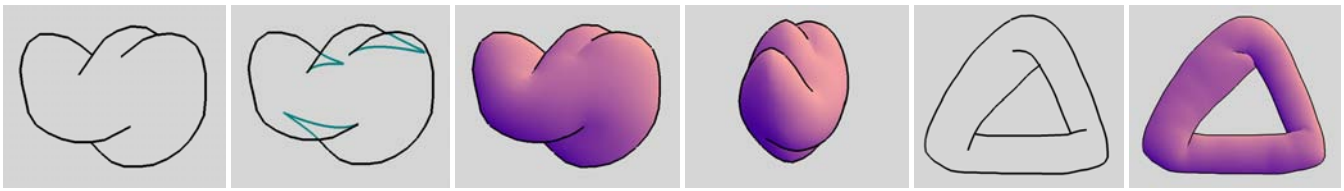

**Figure 1:** The user draws the visible contours of a shape; our program infers the hidden contours, including hidden cusps, and then creates a fairly smooth 3D shape matching those contours. The 3D shape can be viewed from any direction. A smoothed version of Penrose's polyhedral impossible triangle shows that the algorithm can handle objects with complex holes.

## Abstract

We introduce SmoothSketch—a system for inferring plausible 3D free-form shapes from visible-contour sketches. In our system, a user's sketch need not be a simple closed curve as in Igarashi's Teddy [1999], but may have cusps and T-junctions, i.e., endpoints of hidden parts of the contour. We follow a process suggested by Williams [1994] for inferring a smooth solid shape from its visible contours: completion of hidden contours, topological shape reconstruction, and smoothly embedding the shape via relaxation. Our main contribution is a practical method to go from a contour drawing to a fairly smooth surface with that drawing as its visible contour. In doing so, we make several technical contributions:

- extending Williams' and Mumford's work [Mumford 1994] on figural completion of hidden contours containing T-junctions to contours containing cusps as well,
- characterizing a class of visible-contour drawings for which inflation can be proved possible,
- finding a topological embedding of the combinatorial surface that Williams creates from the figural completion, and
- creating a fairly smooth solid shape by smoothing the topological embedding using a mass-spring system.

We handle many kinds of drawings (including objects with holes), and the generated shapes are plausible interpretations of the sketches. The method can be incorporated into any sketch-based free-form modeling interface like Teddy.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling Packages; I.2.10 [Artificial Intelligence]: Vision and Scene Understanding—Shape

*e-mail: koa@cs.brown.edu

†e-mail: jfh@cs.brown.edu



## 1 Introduction

Our visual system, presented with a line-drawing, makes nearly-instant inferences about the shape that the drawing represents. Some aspects of this inference mechanism are surely based on expectation — when we see something that looks like an ear, it's easy to infer that the nearby protrusion must be a nose, for instance. But other aspects depend on local cues — a contour that ends, or that disappears behind another — and a gestalt view that helps us integrate these local cues into a coherent whole [Hoffman 2000].

Dual to this recognition ability is our ability to learn to draw contours of objects in a way that lets us communicate their shape to others. While drawing *well* can be difficult, even children can draw easily recognizable shapes. On the other hand, while drawing outlines or contours is relatively easy, we know few people who can reliably draw the hidden contours of even simple shapes.

When we seek to create shapes with a computer, however, there are few interfaces based directly on drawing; inferring a shape from a complex contour-sketch has generally proved too difficult. The value in doing so, from the sketching point of view, is that it allows a user to draw what he or she is thinking of directly. Teddy's inflation algorithm is a good step, but limited to simple closed curve contours. Our work extends this substantially, although it is by no means a final answer. Such a final answer may never be found, though—it's easy to draw contour sets that are so complicated that different viewers make different inferences about them. The best one can hope for is to create plausible shapes for a fairly large class of contours on which users agree on the interpretation. That is what our work does.

Our work, therefore, is not about a *system* like Teddy; it's about a *component* that can be used in a free-form-sketching interface like Teddy. We believe that a sketching program should let the user and the computer share the work, each doing what it does best. The computer can infer a plausible shape from a moderately complex contour like the ones shown in this paper. Then, to create more complex objects (or to, say, modify the thickness of the inflated models), a user would use various gestures like the ones available in Teddy and other sketch-based systems. We believe we succeeded in the first step, and thus provided a new starting-point for sketch-based systems.

Our system takes a user's contour-drawing of a smooth, compact, oriented, embedded surface-without-boundary (which we'll call a *good* surface) and determines a 3D surface whose contours match those that the user drew. Because this is an under-determined

problem—is that circle a contour drawing of a pancake? a sphere? a cigar viewed end-on?—we aim to generate surfaces that have generally low curvatures, to the degree allowed by the constraints of the contours. The contours must be *oriented*, i.e., drawn so the surface lies on the left. Thus to draw a torus, a user would draw a counter-clockwise outer stroke and a clockwise inner stroke.

Williams' thesis [1994] and subsequent work lay out a plan for finding a surface fitting a given collection of visible contours[1]. The steps involved are

1. *Complete* the drawing by inferring the hidden contours, and provide a Huffman-labeling for it [Huffman 1971].
2. *Convert* the completed-drawing to an abstract topological surface, and map this surface to $\Re^2$ so that the "folds" of the mapping match the contours of the drawing.
3. *Lift* this mapping to a smooth embedding in $\Re^3$ whose projection is the mapping to $\Re^2$.

Having laid out this scheme, Williams then completes several significant parts: he completes step 1 for "anterior surfaces" – roughly the front-facing parts of scenes, which generically have no cusps, and does step 2 for both these *and* for drawings of smooth surfaces. For our purposes, we need step 1 for good surfaces, and we need step 3, which we do in two steps: first we lift to a topological embedding in $\Re^3$, and then we smooth that. We cannot claim to produce a *smooth embedded* manifold; with our current tuning constants, our results are usually immersed (i.e., have self intersections) rather than embedded. Furthermore, the contours of the surfaces we construct cannot in general project exactly to the input drawing, because, for example, the projection of the contour near a cusp always has infinite curvature at the cusp [Koenderink 1990], while our user's input may not satisfy that constraint. We produce a fairly smooth mapping of the manifold into 3-space, whose visible-contour projection nearly matches the user's drawing.

In carrying out step 2, Williams (a) observes that the projection of the complete contours of a generic view of a good surface onto an image plane gives a knot-drawing-with-cusps which can be labeled by Huffman's labeling scheme for smooth surfaces (see figure 2) [Huffman 1971], and (b) gives a method (the *paneling construction*) [Griffiths 1981] to build an abstract manifold $M$ that can be embedded so that its projection has contours matching the knot-drawing. (He does not actually construct an embedding $e : M \to \Re^3$, but instead describes a map $f : M \to \Re^2$ with the property that if such an embedding $e$ exists, and if $P$ is projection onto the drawing plane, then $f = P \circ e$.)

To carry out step 1 for good surfaces, we must establish which drawings of visible contours can be extended to complete contour drawings; not all can, as figure 3 shows. We partially solve this problem by exhibiting a large class of drawings that admit such extensions; the general problem of characterizing extendable visible-contour drawings remains open, however.

For the anterior-surface case, Williams and Jacobs [1997] (and Mumford [1994]) describe an approach to completing the hidden contours, which generically join tee-points in the drawing (see figure 4). To join a pair of tees, they consider all $C^1$ random walks (i.e., random walks in which the tangent direction $\theta$ changes by an amount $X$ at each point, where $X$ is a Gaussian random variable) starting at the first tee, headed in the right direction, and ending at the second, and assign to each a probability based on the product of the probabilities of each angle-change and $e^{-\lambda}$, where $\lambda$ is the length of the curve. They posit that the maximum-likelihood random walk is a good candidate for the completion; when multiple pairs of tees might be joined, they compute which pairings have largest likelihoods and choose those.

[1]The reader interested in implementing the ideas of this paper will need first to become acquainted with Williams' work.

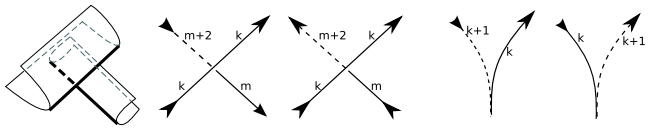


**Figure 2:** Huffman's labeling scheme for contours of generic smooth projections. Labels indicate the number of surfaces in front of the contour (visible contours have label zero); the surface on which the contour lies is to the left when you traverse it in the direction shown by the arrow. *Any* cusped knot diagram—i.e., collection of circles in the plane, possibly intersecting themselves and each other, and smoothly immersed except at finitely many cusp points, which are distinct from the crossing points—that can be so labeled corresponds to the projection of a smooth surface in 3-space (mostly proved by Williams), and all generic projections of smooth surfaces have this property (proved by Huffman). The first picture shows the surfaces corresponding to the first case of Huffman's labeling (the second picture). Figure 6 (left) corresponds to the last case.

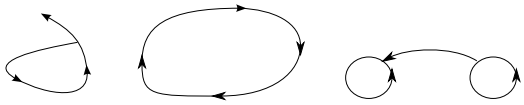

**Figure 3:** None of these drawings can be extended by invisible contours to be the contour set of any good manifold projection. They exhibit two problems: in the first, a cusp appears in the outer region of the plane surrounding the figure. In the second, the outermost path around the drawing is clockwise. Although the third has neither of these problems, it is still not extendable.

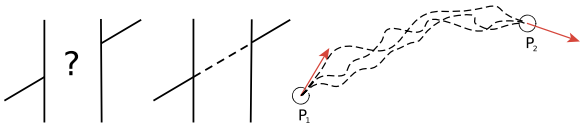


**Figure 4:** How can we join the two tee-points on the left? With an optimal completion, as shown in the middle. Optimality is determined by choosing, among all $C^1$ random walks from $p_1$ to $p_2$, the most likely one, under a simple probabilistic model. Mumford shows such curves are elastica, which had been studied by Euler.

We extend this approach, in Section 4.2, to the cases where a T-point must be joined to a cusp, or two cusps must be joined. To determine which visible endpoints (tees or cusps) should be joined to which, we use a greedy search similar to Nitzberg *et al.* [1993].

To carry out step 3, we take the results of Williams' paneling construction — an abstract manifold and a continuous mapping $f$ of it to $\Re^2$ and "lift" it to a mapping $e$ into $\Re^3$ whose projection is $f$. We construct $e$ a dimension at a time, first placing the vertices of the paneling construction, then embedding the edges, and finally each panel. This algorithm is described in section 5.1. The result is a topological embedding (i.e., a 1-1 continuous map from the surface into $\Re^3$). Finally, in section 6 we talk about smoothing out the creases in this topological embedding by an ad-hoc mass-spring system to produce the desired fairly smooth mesh in 3-space.

## 2 Related Work

**Shape from drawings.** The problem of inferring 3D shape from 2D drawings has been studied in a great many forms; if one extends it to include determining drawings from images as a first step, it occupies much of the computer vision literature [Witkin 1980]. We'll only describe the work most closely related to this paper.

Much early shape-from-drawing work applied to blueprint-like drawings of machined surfaces. The important features of such shapes are sharp bends, like the edges of a cube – and their trihedral intersections. Lipson and Shpitalni [1997] introduced a system in which a user sketches both visible and hidden contours and boundaries of a rectilinear CAD-like geometric object, and the system infers a shape. Their approach is based on correlations among arrangements of lines in the drawing and lines in space.

Pentland and Kuo [1989] presented a system that infers simple 3D curves and surface patches from 2D strokes by minimizing the energy of the corresponding *snakes*.

A classic paper in this area is by Huffman [1971], who developed two labeling schemes—one for objects made from planar surfaces, one for smooth objects—and proved that their complete contour drawings must have the corresponding sorts of labeling. Williams [1994] [1997] did the defining work in inverting the smooth-surface labeling scheme, as described in the introduction.

**Contour completion.** Of course, Huffman labelings are for complete contour projections — the projections of both the visible and invisible parts of an object's contours. Given a drawing of the visible parts of a contour, we must infer where the invisible parts lie. Kanizsa's work [1979] on contour completion (and its relationship to the mechanisms of the human visual system) forms the basis for much of the later work in the area.

A solution proposed first by Grenander [1981] was to use a stochastic process to model the space of all possible edges. Mumford proves that elastica that arise in the completion problem described in the introduction could be modeled by a white noise stochastic process and gives needed formulas. Williams [1997] approximates the solution by considering a sampling of the space of all random walks (with varying $\Delta\theta$ — the direction of the walk) starting from the first point with the first direction and coming to the second point with the second direction, and taking the random walk with the highest probability as the best path connecting two edges.

Although contour completion is a well-studied research topic, many problems are still open; in section 4.2 we propose our solution, inspired by the work of Williams and Mumford, to the problem of finding a hidden cusp for a cusp-contour completion case.

**Sketching interfaces**. Several gestural interfaces for sketching 3D shapes have been developed for different classes of models. For rectilinear objects, the Sketch system described by Zeleznik *et al.* [1996] lets a user create and edit models through gestural interface, where geometric aspects of gestures determine numerical parameters of the objects; a cuboid is created by drawing three lines meeting at a point; the lengths of the lines and position of the point determine the geometry of the cuboid. These ideas were extended by several research groups [Shesh and Chen 2004] [Pereira et al. 2004], and appear in the SketchUp [SketchUp ] architectural design software.

For free-form objects, Igarashi's Teddy [1999] was the first interface for free-form modeling via sketching. In it, a user inputs a simple closed curve and the system creates a shape matching this contour. Then the user can add details by editing the mesh with operations like extrusion, cutting and bending, all done gesturally. The Smooth Teddy [Igarashi and Hughes 2003] system extended this by adding algorithms for beautification and mesh refinement, as well as organizing the shapes into a hierarchy.

Karpenko *et al.* [2002] described a system for creating shapes from free-form sketches; the primitive objects were variational implicit surfaces, which facilitated operations like surface blending. ShapeShop [Schmidt et al. 2005] uses hierarchical implicit volume models to let a user interactively edit complex models via a sketching interface. Alexe *et al.* [2005] extract the skeleton from the sketch and then construct a convolution surface. None of these systems handle complex strokes containing tees and cusps.

Nealen *et al.* [2005] presented a sketch-based interface for laplacian mesh editing where a user draws reference and target curves on the mesh to specify the mesh deformation. A similar interface was developed by Kho and Garland [2005] for posing 3D characters, in particular, bodies and limbs.

Finally, Karpenko and Hughes [2005] demonstrated a method for inferring certain free-form shapes from sketches by detecting 'templates' in the sketches and building a part of the 3D surface from a standard recipe for each template.

**Pseudo-3D models**. Tolba *et al.* [2001] describe a system that lets a user draw a scene with 2D strokes and then view it from several new locations as if a 3D scene had been created. This is done by projecting the 2D strokes on a sphere centered at the eye point and then viewing them in perspective. Bourguignon *et al.* [2001], describe a system that takes a set of 3D strokes representing contours and creates a small piece of surface near each stroke whose contour is the given stroke; contours of this surface, seen from nearby viewpoints, give the appearance of a full-fledged 3D model, although in distant viewpoints the illusion is lost. Johnston [2002] computes lighting on 2D drawings without reconstructing 3D geometry by estimating surface normals from the drawing.

**Shape from contours for special classes, and other shape-from methods.** Ulupinar *et al.* [1995] solve the "shape-from-contour" problem for images by considering only a special class of symmetrical 3D shapes: straight homogeneous- and constant cross section generalized cylinders. Apart from inferring the shape from the contour, researchers have long tried to infer shape from texture, shading, and other cues. An overview of some of these methods can be found in the paper by Ulupinar [1993].

## 3 Notation and problem formulation

Much of the material that follows relies on ideas from differential geometry and combinatorial and differential topology. We refer the reader to the books of Guillemin and Pollack [1974] and Koenderink [1990] for clear expositions of the necessary background.

Suppose that $S$ is a smooth, closed, compact, orientable surface-without-boundary (i.e., a *good* surface) embedded in the $z > 0$ half-space of $\Re^3$. The orthogonal projection of $S$ onto the $z = 0$ plane will have a compact image. Following Williams and Mumford, we will assume that the embedding and this projection are *generic*, i.e., that no probability-zero events occur, e.g., no projector meets three contours, no cusp projects to a point on another contour, etc. If the projector through the point $s \in S$ lies in the tangent plane at $s$, then $s$ is called a *contour point*; if the projector first meets $S$ at $s$, then $s$ is a *visible* contour point (see figure 5). For a generic projection, the set of all contour points forms a compact 1-manifold-without-boundary $C$ in $S$, i.e., a collection of disjoint topological circles in $S$. The set of visible contour points form a compact 1-manifold-with-boundary, $V$ in $S$, i.e., a collection of disjoint topological circles and

line-segments. The projection of $C$ to the $z = 0$ plane is the *contour drawing* of $S$; the projection of $V$ to the $z = 0$ plane is the *visible contour drawing* of $S$.

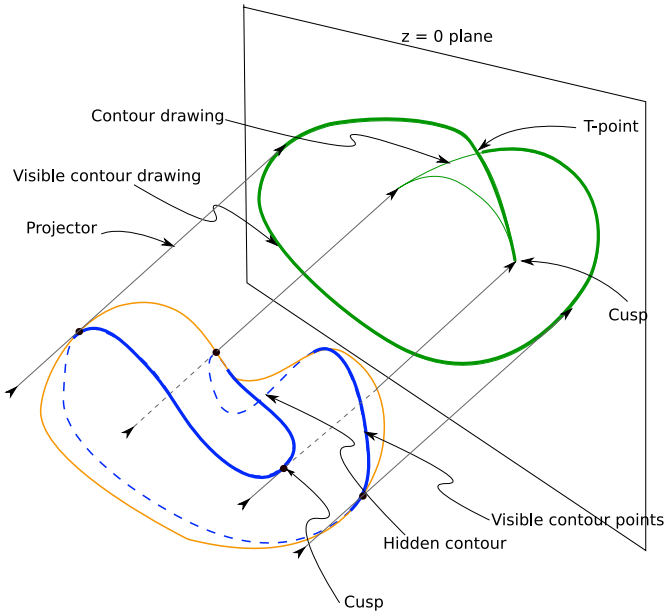


**Figure 5:** (Adapted from Williams' [1997]) The contour, in blue, of a good surface embedded generically in 3-space projects to a contour drawing, in green; the visible contour (drawn bold) projects to the visible-contour drawing. A point where the projector is tangent to the contour projects to a cusp in the contour drawing. The restriction of the projection to just the contour is 1-1 except at finitely many points, where two contours cross in the drawing; these are called T-points.

The projection from the contour to the contour drawing is an embedding at most points; the exceptions are *crossings*, where two contours meet, and *cusps*. A cusp is a point $s \in S$ where the projector through $s$ is tangent to $C$ at $s$. The projection of a cusp appears as a point where the contour drawing "reverses direction" (see figure 6, left). When an arc of the visible contour drawing reaches a crossing, it appears as a *T-point*: one part of the contour becomes invisible there.

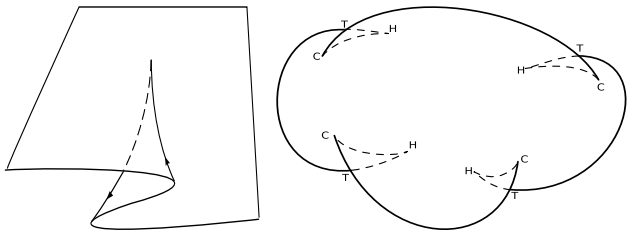


**Figure 6:** (Left) The generic projection of a contour at a cusp reverses direction at the cusp. (Right) A drawing with the tee-points and cusps marked; hidden contours and hidden cusps that must be inferred are shown in dotted lines.

For a generic smooth surface and viewpoint, tees and cusps of the contour will be isolated, as will curvature zeroes of the contour; this guarantees a unique osculating plane at a cusp, which means the projected contour must reverse direction rather than emanating from the cusp in any other direction (see figure 6, left).

The "bean" example (figure 5) is something of an archetype for the method described in this paper, in the sense that it's the simplest shape that has a cusp; the way that this single cusp is processed is the key to processing more general drawings, hence we use the bean as an example throughout.

In Figure 6 (right) we show in solid lines a typical input drawing; in dotted lines are the projections of invisible contours. Certain hidden contour points are also cusp-points; the visible cusps are marked with a "C" while the hidden cusps are marked with an "H".

Note that the user input is the part of the contour drawn in solid lines. Everything marked by a dashed line is a part of a *hidden* contour and needs to be inferred by our program.

With this terminology, our goal is to take a user-provided directed visible-contour drawing of a good surface as above and to determine a surface $S$ whose visible contours match the given drawing. Note that we do not seek to reconstruct exactly the surface that the user was drawing; the map from surfaces to drawings is many-to-one, hence non-invertible. Note too that we require that the drawing arise from *some* surface, so that the problem has at least one solution; a drawing consisting of a single line segment, for instance, cannot be the projection of the visible contours of any surface.

Our system currently produces *a* surface consistent with the user's drawing, and one which we generally find to be plausible. Eventually, we would like to solve a more general problem: we want not only to produce *one* of the reasonable-looking shapes, we would like to return *the most natural shape*. Of course, "most natural" can be very subjective and depend on a user's preferences, but experience shows that people generally agree about what a drawing conveys. We could take cartoon illustrations (see figure 7) as an example. These pictures vary from simple to very sophisticated, but their expressiveness is such that people interpret them immediately. Such a degree of "naturalness" (or indeed, any way of measuring it) appears to be a very long-range goal.

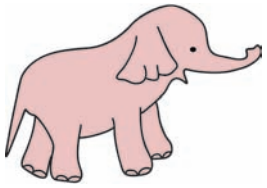

**Figure 7:** This cartoon-like illustration shows us how even the simplest drawings can have complex contours.

## 4 Figural Completion for Smooth Surfaces

Given the visible contour drawing, in the $z = 0$ plane, for a good surface in $\Re^3$, we describe an approach to completing the drawing, i.e., adding hidden contours so that the resulting drawing can be Huffman-labeled. The approach works in a large number of cases, although not all. We begin by showing a construction that provably works for a large class of drawings, but often produces "unlikely" completions according to the Williams-Mumford measure of likelihood; we then describe our actual implementation, which approximates the construction while preferring 'more likely' completions that occasionally lead to problems.

### 4.1 Completable drawings

Although the class of visible-contour drawings that can be completed has not been characterized, to the best of our knowledge, we can demonstrate that a large class *can* be completed. Once again we consider the 'bean' as our example (see figure 8). Taking a directed visible contour as input, we consider the regions into which it divides the plane; traversing the boundaries of these regions, and pushing slightly into each region, gives a collection of disjoint embedded curves (which we call *red curves*), each of which may pass by some number of T-points or cusps. We give the key steps in an argument that if (a) the curve for the outermost region passes no such points, and has turning number one (i.e., can be smoothly deformed to a counter-clockwise circle) and (b) all other curves pass an equal number of "starting" and "ending" points, then there is a Huffman-label-able completion of the visible contour. The formal proof involves careful application of the tubular neighborhood theorem, the isotopy extension theorem, and other standard techniques from topology; we present only the essential insights here.

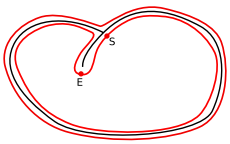


**Figure 8:** The visible contours of the bean divide the plane into two regions; traversing a path slightly displaced from the boundaries of these regions gives the two red curves shown. The inner one passes the cusp and then the T-point (we don't count where it makes a turn at the T-point as "passing" it); the passages are marked with dots. The outer one passes no endpoints at all.

The first step is to assign depth 0 to all visible edges. Now consider one of the red curves that meets at least two endpoints. Starting from any point of the red curve, we traverse it, noting whether the points we encounter are "starting points" (S) or "ending points" (E) of the visible contour arcs. The resulting circular sequence of Ss and Es contains at least one of each, by hypothesis; there must therefore be an adjacent pair of points, one a starting point and one an ending point. We'll show how to remove these from the sequence by completing the two contours; induction then shows that all contours can be completed and we are done.

The adjacent S and E points can be either cusps or Ts. Figure 9 shows how these can be joined. In the cusp-tee case, we can add a hidden contour and a hidden cusp, all within the region between the red curve and the contour between the two points being processed; after this addition, the red arc can be redrawn; the points S and E are no longer arc endpoints, and thus the start-end sequence for the arc is now two characters shorter. Similarly, in the T-T case, we can add a short completion arc. The only remaining case is the cusp-cusp case. Depending on whether the cusps appear in S-E order or E-S order, one of two standard solutions shown provides the necessary completion.

We note that there are drawings that do not satisfy the criterion above, but which nonetheless admit completions, so this class, although large, is not exhaustive.

### 4.2 Practical contour completion

The completions described in the previous section are formally correct, but since many of them have sharp turns in the hidden contours, they are, from the Williams-Mumford random-walk perspective, unlikely. As a practical matter, therefore, we take a different approach in our program: we consider all visible-contour endpoints, and estimate the likelihood of a hidden contour joining each possible pair. Following Nitzberg *et al.* [1993], we pair up points using their greedy algorithm, testing multiple configurations for (a) probability, and (b) consistency (can they be Huffman-labeled?); if the most-likely configuration is inconsistent, we move to the next-most-likely, and so on.

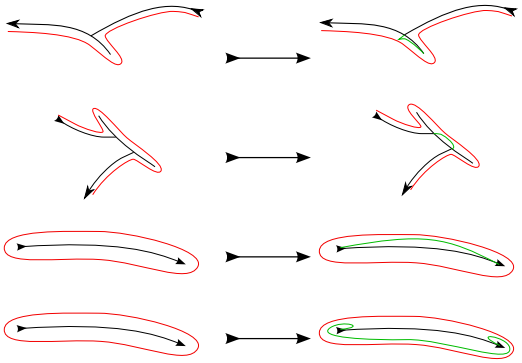

**Figure 9:** Adjacent cusp-T pairs can be joined with two arcs and a hidden cusp; adjacent T-T pairs can be joined with a single arc, as can adjacent cusp-cusp pairs; the side on which the arc lies, in this case, depends on the order in which the two cusps were encountered.

**Pairwise completions.** First, for each pair of endpoints of the visible contour we compute an initial estimate of the probability that they are connected by a hidden contour. Each endpoint has a location and associated direction for the completion curve (for T-points, the direction is given by the tangent ray of the visible contour; for cusps it is the opposite). To compute the likelihood of joining two tees or two cusps, we compute an energy function for the pairing, inversely proportional to the likelihood. The energy function of the pairing is a sum of two energy functions $E = E_{curve} + E_{endpoints}$, where $E_{curve}$ is the energy of the curve that would connect them were they to be matched and $E_{endpoints}$ is the energy corresponding to the heuristic defined by the endpoint tangent directions. We now describe each in detail. First, we approximate the elastica curve with a Bézier spline connecting the endpoints given their tangent vectors. The Bézier curve is defined by the two endpoints and the points displaced from the endpoints along the tangent vectors. The distance by which the endpoints are displaced along the tangents is $\frac{1}{3}$ of the distance between the endpoints. The Bézier curve is then uniformly sampled and the energy function of the resulting polyline is computed as follows (see figure 10):

$$E_{curve} = e^{\Sigma_i l_i} \cdot \sum_i \Delta\theta_i$$

where $l_i$ is the length of the $i$-th segment of the polyline, and $\Delta\theta_i$ is the absolute value of the angle change between two consecutive segments of the polyline. $E_{endpoints}$ corresponds to another heuristic similar to [Nitzberg et al. 1993], where we use the tangents at the endpoints to estimate the likelihood of the matches. Intuitively, if the tangent directions at the two endpoints are very similar, it is likely for them to be paired even if the length of the curve connecting them would be long (think of a fat snake whose tail passes behind its body). Similarly, if the tangents at the endpoints are very different, it should be pretty unlikely for them to be paired up. Currently, $E_{endpoints}$ is a constant 1.0 if the angle between the tangents at the endpoints is between $\varepsilon_1 = 0.3$ and $\varepsilon_2 = 2.5$, 0 if the angle between them is $\leq \varepsilon_1$ and proportional to $e^{angle}$ if the angle is $\geq \varepsilon_2$.

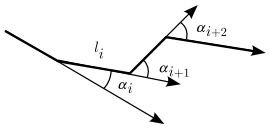


**Figure 10:** The energy of the polyline approximating the Bézier spline is computed as a product of the sum of angle changes between the consecutive segments and the exponent of the sum of the segment lengths.

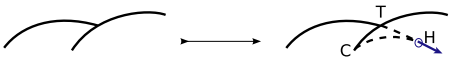


**Figure 11:** When we have a T-point and a cusp to match it to, we seek the location of a hidden cusp such that the two hidden contour parts joining our points to the hidden cusp have the highest probability.

When we want to join a T and a cusp (see figure 11), we ask the question "for all possible locations of a hidden cusp, and all possible tangent directions there, what are the probabilities of a $C^1$ random walk joining the T to this cusp and of another joining the visible cusp to the hidden cusp?" We treat the product of these probabilities as the probability of this point-and-direction being the hidden cusp. We posit that the ideal location of the hidden cusp is the one with the highest probability. Unfortunately, computing this probability directly by generating many random walks, etc., is impractical and slow. We therefore did this once, offline, and stored the precomputed probabilities in a table. That is to say, we placed one point at the origin, with a tangent ray along the positive x-axis; we generated many (ca. $10^8$) $C^1$ random walks from there and recorded (in a discretized form) where each ended, and in which direction it was going. We did the same for a second point, situated on a unit circle, and with a given initial direction; we then found the point and direction that was most likely to be an endpoint of both sets of random walks. This was repeated for multiple points on the unit circle, and multiple initial directions, and the results stored in a table. This table, then, represents the function $(P,\phi) = h(Q,\theta) =$ the location and direction of the hidden cusp when one point is at the origin and has horizontal tangent, and the other is at $Q$ and has tangent direction $\theta$. Given an actual tee and cusp, we can translate one to the origin and rotate so that its tangent is in the positive-$x$ direction; we then use the other point's location and direction to look up an answer in the table. (Note that this assumes that the optimal answer is scale-invariant, in that the second point may not be at a unit distance from the first, and we must scale this distance to one in order to use our table.) We connect the hidden cusp to the tee point and the visible cusp with Bezier curves, and compute $E_{curve}$ for their union as described above.

**Greedy search for the best configuration.** After a likelihood for each pair of endpoints is computed, we need to match up pairs to find the best total *configuration* (a configuration consists of endpoint pairs, where each endpoint appears in only one pair). For instance, if we have 4 endpoints numbered 1 to 4, the possible configurations are: $\{(1,2),(3,4)\}$, $\{(1,3),(2,4)\}$ and $\{(1,4),(2,3)\}$. The likelihood of a configuration is defined as the product of likelihoods of its pairs. It is not practical to compute the likelihoods of all possible configurations as the number of them grows exponentially in the number of tees and cusps. Instead, we do a greedy search similar to [Nitzberg et al. 1993]; starting with several best pairs, for each of them we choose the next best pairs from the set of valid configurations, and so on; we keep track of the 10 best configurations at any time.

#### 4.2.1 Limitations of the figural completion algorithm

The figural completion approach that we presented has a number of limitations.

The location of the hidden cusp provided by the method above may be unsatisfactory. Indeed, in the bean-like case shown in figure 12, the hidden cusp is estimated to lie at a point that is not, in fact, hidden. Figure 12 (right) shows another example where the locations of hidden cusps are estimated incorrectly because of the failed assumption that the precomputed positions of hidden cusps are scale-invariant.

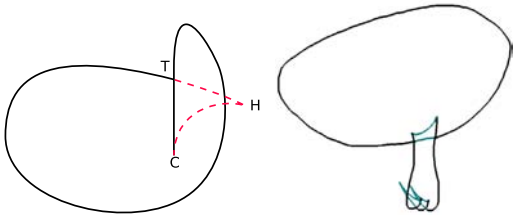


**Figure 12:** Problem cases: our method can produce a contour completion which places the hidden cusps in impossible locations. This happens because our method only considers local probabilities, and not the shape of the remainder of the visible contour.

Consider a dog's body with one leg on the left hand side, seen from the right hand side (see figure 13). This is a case that our contour-completion algorithm cannot handle. The "completion" of the obscured contours consists of two hidden cusps connected by a U-shaped hidden contour, and two "straight" segments connecting the hidden cusps to two t-points. In the two-hidden-cusp completion, the location of the two cusps is ambiguous. The algorithm for finding a hidden cusp for a t-point/visible-cusp pair will not work for this case, because there are no visible cusps.

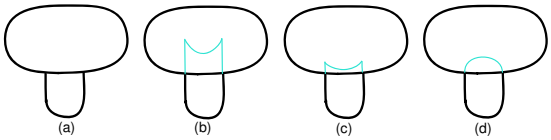


**Figure 13:** (a) The back-leg drawing case; (b),(c), and (d) show possible completions; our system would produce completion (d), but would fail at later stages (although if we allowed multiple-component surfaces, then (d) could be created).

The figural completion for this case could equally well consist of just an arc joining the two t-points — there's no a priori reason for the system to assume that the shape being drawn has only one connected component. Without the context (knowing that this is a leg), we do not know of any principled algorithm to guess the locations of the hidden cusps.

The back legs issue is something that can be handled pretty easily by adding gestures to our system (where, say, a user could change the view and draw a stroke corresponding to the back leg), or, by incorporating our system as an inflation component into one of the existing free-form sketch-based interfaces like Teddy. Such a system would also ideally include the ability to sketch or edit the hidden contours (i.e., provide user-guidance to the optimization algorithm).

Our contour-completion algorithm, based on the table-lookup, should probably be improved. We would like to find a good approx-

imation to the data in the table so that a lookup (and the computation and storage of the table) is unnecessary; if such a function were based on a deep understanding, the unprincipled "scale-invariance" assumption might also be eliminated.

Given that figural completion is an expensive search problem, it becomes slower for a large number of tee/cusps (say, more than 15) and is more likely to make incorrect inferences as the drawing gets more and more complicated (we find only an approximate solution to the optimization problem to make it tractable; as a result, the approximate minimum is not always the global minimum).

We have shown that a large class of visible contour drawings admit completions; it is known that others do not. But we do not know a complete characterization of which drawings admit completions.

## 5 From drawing to topological embedding

At this point we have a completed contour drawing, with a Huffman-labeling. This drawing consists of directed arcs (which we call *edges*) between *vertices* corresponding to T-points and cusps. The drawing partitions the plane into regions; Williams' paneling construction tells how to take a disjoint union of multiple copies of these regions and identify edges in pairs to produce an abstract manifold. Williams' description misses one subtlety: the regions he is identifying must be closed sets so that they contain their boundary points, for it is boundary points that are identified in pairs. For regions like the large region of the bean, the "crossing point" at the top must be counted twice – once as a point on the left half of the contour, and once as a point of the right half, or the object resulting from the identification of edges will not in fact be a manifold. This can be addressed by examining the boundary of each region for self-intersections and, if any are present, subdividing the region into two smaller regions; the details are finicky but not difficult. We'll simply treat that crossing point as two ever-so-slightly-separated points for the purpose of this explanation.

In our implementation, each copy of the region at this point is a 2D mesh created by triangulating the boundary of the region. We use Triangle [Shewchuk 1996] which performs the constrained Delaunay triangulation algorithm on the given boundaries of the regions.

We consider (see figures 14, 15) the disjoint copies of a region $R$ as being of the form $R_i = R \times \{i\}$, where the index $i$ never appears more than once in all copies of all regions. A typical identification in Williams' scheme is then that the point $(r,i)$ is identified with $(r,j)$, where $r$ is a point on the boundary of region $R$, and $(r,i)$ and $(r,j)$ lie in $R_i$ and $R_j$ respectively; another might be that $(r,i)$ is identified with $(s,j)$, where $R$ and $S$ are adjacent regions in the plane both containing the point $r = s$ on their boundaries, $(r,i) \in R_i$ and $(s,j) \in S_j$. The disjoint union of all the copies of all the regions will be called $U$; there's a natural map $\pi : U \to \Re^2 : (r,i) \mapsto r$ in which the multiple copies of any point $r$ are all mapped to $r$.

For a point $P$ in the plane, the set $\pi^{-1}(P)$ is a set of points of the form $(P,i)$; we call this the "stack over $P$." Similarly, we can consider the stack of edges over an edge in the plane, or the stack of panels over a panel in the plane. If an edge $e$ in the plane goes from $P$ to $Q$, we write $\partial e = (P,Q)$ to denote that the boundary of edge $e$ consists of the points $P$ and $Q$, in that order. If $e_i$ is an edge in the stack over $e$, then $\partial e_i = (P_i, Q_i)$ as well.

Williams identifies certain panel edges in pairs (see figure 15), that is, for certain $i$ and $j$, he declares that $e_i$ is to be identified with $e_j$, which means that the point $(x,i) \in e_i$, is identified with the point $(x,j) \in e_j$. This identification induces an identification on the stacks above vertices: if $e_i$ is identified with $e_j$, and $\partial e = (P,Q)$, we declare $P_i \sim P_j$, and $Q_i \sim Q_j$. The transitive closure of the relation $\sim$ partitions stacks into equivalence classes that we call *clusters*; each cluster in each stack corresponds to a vertex in Williams' surface, which we'll eventually embed.

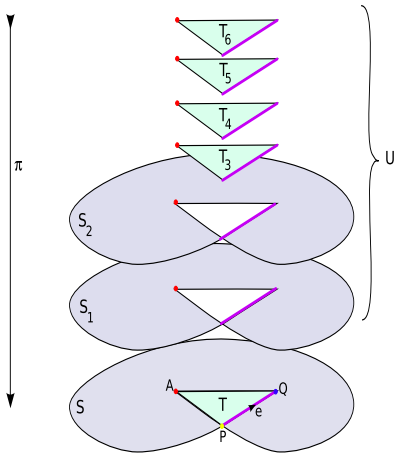


**Figure 14:** Schematic view of the disjoint union of panels that are glued to form the topological manifold homeomorphic to the bean. Each copy of each panel lies in a different layer; the union of all these copies is called $U$. The map $\pi$ is "projection back to $\Re^2$ along $z$." The collection of all points that project to $A$ (the red dots) is called the "stack above $A$". The magenta edges are the stack above the edge $e$. Each panel is indexed by its height in $z$, so all panels have different indices.

**Ordering the clusters**. Williams' construction gives a depth order to the panels in each panel-stack; this order is generally unrelated to the indices above. This order induces an order on the clusters as follows: if $P_i$ and $P_j$ are in two clusters, and $R$ is a region containing $P = \pi(P_i) = \pi(P_j)$ consider all the faces in the stack over $R$ that are adjacent to vertices in the first cluster, and all those adjacent to vertices in the second cluster. By Williams' construction, faces in the first group will either be all in front of or all behind the faces in the second group; we say that the first cluster is in front of or behind the second group accordingly. Again by construction, this order is independent of the adjacent region $R$ that we choose.

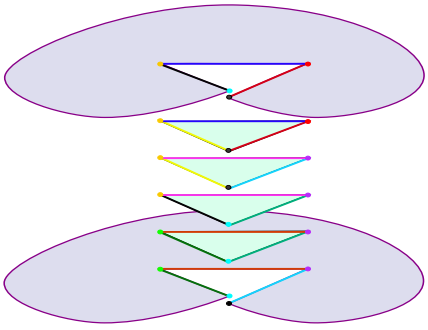

**Figure 15:** The panels, re-ordered for visibility; edges with the same colors are identified. This identifies *clusters* of vertices in each stack; vertices with the same color form a cluster. Note that the near vertex in the two large panels has been split into two copies.

**Extra vertices.** One important issue remains: if two edges $e$ and $e'$ in the same edge-stack have the same clusters as their endpoints but are *not* identified in the topological manifold, these distinct edges would be assigned the same depth in the constructed surface, which

would result in a non-embedding. Figure 16 shows two such edges in the lower portion of the leg case. In such cases, we add a new vertex at the midpoint of each of the edges $e$ and $e'$ of the contour (and to any other edges that are identified with these). The stacks and the clusters within these stacks are then created for these newly-inserted points in the same way we described above.

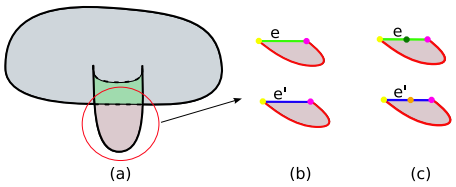


**Figure 16:** (a) A contour-completed drawing of a leg attached to a body, with panels colored. (b) The two panels for the bottom of the leg, colored to show edge identifications and vertex clusters. Note that the top edges $e$ and $e'$ share endpoints but are not identified. (c) We add mid-edge vertices, sort, and cluster them as before.

### 5.1 Constructing a topological embedding

We now present a novel algorithm that constructs a topological embedding from Williams' abstract manifold.

**Embedding vertices** To each cluster of the vertex stack over a vertex $P$, we associate a vertex whose $xy$-coordinates are those of $P$, and whose $z$ coordinate is yet to be determined (we call these *cluster vertices*). We determine the $z$-placements using a mass-spring system. Suppose that the vertices corresponding to the clusters of one stack are $X_\alpha$, where $\alpha$ ranges over the clusters. If cluster $\alpha$ is behind cluster $\beta$, we want the $z$-coordinate, $z_\alpha$ of $X_\alpha$ to be less than that of $X_\beta$. For each such order-relation between two of the $X$s, we attach a spring whose rest-length $z_0$ is one, and for which the spring force follows the rule

$$F(d) = \begin{cases} 0 & d \geq 1, \\ Ce^{1-d} & d < 1 \end{cases}$$

which ensures that if the $z$-order is inverted, there's a substantial force pushing back towards the proper ordering.

This ordering and set of $z$-values could also be found by simply sorting the vertices; we use the mass-spring system as a way to relate the $z$-depths for vertices in separate stacks. In particular, if $P$ and $Q$ are distinct vertices joined by an edge $e$, then each cluster over $P$ is joined to one or more clusters over $Q$ by edges in the stack over $e$. For each such connection, we add a spring with rest-length zero between the corresponding cluster-vertices; we use a sufficiently small spring constant that the intra-stack ordering is not disturbed. Our goal is to make each edge want to be somewhat parallel to the $z = 0$ plane, rather than having vertices associated with one stack be far in front of all others, for instance.

The mass-spring system acts on the points, which are constrained to move only in $z$. Clearly if the spring constant for the inter-stack springs is small enough, each stack will be ordered correctly. In our implementation, we use the constant 1.3, which seems to perform well on examples like the ones shown in this paper and the associated video. The points of the drawings in our system lie in the bounding box of $-1.0$ to 1.0 in each direction.

**Embedding edges** Having embedded the cluster vertices (i.e., the vertices of the manifold that Williams constructs), we can extend the embedding to edges by linearly interpolating depth along each edge. The ordering of edges in Williams' construction is generally sufficient to show that if $e_i$ and $e_j$ are distinct edges of the manifold corresponding to contour edge $e$, then they do not intersect except, perhaps, at endpoints which they share. In the event that $e_i$ and $e_j$ share *both* their endpoints, linear interpolation would assign them the same depths at all points, and our mapping would not be an embedding. Fortunately, the "extra vertices" step above inserts points exactly when necessary to prevent this; thus we have an embedding of both the vertices and the edges of Williams' manifold.

**Embedding faces** We extend the embedding over the panel interiors using Poisson's formula to find a harmonic function on the panel whose values on the boundary are the given depth values that we've already assigned to the edges of the panel. Each interior point is assigned a depth that is a weighted average of the depths of points on the boundary edges. To prove that two panel interiors in the same stack never intersect, suppose that $P$ is a point of some panel $R$, and that $X$ and $Y$ are points in the panel-stack over $R$, and that $\pi(X) = \pi(Y) = P$. Suppose that the panel to which $X$ belongs, $R_i$, is in front of the panel to which $Y$ belongs, $R_j$, so that the $z$-value for $X$ should be larger than the $z$-value for $Y$. Then points on the edges of $R_i$ are in front of (or equal to) the corresponding points on the edges of $R_j$. The $z$-coordinates of corresponding points cannot all be equal unless the boundaries of $R_i$ and $R_j$ are identical, in which case the union of $R_i$ and $R_j$ is a spherical connected component of the manifold, and is handled as a special case. In the remaining cases, since the $z$-values for $R_i$ are greater than or equal to the corresponding values for $R_j$, and the $z$ value for $X$ is a weighted sum of these values *with all nonzero weights*, and the $z$-values for $Y$ is the corresponding weighted sum of the other $z$-values, with the same weights, we find that the $z$ value for $X$ is strictly greater than that for $Y$. Thus the interiors of faces do not intersect. We have thus constructed a continuous 1-1 map from Williams' abstract manifold into $\Re^3$, i.e., a topological embedding.

## 6 Smoothing the embedding

Now that mesh vertices corresponding to each panel have been assigned depths, we "stitch" the meshes of individual panels into a single mesh. We start with the first panel, and stitch panels to it one at a time. If two panels are identified along an edge $e$, we alter the vertex indices on second to match those of the first. The edge correspondences for the stitched panel (excluding the edge we stitched along) come from the correspondence information of the two component panels. Although the resulting stitched mesh has the proper "contour projection" (for an appropriately modified definition of "contour"), its shape is generally unsatisfactory, as can be seen in the accompanying video. We therefore perform several optimization steps. During these steps, we constrain the vertices lying on the visible silhouette to remain on the silhouette so that the contours will match the drawing. That is, these vertices can only move in $z$, while others may move in $x, y$, and $z$.

First, we remesh the model using the algorithm proposed in [Kobbelt et al. 2000] in order to create more regular triangles, as the behavior of the mass-spring system is sensitive to the quality of the triangulation. Then, ten iterations of Taubin's $\lambda/\mu$ smoothing [1995] are applied to the mesh.

At this point, the mesh is smoother, but rather flat and sharp along the edges (because the silhouette constraints have not been incorporated smoothly). The next goal is to "inflate" the model, making it more rounded. To achieve this, we construct a mass-spring system on the initial mesh, with masses at the vertices and with two types of springs: length springs and what we call "pressure force" springs. The length springs try to keep the length of each edge as close to zero as possible, while the "pressure springs" simply push

each triangle outward along its normal with a force proportional to the area of the triangle. We relax this mass-spring system and although the convergence in general is not guaranteed, in practice it converges quite fast. A model like the ones shown in the paper inflates in several seconds on an AMD Athlon 64 3000+ processor.

Our mass-spring approach has several drawbacks; some of them common to all mass-spring systems, others are particular to our choice of springs. First, most mass-spring systems approximate the physics of deformable models very crudely. Further, in our case, even the underlying "physical" model is quite *ad hoc*. We intuitively think of the current model as inflating the initial flat shape as a balloon, but with the restriction on the movement of silhouette points and disregard for surface curvature, it is a very weak analogy.

Secondly, our mass-spring system has several tuning constants that have to be chosen so that they work for most of the examples user draws. Thirdly, there is currently no mechanism in the system to prevent self-penetrations of the surfaces. We would like to address this issue in the future. Sometimes, though, we would like to allow self-penetrations: think of a body-with-two-legs example; there, the legs being slightly pushed inside the body is often more desirable than having them stick out far away from the body. Finally, there is a known problem of stiffness [Gibson and Mirtich 1997] with all mass-spring systems that leads to their potential instability.

Having said all this, the mass-spring system we created seems to work reasonably well on most examples. The results of the relaxation of the mass-spring system are satisfactory except at the areas that were completely flat and skinny initially (like the tips of the legs). Finally, we may choose to apply a few iterations of Taubin's anisotropic smoothing [Taubin 2001], which first filters the normals using $\lambda/\mu$ algorithm, and then filters the vertices, integrating the new normal field in the least squares sense. The final results are shown in figure 17.

## 7 Discussion, Limitations, Conclusions

Our system creates 3D shapes for a wide class of contour drawings; certain limitations prevent it from working universally. One is that the contour-completion approach is local–the completed contour shape depends on the geometry of the starting and ending points, but ignores the remainder of the input shape; it will require a much deeper understanding of contour completion to address this.

Williams' topological manifold construction, followed by our lifting, creates a mesh embedding with "folds" matching the drawn contour. But mesh contours and smooth contours are different. In particular, the curvature of a smooth contour at a cusp goes to infinity, which a mesh-cusp simply projects to some non-zero angle in the contour-drawing. The problem of exactly fitting the drawing is therefore generally impossible, unless users respect the conditions on curvature at cusps. We need to develop a means to characterize when we have adequately approximated the user's drawing.

Our inflation algorithm currently requires tuning constants; the constants that produce the most satisfactory-looking results actually produce self-intersecting surfaces, especially in locations like "armpits" (i.e., between a limb and a body). We would like to find an algorithm that produces embeddings instead. A more principled approach would optimize something about expected shapes of the inflated surface, conditioned on the known shapes of the visible contours, but lacking a prior distribution on all smooth surface shapes, such an approach seems intractable. We anticipate that a minimization of some fairness functional might hold promise.

We would also like to extend our work to include minor surface discontinuities—things like ridges or creases on a surface, which often are perceptually significant (and indeed, in cases like armpits, creased shapes are what a user might want to create).

Finally, although we have developed our system to be agnostic about shape, treating it purely geometrically, users *are* familiar with many shapes. We imagine the possibility of a hybrid system, in which the user's sketch is both inflated *and* matched against a large database of known forms, for possible suggestions ("You seem to be drawing a dog; would you like us to add the hidden legs for you?"). The problems of searching such a database and forming reliable hypotheses, however, seems daunting.

## Acknowledgements

We would like to thank Prabhat, Gabriel Taubin, Anatoly Karpenko, Tomer Moscovich and Peter Sibley for many useful suggestions and helpful discussions, and the reviewers for their feedback. This work was partly supported by a gift from Pixar Animation Studios.

## References

ALEXE, A., BARTHE, L., CANI, M.-P., AND GAILDRAT, V. 2005. Shape modeling by sketching using convolution surfaces. In *Pacific Graphics*, Short paper.

BOURGUIGNON, D., CANI, M.-P., AND DRETTAKIS, G. 2001. Drawing for illustration and annotation in 3D. *Computer Graphics Forum 20*, 3, 114–122.

GIBSON, S., AND MIRTICH, B. 1997. A survey of deformable modeling in computer graphics. Tech. Rep. TR-97-19, Mitsubishi Electric Research Lab., Cambridge, MA.

GRENANDER, U. 1981. *Lectures in Pattern Theory*, vol. 1-3. Springer-Verlag.

GRIFFITHS, H. B. 1981. *Surfaces*. Cambridge University Press.

GUILLEMIN, V., AND POLLACK, A. 1974. *Differential Topology*. Prentice Hall.

HOFFMAN, D. D. 2000. *Visual Intelligence: How We Create What We See*. W. W. Norton.

HUFFMAN, D. A. 1971. Impossible objects as nonsense sentences. In *Machine Intelligence 6*, B. Meltzer and D. Michie, Eds. American Elsevier Publishing Co., New York.

IGARASHI, T., AND HUGHES, J. F. 2003. Smooth meshes for sketch-based freeform modeling. In *Symposium on Interactive 3D Graphics*, 139–142.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH 99*, 409–416.

JOHNSTON, S. F. 2002. Lumo: illumination for cel animation. In *Proceedings of the Symposium on Non-photorealistic animation and rendering*, 45–52.

KANIZSA, G. 1979. *Organization in vision*. Praeger, New York.

KARPENKO, O., AND HUGHES, J. 2005. Inferring 3d free-form shapes from contour drawings. In *Siggraph 2005 Sketches Program*.

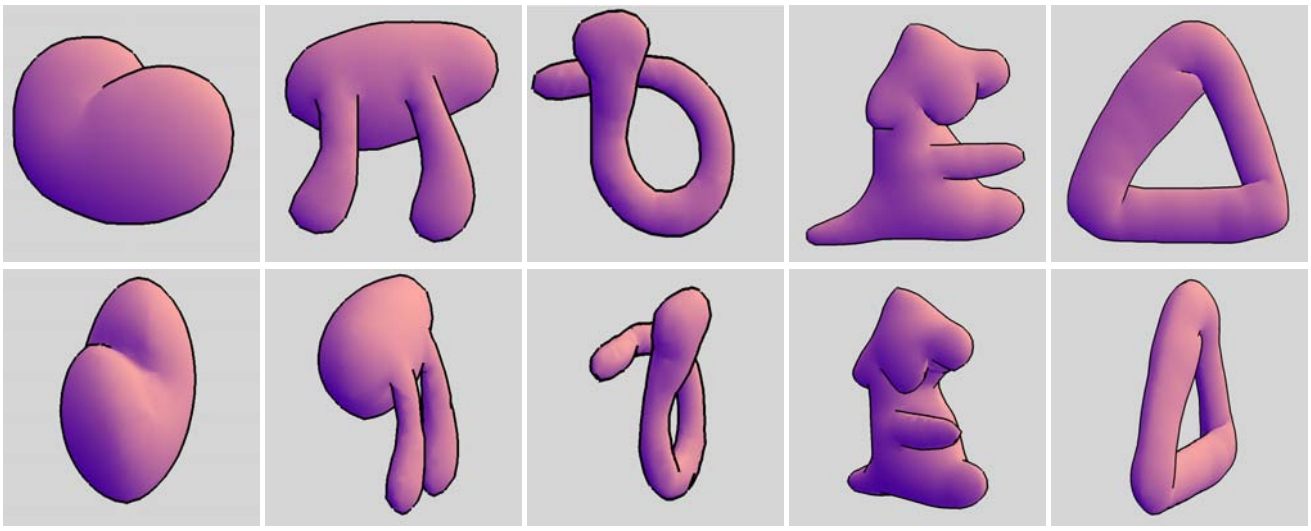

**Figure 17:** The examples of the shapes created with our system from user drawings. The top row shows the shapes from the sketching viewpoint, and the bottom row shows them from a different view.


KARPENKO, O., HUGHES, J., AND RASKAR, R. 2002. Free-form sketching with variational implicit surfaces. In *Eurographics Computer Graphics Forum*, vol. 21/3, 585–594.

KHO, Y., AND GARLAND, M. 2005. Sketching mesh deformations. In *Symposium on Interactive 3D Graphics and Games 2005*, 147–154.

KOBBELT, L. P., BAREUTHER, T., AND SEIDEL, H.-P. 2000. Multiresolution shape deformations for meshes with dynamic vertex connectivity. *Computer Graphics Forum 19*, 3.

KOENDERINK, J. J. 1990. *Solid Shape*. MIT Press.

LIPSON, H., AND SHPITALNI, M. 1997. Conceptual design and analysis by sketching. In *AIEDAM-97*, vol. 14, 391–401.

MUMFORD, D. 1994. Elastica and computer vision. In *Algebraic Geometry and Its Applications*, C. L. Bajaj, Ed. Springer-Verlag New York Inc.

NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM SIGGRAPH Transactions on Graphics*, 1142–1147.

NITZBERG, M., MUMFORD, D., AND SHIOTA, T. 1993. *Filtering, Segmentation, and Depth*. Springer-Verlag.

PENTLAND, A., AND KUO, J. 1989. The artist at the interface. Tech. Rep. 114, MIT Media Lab.

PEREIRA, J. P., BRANCO, V. A., JORGE, J. A., SILVA, N. F., CARDOSO, T. D., AND FERREIRA, F. N. 2004. Cascading recognizers for ambiguous calligraphic interaction. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*.

SCHMIDT, R., WYVILL, B., SOUSA, M. C., AND JORGE, J. A. 2005. Shapeshop: Sketch-based solid modeling with blobtrees. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 53–62.

SHESH, A., AND CHEN, B. 2004. Smartpaper: An interactive and user-friendly sketching system. In *Eurographics Computer Graphics Forum*, vol. 23/3, 301–310.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, M. C. Lin and D. Manocha, Eds., vol. 1148. May.

SKETCHUP. SketchUp software: 3D sketching software for the conceptual phases of design. http://www.sketchup.com.

TAUBIN, G. 1995. A signal processing approach to fair surface design. *Computer Graphics 29*, Annual Conference Series, 351–358.

TAUBIN, G. 2001. Linear anisotropic mesh filtering. Tech. Rep. RC-22213, IBM Research.

TOLBA, O., DORSEY, J., AND MCMILLAN, L. 2001. A projective drawing system. In *2001 ACM Symposium on Interactive 3D Graphics*, 25–34.

ULUPINAR, F., AND NEVATIA, R. 1993. Perception of 3D surfaces from 2D contours. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15/1, 3–18.

ULUPINAR, F., AND NEVATIA, R. 1995. Shape from contour: Straight homogeneous generalized cylinders and constant section generalized cylinders. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17/2, 120–135.

WILLIAMS, L. R., AND JACOBS, D. W. 1997. Stochastic Completion Fields: A Neural Model of Illusory Contour Shape and Salience. In *Neural Computation*, vol. 9/4, 837–858.

WILLIAMS, L. R. 1994. *Perceptual Completion of Occluded Surfaces*. PhD thesis, University of Massachusetts.

WILLIAMS, L. R. 1997. Topological reconstruction of a smooth manifold-solid from its occluding contour. *Intl. Journal of Computer Vision 23*, 1, 93–108.

WITKIN, A. P. 1980. *Shape from contour*. PhD thesis, MIT.

ZELEZNIK, R. C., HERNDON, K., AND HUGHES, J. 1996. Sketch: An Interface for Sketching 3D Scenes. In *Proceedings of SIGGRAPH 96*, 163–170.