EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

Volume 21 (*2002*), *Number 3*

Free-form sketching with variational implicit surfaces

Olga Karpenko[◇], John F. Hughes[◇] and Ramesh Raskar[†]

[◇] Department of Computer Science, Brown University, Providence, RI, USA
[†] Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA
{koa, jfh}@cs.brown.edu, raskar@merl.com

Abstract

With the advent of sketch-based methods for shape construction, there's a new degree of power available in the rapid creation of approximate shapes. Sketch [Zeleznik, 1996] showed how a gesture-based modeler could be used to simplify conventional CSG-like shape creation. Teddy [Igarashi, 1999] extended this to more free-form models, getting much of its power from its "inflation" operation (which converted a simple closed curve in the plane into a 3D shape whose silhouette, from the current point of view, was that curve on the view plane) and from an elegant collection of gestures for attaching additional parts to a shape, cutting a shape, and deforming it.
But despite the powerful collection of tools in Teddy, the underlying polygonal representation of shapes intrudes on the results in many places. In this paper, we discuss our preliminary efforts at using variational implicit surfaces [Turk, 2000] as a representation in a free-form modeler. We also discuss the implementation of several operations within this context, and a collection of user-interaction elements that work well together to make modeling interesting hierarchies simple. These include "stroke inflation" via implicit functions, blob-merging, automatic hierarchy construction, and local surface modification via silhouette oversketching. We demonstrate our results by creating several models.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Modeling packages I.3.6 [Computer Graphics]: Interaction techniques

1. Introduction

We present a system that allows the creation of certain free-form objects with a sketching interface similar to Teddy [12] and Sketch [23]. Everywhere in this paper by "sketching" we mean using gestural marks to model shapes. In recent years, free-form sketching of 3D shapes has become feasible, as demonstrated by the work of Igarashi *et al.*[12] and Bourguignon *et al.*[7]. In Igarashi's Teddy system, the underlying surface representation is a polygonal mesh, while we present another way of modeling similar free-form shapes - using implicit functions[6]. This allows us to introduce new operations on such models easily. Furthermore, implicit models provide very nice smoothness characteristics.

In this paper, we take a particular representation for implicit surfaces — the *variational implicit surfaces* of Turk and O'Brien [20] — and show how it can be used to support a collection of free-form modeling operations. The principal contributions of the paper are

- operations for easy editing of free-form shapes, and
- a demonstration of how these operations are implemented in the context of variational implicit surfaces.

We describe the user-view of the operations in detail in section 3. The operations are *inflation*, *hierarchy generation*, *merging*, and *local modification*. The first operation is similar to inflation in the Teddy system; the others have some similarities to Teddy and some differences.

The implementation is described in section 5, after an introduction to the surface representation. Except for hierarchy generation, all operations are implemented in a similar way: by the removal of some constraints on an implicit surface and the introduction of others. This uniformity leads to simplicity of structure and coding.

2. Related Work

Several interfaces for sketching three-dimensional shapes have been developed for different classes of models.

For instance, Sketch[23] by Zeleznik *et al.* was designed to help a user with geometric modeling. It uses a system of intuitive guesses to guide a user through geometrical object creation. For example, if a user draws three segments meeting in one point, and parallel to the projections of the *x*-, *y*-, and *z*-axes of the scene, the system creates a cube whose dimensions are determined by the segment lengths. Once a user has mastered a set of agreements like this, it is easy to create complex models consisting of many primitives.

Lipson and Shpitalni[13] introduced a system for sketching CAD-like geometric objects. In an input sketch a user draws both visible and hidden contours of a rectilinear object and their system infers a shape. Their approach is based on correlations among arrangements of lines on the drawing and lines in space. For instance, by doing experiments, they can notice that "the more two lines are parallel in the sketch plane, the more they are likely to represent parallel lines in space"[13]. They rely on a geometric correlation hypothesis that claims that we could evaluate the likelihood of human understanding of the sketch by the joint probability of all correlations in the sketch.

Teddy[12] is an interface for free-form modeling. There, a user first inputs a simple closed stroke and the system creates a shape matching this contour. Then the user can add details by editing the mesh with operations like extrusion, cutting, bending, and drawing on the mesh. These let the user create fairly interesting models.

Tolba *et al.*[16],[19] 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 were created from it. This is done by projecting the 2D strokes on the sphere with the center at the eye point and then viewing them in perspective. This system's goal is different from ours: their system is a tool for perspective drawing and does not construct a 3D model.

Petrovic *et al.*[17] correlate features in a simple, textured, 3-D model with features in a hand-drawn figure, and then distort the model to conform to the hand-drawn artwork. The warp distorts the model in only two dimensions to match the artwork from a given camera perspective, yet preserves 3-D effects such as self-occlusion and foreshortening.

With respect to underlying representations, there's a long history of implicit-surface modeling, nicely summarized in a book by Bloomenthal [6]. Here we describe a few areas of research that are particularly relevant to this paper.

The "Skin" system[15], developed by Markosian *et al.*, supports a form of constructive drawing, in which the user creates a set of basic forms over which he places a skin, whose characteristics are then modified by small adjustments to offset distances at various scales of subdivision of the skin mesh. This skin is basically a polygonization of an implicit surface, but one that's defined by a combination of signed-distance representations of the underlying forms. No provision is made for directly creating the underlying blobs, aside from the operations described in Zeleznik's "Sketch" work, or for any blob hierarchy.

By contrast, Wyvill *et al.*[21] describe the blob-tree, a CSG-like hierarchy of implicit models in which shapes are combined by a rich collection of operators, including various deformation operators. Their emphasis is on the representation of complex implicit surfaces through a tree-like structure. We, too, build a tree-like structure with implicit surfaces at the leaves, but ours is a more conventional modeling-transformation hierarchy, in which the internal nodes represent grouping or linear transformations; when we merge surfaces or make small modifications to them, these edits are applied directly to the underlying implicit representation.

Our editing operations use a key idea described by Barthe *et al.*[2], a method for blending a pair of implicit surfaces defined as zero-sets of functions f and g by considering each point of R^3 as having f and g "coordinates," i.e., by treating the map

$$(f,g) : R^3 :\longrightarrow R^2 : p \mapsto (f(p), g(p))$$

as a kind of "coordinatization" of 3-space. The surfaces are then the preimages (i.e., points that map to) of the $f = 0$ axis and $g = 0$ axis respectively. By looking at the preimages of other curves in (f,g)-space, especially ones containing large portions of these two axes, they create blends between the two surfaces.

Finally, our surface representation is based on *variational implicit surfaces*, as described by Turk and O'Brien[20] (see section 4). Such variational implicit surfaces can be used to represent some very complex objects, as shown by Carr *et al.*[8]; their methods show how one can simplify the representation somewhat by judicious deletion of constraint points.

Another approach is to use the polygonal representation and direct mesh algorithms, but improve the smoothness of the meshes by using techniques like mesh subdivision and mesh fairing [14,18].

Our interaction techniques are derived from those in Sketch, Teddy, and the curve-oversketching idea presented by Baudel[4] for the case of 2D piecewise parametric curves.

3. Overview of Operations

Just as in children's sketching books[1], one is taught to first draw the general forms of things starting from simple pieces (cylinders, spheres, cones, blobs, ...), and then to draw a more careful outline and erase the underlying shapes, we provide the user the opportunity to roughly sketch out shapes and then modify them to provide a final form. We call the sketched shapes "blobs" throughout this paper.

A typical interaction session begins with a view of the world with nothing in it but a horizontal "ground plane" (which corresponds to the *x*- and *z*-directions in our coordinates, with the *y*-direction pointing upwards). The user may

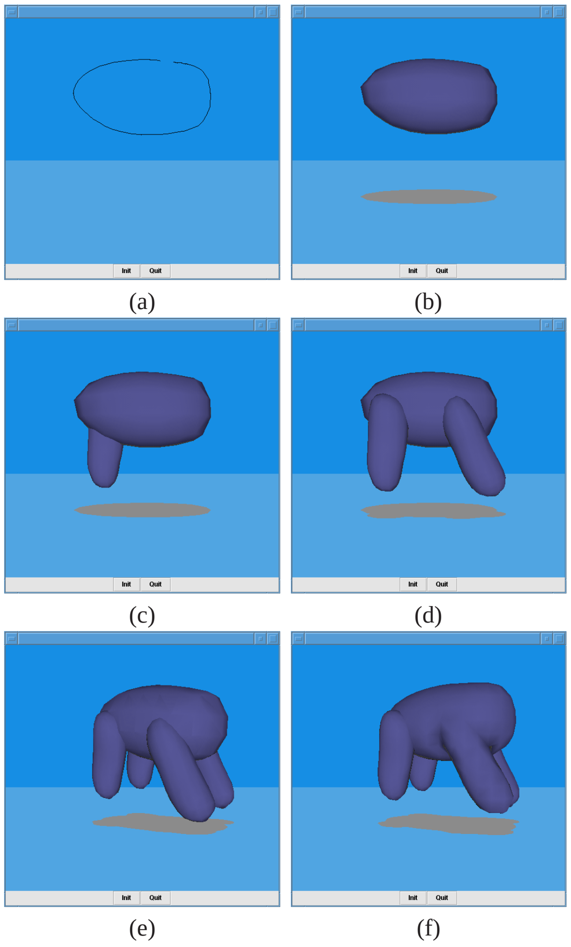

Figure 1: *(a) A single closed stroke above the ground-plane, which (b) is "inflated" to become the body. (c) Another stroke overlapping the body generates a leg attached to the body. (d) Another leg has been added, and the two near-side legs, their placement having been adjusted slightly, are duplicated with a "symmetrize" operation. (e) A different view. (f) The foreleg is merged with the body.*

draw an outline of a blob depicting the body of an animal, for example, as shown in figure 1(a). This outline is "inflated" into a 3-dimensional shape whose thickness is related to the dimensions of the 2D outline.

The user can then draw further blobs indicating the legs of the animal, as in figure 1(c). These are again drawn at the same depth as the first blob, but the depth may be adjusted as described below. Because these overlap the first blob, the system infers that they are to be attached, and places the new blob in a modeling hierarchy with the first blob as parent; if the parent is later rotated or translated (through a Unicam-like 3rd-button mouse interface[22]) the child is moved as well. As the parent is determined, it briefly flashes pink to tell the user what inference has been made. If a newly-drawn blob overlaps multiple others, one of these is chosen as its parent (based on degree of overlap in the image plane).

If the position of the newly-created blob is not ideal, the user may select the blob (or any other blob) by left-clicking it once; at this point, a transparent bounding sphere appears, ready for use as a "virtual sphere rotater," and the color of the shadow of the selected blob is changed to make it easier to select. The user then may

- translate the blob and its children in the *xy*-plane,
- translate the blob and its children in the *xz*-plane by dragging the "shadow" of the object, as described by Herndon *et al.*[11], or
- rotate the object around either its center of mass or the center of mass of its intersection with the parent (implemented only for ellipsoids).

The choice of operation is based on where the user clicks when starting to drag the mouse cursor: a click on the bounding sphere indicates trackball rotation; a click on the "shadow" of the blob or near it (which we define as "inside the shadow of the bounding sphere of the blob") moves the object in the *xz*-plane; otherwise the object moves in the *xy*-plane. The amount of *xy* translation is determined by the vector from the first-click to the current mouse-point.

Having drawn the legs, the user may "symmetrize" the model, relative to some parent node, i.e., may create a symmetric copy of all children relative to the plane of symmetry of the parent, by pressing the "S" key, as in figure 1(d, e). The user can add further blobs to the model (e.g., a tail, a head, a nose, ...) and adjust their positions and orientations. When the approximate form is correct, the user may begin to *merge* blobs in one of two ways. In the first, the user selects two blobs and the two are merged together into a single larger blob with a fillet at the junction between them, as in figure 1(f), where foreleg and body have been merged.

In the second form, the user draws a "guidance stroke" starting on the silhouette of one blob and ending on the silhouette of the other, and the fillet is placed so as to match (approximately) this guidance stroke, as shown in figure 2. In this figure, all blobs are transparent; the original blobs that are merged are gray, the guidance strokes are red, and the resulting surface is pink. Notice that this operation successfully merges even non-intersecting blobs, as long as they are reasonably close to each other. This can be used, for example, to join the head and the body of an animal.

In our current implementation, the two merged blobs are separated from the modeling hierarchy entirely, and the new merged blob is inserted to the hierarchy at the same level as the first blob; all the children of the second blob are also attached to it. If the first blob had no parent, then before a new merged blob is inserted in the hierarchy, the system would try to find a parent for it. While this is asymmetric, it seems reasonable compared to placing this merged blob into the hierarchy as an independent blob.

Guidance strokes have another use: the user may draw a stroke starting on the silhouette of an object, briefly leaving that silhouette, and then returning to it; points on the surface near the stroke are displaced to lie on the newly-drawn guidance curve. The user can thus make small deformations (giving a camel a hump, for instance, or putting a dent in a cushion) directly by sketching. This operation, called "oversketching", is demonstrated in figure 3.

Some examples of the animal hierarchies we can build with the program are shown in figure 4.

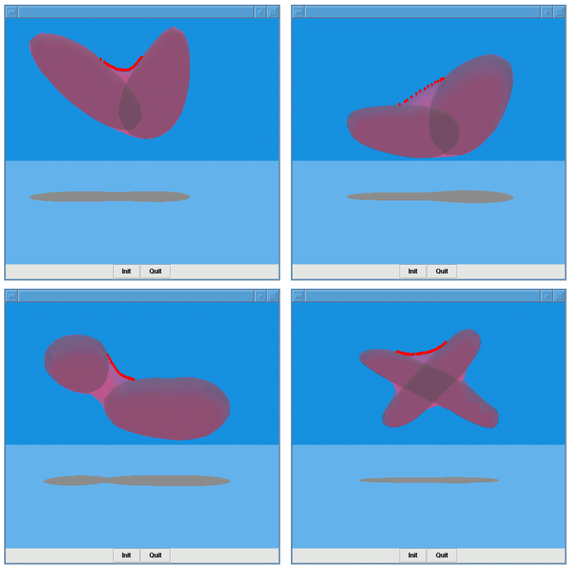

Figure 2: *Merging with guidance strokes. In each case, the guidance stroke constraint points are shown as red dots, and the new surface is shown in pink.*

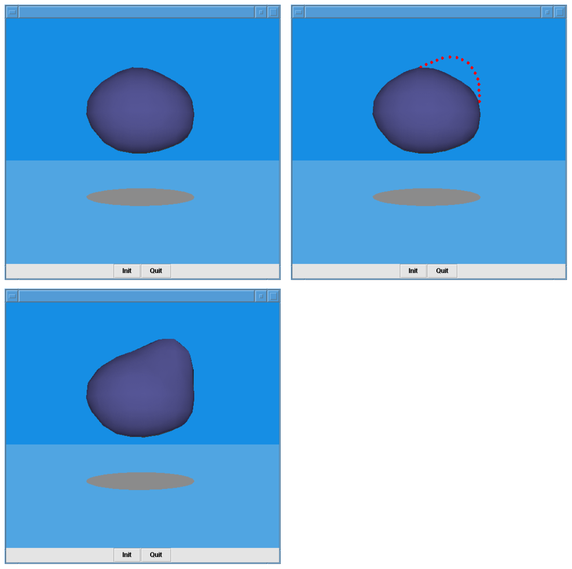

Figure 3: *Oversketching: The upper left blob is modified by the user's drawing a stroke near the silhouette (upper right). The surface deforms to match the stroke (lower left).*

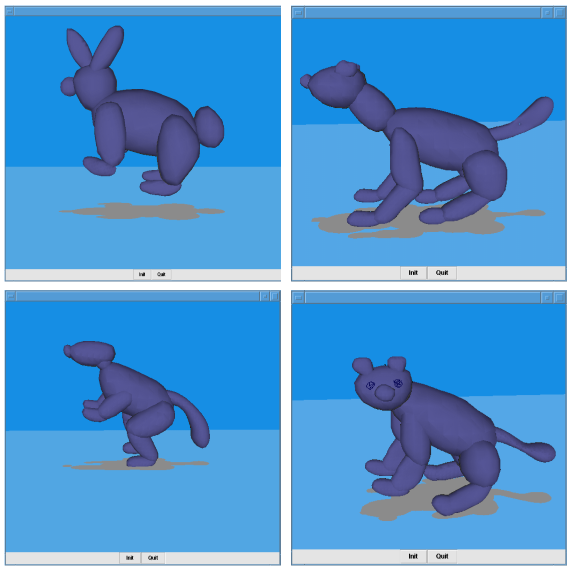

Figure 4: *Examples of hierarchies created automatically.*

4. Surface representation

Variational implicit functions are based on thin-plate interpolation[20], which is widely used in solving scattered data interpolation problems where one needs a smooth function, minimizing squared second-derivatives, that passes through a given set of data points. Such a function (a *thin plate* function) is known to be a sum of an affine term and a weighted sum of 'radial basis functions,' i.e., functions of the form

$$f_i(p) = \phi(\|p - q_i\|)$$

which depend solely on the distance from the argument to the *i*th data point q_i. The exact form of ϕ depends on the dimension; for functions on R^3, the form is $\phi(x) = x^3$.

Thus, any thin plate function can be expressed as

$$f(x) = \sum_{j=1}^{n} d_j \phi(x - c_j) + P(x)$$

where the d_js are real-number coefficients, the c_js are "centers" for the basis functions, and

$$P(x) = a \cdot x + b$$

is an affine function of position x.

Given a collection of n locations at which f is to take on particular values, one can choose the c_js to be the first $n-4$ given locations, and can then solve for the d_js and a and b; this is simply a large linear system.

Thus, the specification of a variational implicit function requires the specification of n values at n points; we refer to such specifications as *constraints*. Turk and O'Brien consider two kinds of constraints: "zero points" and "plus points." A zero point is a point c_j at which $f(c_j) = 0$. This means that the implicit surface $S = \{x | f(x) = 0\}$ passes through such a point.

A "plus point" is one for which $f(x) = 1$ must be satisfied. We will consider points for which $f(x) < 0$ to be "inside," so these "plus points" determine locations which must be outside the implicit surface we are defining.

Turk and O'Brien observed that if one had a set of points through which one wanted a surface in R^3 to pass, one could build a function $f : R^3 \longrightarrow R$ by placing zero-points at the given points, and plus-points at the endpoints of the desired normals. The zero-level set of this function is then a surface interpolating the points and with approximately the given normals.

It's also worth noting that if the function

$$f(x) = \sum_{j=1}^{n} d_j \phi(x - c_j) + a \cdot x + b$$

interpolates the points c_j with values e_j, then the function

$$\begin{aligned} f(x-k) &= \sum_{j=1}^{n} d_j \phi(x - k - c_j) + a \cdot (x-k) + b \\ &= \sum_{j=1}^{n} d_j \phi(x - (c_j + k)) + a \cdot x + (b - a \cdot k) \end{aligned}$$

interpolates the points $c_j + k$ with the same values e_j, i.e., that translating the "interpolation centers" induces no change on the radial-basis-function coefficients, and a simple change in just the constant of the affine term. Similarly, if one multiplies all the control points by some fixed rotation matrix R, i.e., replaces c_j with $R(c_j)$, then again the coefficients d_j remain the same, and only the coefficient a changes, being replaced by $a' = Ra$.

Finally, note that if we have a real-valued function and build a mesh-approximation of its zero-level surface, then we can take the points of that mesh as zero points for a new variational implicit function, and the endpoints of mesh normals as plus-points, and thus create a new real-valued function whose zero-level surface will closely resemble the mesh. This idea, which comes from Turk and O'Brien is critical in our merging and small-modification algorithms.

5. Implementation details

In this section, we will describe in more detail each of the operations previously mentioned, and then how each is implemented in the framework of a variational implicit surface. We begin with some basic facts about our implementation, and then move on to inflation and the other operations.

Our system's coordinates are such that the initially-visible section of the world, at middle-depth, is about 6 units wide; our view of the world is through a 512 pixel wide window. By "middle-depth," we mean "the depth at which strokes are placed into the world," namely, on the $z = 0$ plane. This choice of size, although arbitrary, is necessary to clarify the other dimensions mentioned below.

5.1. Inflation

Inflation is the process of converting a user stroke to a 3D blob, represented as a variational implicit surface, whose silhouette matches the given stroke.

We need to go from a 2D visible-contour drawing to a 3D shape. We follow a sequence of operations, namely:

1. Collect the user-input, i.e. a *stroke*
2. Re-sample the stroke;
3. Assign depths (i.e., distances from the eye) to the points of the stroke; we call the resulting path in 3D the *contour*.
4. Create a surface model consistent with the locations, including depth, of the points of the contour, represented as a variational implicit surface.

5.1.1. Preprocessing an input stroke

User input is gathered from the mouse as a collection of screen-space points. The user begins a stroke by left-clicking, and then drags over the path of the desired stroke, and finishes by releasing the mouse button. During the input, the 2D points arrive at a rate that we cannot control. We re-sample these points so that they are not bunched up too closely. We use Igarashi's[12] algorithm: when there are too many points close together, we simply ignore some samples. If the distance between the previous point and the current point is less than 15 pixels, we do not add the current point. This rather crude re-sampling seems to have no real impact on the results.

To inflate such an input stroke into a blob, we need to explicitly specify the "constraints" ("plus points" and "zero points") that determine a variational implicit function. We first project the 2D points of this stroke onto the $z = 0$ plane; the resulting points are used as zero-points for the implicit function we want to create; we'll refer to the path defined by these zero-points as the *contour*. (If the view of the world has been rotated, we put the points on a plane through the origin, orthogonal to the current view direction, but it's easier

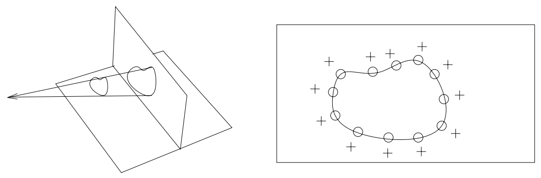

Figure 5: *Inflating a stroke: (a) the user's stroke is resampled and projected onto the $z = 0$ plane. (b) The points defining the stroke are used as zero-points for inflation (indicated by small circles), and points slightly offset along the normals are used as plus-points (indicated by plus-signs).*

to express the remainder of the construction in an un-rotated frame of reference.) Still within the $z = 0$ plane, we compute points slightly displaced (distance 0.05) from the zero-points along the normals to the contour, and make these all plus-points (see figure 5). To place additional constraints in space (having all constraints in a plane leads to a degenerate situation) we tried three approaches:

5.1.2. Placing the constraints

1. Two plus constraints in 3D are placed at the center of mass of the contour, and then moved off the $z = 0$ plane to depths $z = depthConst$ and $z = -depthConst$, where we used $depthConstant = 1.5$. This makes the "thickness" of the object a constant, which makes a leg look almost as "fat" as a body.
2. As opposed to having the same thickness for all objects, we make it depend on the shape of the user's stroke. We wanted, for example, that long cigar-shaped contours become blobs with circular cross-section whose radius is the "width" of the cigar, while spherical-looking blobs should be fairly "fat". We therefore use a simple measure for "width" of the stroke shapes:

   Given the 2D shape as a contour consisting of points, we first find the two points that are furthest from each other, and call this the *axis* of the shape. Then we find the center of this axis, draw a perpendicular through this point and find the point closest to the center along this perpendicular. This closest distance is our measure for the "width" of the 2D shape drawn by a user.

   We now place the two additional "plus points" as before, but instead displace them not by $z = \pm 1.5$, but rather by $z = \pm 1.5 \cdot width$.
3. To ensure that the surface curves "in" (i.e., away from the outward normal at the $z = 0$ slice) as one moves away from the $z = 0$ plane, we take two copies of the stroke points and translate one of them in the positive z direction, the other in the negative z direction, and put plus-points at the resulting locations. The distance we move each of these copies is calculated as in our second method above – but this time using $0.8 \cdot width$.

In all the examples in the paper we have used the second inflation technique, although there are some cases where it is not ideal, such as highly convex curves. Finding a more principled approach to inflation is critical for future development.

Having determined the set of all zero-points and plus-points, we use the method described in section 4 to compute a variational implicit surface. We then use Bloomenthal's[5] polygonizer to create a 3D mesh corresponding to this surface, which is what we actually display. We set the size of the grid-cells in the polygonizer to be $0.5 \cdot width$, so that even thin objects get polygonized reasonably.

5.2. Hierarchy

Our current implementation is in Java3D, so we organize all the blobs in a hierarchy that's stored in the Java3D scenegraph. Our scenegraph is a tree consisting of branch nodes, each of which contains a blob, with a modeling hierarchy above them. Each branch node has an associated transformation and a shape to render. In our case, the shape for each blob node is the mesh built by polygonizing the implicit surface of the blob. When rendering each blob, Java3D automatically parses the scenegraph transforming each leaf node with the transformation accumulated by multiplying all the transformations from the nodes that are in the path to this shape. All blob nodes have the structure shown in Figure 6. Initially, each blob is placed according to the position of the 2D input stroke and at some fixed depth. Next, the system attempts to find a parent of this blob based on the intersection with the other blobs. The algorithm for finding a parent for a blob is given below. If a blob does not have "big enough" overlap with any of the previous blobs, it has no parent and is placed on the high level of the hierarchy as an independent blob. If the system finds the parent, then the transformation of the child is updated as follows:

- Find the "accumulated transformation" of the parent - a multiplication of all the transformations in the nodes above the parent and the transformation of the parent node itself.
- Multiply the blob's transformation from left with the inverse of the parent's "accumulated transformation". Place the result into the transform node of this blob.

Operations such as rotation around the center of mass of the blob and translation either in *xy*-plane or *xz*-plane update the transformations of the corresponding blob in the scenegraph. Children are updated automatically because of the way the blobs are placed in the hierarchy.

The algorithm for finding the parent of the blob is the following:

1. Render this blob to an off-screen buffer and save it in an image; find the 2D bounding box of the blob's projection.
2. For each blob previously placed in the hierarchy, render it to the off-screen buffer and find the overlap with the new blob by comparing pixels of the two images. Note that we only need to compare pixels in the sub-area of the image corresponding to the 2D bounding box found in step 1.
3. Calculate the ratio of overlap with this blob by dividing the number of points found in step 2 by the area of the current blob.
4. Among all blobs for which the ratio is more than a certain threshold, choose the one with maximum overlap. Choose this as the parent.

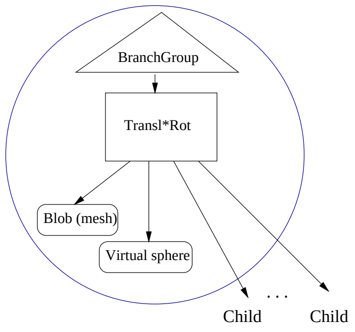

Figure 6: *The underlying branch graph for a blob*

For the "symmetrize" operation, for each child of the parent, we find its "symmetrical" copy as follows:

- First we reflect/flip the vertices and normals of every mesh stored in the branch subgraph of a child around the center of the parent and with respect to the normal of the parent; Then we change the transformations in each transformation node in this subgraph as discussed in the second step.
- Let's say the child has the transformation matrix T relative to the parent, we extract the translational Tr and rotational Rot components from this matrix. We define the operation "flip" applied to the vector v as follows:

  $$Flip(v) = v - 2 * (v \cdot n)n$$

  where the n is the normal to the plane of symmetry of the parent. Then we create a new transformation whose translation component is $Flip(v)$ and whose rotational component is determined as follows: if the rotation matrix is the rotation around the axis a with angle ϕ, then the new rotation matrix is the rotation around the axis $Flip(a)$ with the angle $-\phi$.

This set of operations on the hierarchy, although simple, allows us to create a variety of interesting shapes as seen previously.

5.3. Merging

Blending operations to allow smooth transition between two implicit surfaces defined by $f : R^3 \longrightarrow R$ and $g : R^3 \longrightarrow R$ have been proposed by many researchers; we follow the general idea laid out by Barthe et al.[3]. Here is the central idea:

Define a function

$$H : R^3 \longrightarrow R^2 : P \mapsto (f(P), g(P)).$$

Clearly H sends all points on the level-zero isosurface of f (which we'll call the "isosurface" from now on, the "level zero" being understood) to points on the y-axis of R^2; similarly it sends those on the isosurface of g to points on the x-axis. Consider a point Q on the positive x-axis. A point in 3-space that maps to Q is evidently in the isosurface for g, but *outside* the isosurface for f. Similarly, anything that maps to a point on the positive y-axis is on the isosurface for f, but outside the isosurface for g. Letting L denote the union of the positive x- and y-axes, we see that the preimage of L, i.e., $H^{-1}(L)$, is simply the union of parts of the two isosurfaces that are outside each other (see figure 7).

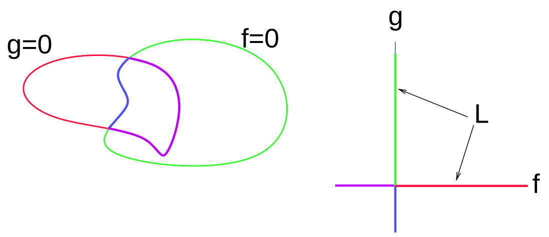

Figure 7: *Isosurface blending in 2D. The level set for f and the level set for g intersect. When points of these level sets are mapped to R^2 by H, they land at corresponding points in the right-hand figure. One can see that the preimage of the positive axes consists of a merge of the "outside" parts of the two level sets.*

If we have a function, $G : R^2 \longrightarrow R$ whose level-zero level set is exactly L, the union of the two positive axes, then $G \circ H : R^3 \longrightarrow R$ is a function on R^3 whose level set is the union of the outside parts of the two level surfaces. If, on the other hand, we have a function G whose level-zero level set is an approximation of L, then the level-surface of $G \circ H$ will be very similar to the union of the two outside parts; in particular, if the level set for G deviates from the axes only near the origin, then the level set of $G \circ H$ will deviate only near the intersection of the level sets of f and g.

5.3.1. Automatic Merging

Let's say we want to merge two blobs with implicit functions f and g. We implement the idea just described in a particularly simple way. First, each blob is sampled at a fixed resolution and polygonized to create vertices and corresponding normals, which are used as zero-points and plus-points for a new implicit representation of the blob.

We now create a new implicit function from a subset of these vertices and normals. The vertices (and corresponding normals) to be eliminated are determined by the restriction $(G \circ H)(P) < 0$; in other words, we eliminate all vertices v (and their corresponding normal points) that lie inside the intersection of the two blobs, i.e.,

$\forall v \in$ first blob, eliminate v if $g(v) < 0$.

$\forall v \in$ second blob, we eliminate v if $f(v) < 0$.

The implicit surface reconstructed from the remaining vertices (and corresponding normals) is a smooth merge between the two surfaces (see figure 8). The same idea can be used for merging three or more blobs.

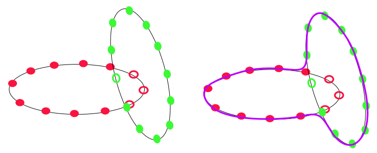

Figure 8: *Automatic merging: the hollow constraint points on the left are eliminated, leading to the new iso-set in purple on the right.*

5.3.2. Merging with a guidance stroke (with local influence)

The user can specify the shape of the merged blob in one of the cross-sectional planes along the intersection. In this case, the input consists of two blobs and a 2D guidance stroke (Figure 9). In addition, the user may specify a limit on the region of influence of the guidance stroke specified by the 3D distance σ (set to the constant value 0.8 in our implementation). The influence of the guidance stroke decreases with distance from the stroke, until at distance σ there is no influence at all. Beyond the region of influence on the two implicit surfaces, the resultant merged surface is same as the one produced by automatic merging. Below we describe the process of computing the blending function G for this guided version of merging.

We first find the 3D location of the start and end point of the guidance stroke, S and E, by finding the nearest silhouette point on the appropriate blob. We then project the 2D guidance stroke onto the plane passing through S and E and parallel to the average normal at those two points. Without loss of generality, let's say S lies on the first blob, defined by the function f and E lies on the second, defined by the function g. The "extrusion space coordinates" for the point S are then are $(f(S), g(S)) = (0, s)$; for E they are $(f(E), g(E)) = (e, 0)$. We sample the guidance stroke and plot the corresponding points in the extrusion space by computing implicit function values (f, g). To achieve a smooth transition with limited region of influence, we add a third dimension to the extrusion space. The third coordinate $d(x)$ for a 3D point x is the minimum of the Euclidean distance to the points S or E. In the extrusion space, the surface generated by our guidance stroke in the f-g plane and the point $(0,0,\sigma)$ defines the blending surface on which $G(f1, f2, d) = 0$. The shape of this surface naturally determines the shape of the new surface in the blended region. (Note, however, we do not explicitly construct this surface in the extrusion space.)

For a practical implementation, we eliminate all vertices in the region of influence (and the plus-points corresponding to their normals) and add new zero points corresponding to the sampled locations on the 3D guidance stroke. For example, for vertices on the first blob, we eliminate all v for which $(g(v) < \delta_2)$ or $(f(v), g(v), d(v))$ is inside the triangle with vertices $(0, s, 0)$, $(0, 0, \sigma)$ and $(0, 0, 0)$. Similarly for the second blob: we eliminate all vertices v for which $(f(v) < \delta_1)$ or $(f(v), g(v), d(v))$ is inside the triangle with vertices $(e, 0, 0)$, $(0, 0, \sigma)$ and $(0, 0, 0)$.

For each new zero-point added from the projected and sampled guidance stroke, we add a corresponding plus-point by using the normal to the guidance curve in the plane of projection as the surface normal, and putting a plus-point a small distance (0.05) out along this normal. Once the appropriate constraints on the original blobs have been eliminated, and the new ones corresponding to the guidance stroke have been added, a new merged surface is generated.

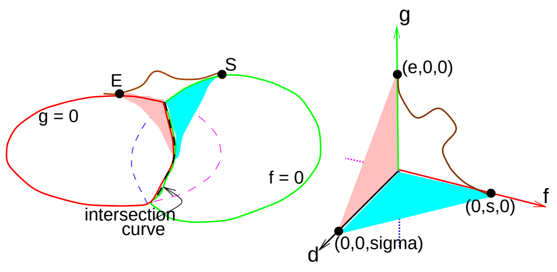

Figure 9: *Merging with a guidance stroke: The region of influence is shown in gray and light gray colors.*

5.4. Small modifications on the blobs

The user can make local modification of the profile of a blob simply by drawing a new target profile. The shape near to the target profile stroke is modified with the influence limited by 3D distance from the stroke. There are many 3D techniques to make local distortions, but most of them work on polygonal meshes. They involve cumbersome process of finding displacement vectors in the region of influence that vary smoothly across the region. Some 2.5D techniques deform the polygonal meshes so that they conform to the target profile stroke from the given viewpoint, but may have artifacts when seen from a different viewpoint[9].

We instead use an idea similar to the one used for merging with guidance stroke above. The input is a blob and a target profile stroke is drawn near its silhouette. In addition, there is a limit on the region of influence specified by 3D distance τ (again, set to 0.8 in our implementation). The main idea is to eliminate zero points (and corresponding plus points) in the region of influence and add new zero (and plus) points from sampled and projected 3D target profile stroke. Note, that the user draws only a target stroke and does not draw a source stroke, as required earlier work [12],[9]. The source silhouette is automatically computed for the viewpoint in which the target profile was specified. This is an important improvement. It provides simplicity and eliminates the often seen error in clicking on or specifying the source curve.

By providing only new constraints (and relying on the mechanism of variational implicit surfaces), we also avoid the problem of enforcing smooth variation in the displacement vector along the modified surface.

To create the new implicit surface constraints, we first find the nearest silhouette point for start and end of the target (profile) stroke, and assign 3D coordinates, S and E as before. The target stroke is assumed to be snapped to these points if the stroke is within a few pixels. Next, we project the target stroke on the plane passing through S and E and parallel to the average normal at those two points, creating a planar 3D curve. For each point on the target curve, we find the nearest silhouette point on the blob. This creates the corresponding source stroke.

We find the region of influence, by comparing 3D Euclidean distance of each vertex on the blob with respect to the source stroke.

We assign a normal to each point on the sampled target stroke in the direction normal to the stroke curve and parallel to image plane. This provides new zero points and corresponding plus point constraints. The vertices in the region of influence are eliminated. The reconstructed implicit surface has the matching target profile because the surface interpolates the zero points on the target stroke and respects the specified normals parallel to the image plane.

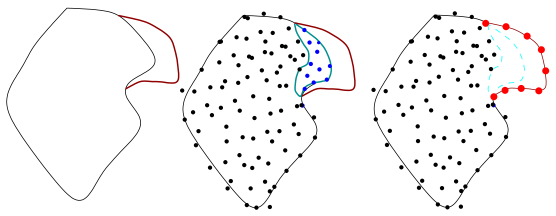
Figure 10: *Left: Blob and modification stroke. Middle: Blue dots indicate zero points in the green region of influence. Right: The blue points are eliminated and the new red zero points from the target stroke are added.*

6. Results, Limitations, and Future Work

We have described our system for creating free-form models from gestural input, using variational implicit surfaces in a modeling hierarchy as a surface representation. Our choice of modeling operations allows a user to construct shapes in a way that closely parallels the way that they draw, starting from basic forms and then refining.

The advantage of the implicit surface representation is that the natural modeling operations — inflation, merging, stroke-based merging — are all easy to implement in this context. Nonetheless, there are some limitations in our approach. As the number of constraints increases, the time it takes to compute the coefficients for the variational implicit surfaces grows as well. Presumably it's possible to reduce the number of constraints substantially, using methods like that of Carr *et al.*[8], and we hope to do so in the future.

Because the ratio of an overall size of an object to the cube size in the polygonizer is fixed, it is impossible to represent objects smaller than a certain size. In particular, when a small object is merged with a large one the details of the small one may disappear.

Our representation cannot support sharp edges in a surface; to do so would require a great many constraints. Presumably this can be remedied by some sort of hybrid polygonal/implicit representation or by using anisotropic basis functions[10].

Because our operations depend on implicit function values, and because these values, far from our surfaces, may have no intuitive meaning, our operations may produce unexpected results unless the guidance strokes (for example) stay close to the underlying surface. Furthermore, the region of influence of an operation depends on the gradient(s) of the implicit function(s) involved, and hence may be unpredictable.

There are also some non-critical limitations in our current implementation: at present we cannot handle the reparenting of blobs, although we expect this limitation to be easy to address. Our shadow-selection algorithm is based on the shadow of the bounding sphere of the blob rather than the blob itself. Merging is too slow for truly convenient interactive use. And we would like to improve our overlap-detection algorithm to make it faster.

6.1. Future work

We'd like to extend our merging algorithm to take into account the intersection curve between the two blobs: it would be nice to use a single guidance stroke to make a "reasonable" blend or fillet at all points of this curve.

We'd also like to extend the hierarchy-detection algorithm to include better placement of child blobs (sticking legs to the sides of bodies, for instance, rather than placing them along the body's medial plane and requiring that the user translate them), and a better use of the inferred hierarchy in determining inflation-depths for child blobs.

Finally, we want to add a “painting” component to allow the user to decorate the resulting shapes, drawing feathers on birds’ wings, for example, or spots on a leopard.

7. Acknowledgments

We thank Takeo Igarashi, Thomas Crulli and Tomer Moscovich for their suggestions and helpful discussions. This work was partly supported by the NSF Graphics and Visualization Center (NSF grant EIA 8920219).

References

1. AMES, L. J., Ed. *Draw 50 Animals*. Main Street Books, 1985. 2
2. BARTHE, L., GAILDRAT, V., AND CAUBET, R. Implicit extrusion fields. In *The 2000 International Conference on Imaging Science, Systems, and Technology (CISST’2000)*, CSREA press, pp. 75–81. 2
3. BARTHE, L., GAILDRAT, V., AND CAUBET, R. Extrusion of 1D implicit profiles: Theory and first application. In *International Journal of Shape Modeling* (December 2001), vol. 7/2. 7
4. BAUDEL, T. A mark-based interaction paradigm for free-hand drawing. In *ACM Symposium on User Interface Software and Technology* (1994), pp. 185–192. 2
5. BLOOMENTHAL, J. An implicit surface polygonizer. In *Graphics Gems IV*, P. Heckbert, Ed. Academic Press, New York, 1994. 6
6. BLOOMENTHAL, J., Ed. *Introduction to Implicit Surfaces*. Morgan-Kaufmann, 1997. 1, 2
7. BOURGUIGNON, D., CANI, M.-P., AND DRETTAKIS, G. Drawing for illustration and annotation in 3D. *Computer Graphics Forum 20*, 3 (2001), 114–122. 1
8. CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of SIGGRAPH 2001* (August 2001), pp. 67–76. 2, 9
9. CORRJA, W. T., JENSEN, R. J., THAYER, C. E., AND FINKELSTEIN, A. Texture mapping for cel animation. In *SIGGRAPH’98* (August 1998), pp. 435–446. 8, 9
10. DINH, H. Q., TURK, G., AND SLABAUGH, G. Reconstructing surfaces using anisotropic basis functions. In *Proc. ICCV* (Vancouver, Canada, July 2001), vol. 2, pp. 606–613. 9
11. HERNDON, K. P., ZELEZNIK, R. C., ROBBINS, D. C., CONNER, D. B., SNIBBE, S. S., AND VAN DAM, A. Interactive shadows. In *ACM Symposium on User Interface Software and Technology* (1992), pp. 1–6. 3
12. IGARASHI, T., MATSUOKA, S., AND TANAKA, H. Teddy: A sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH 99* (August 1999), pp. 409–416. 1, 2, 5, 9
13. LIPSON, H., AND SHPITALNI, M. Conceptual design and analysis by sketching. In *AIDAM-97* (1997). 2
14. MAILLOT, J., AND STAM, J. A unified subdivision scheme for polygonal modeling. In *Proc. Eurographics* (2001), vol. 20/3. 2
15. MARKOSIAN, L., COHEN, J. M., CRULLI, T., AND HUGHES, J. F. Skin: A constructive approach to modeling free-form shapes. In *Proceedings of SIGGRAPH 99* (August 1999), pp. 393–400. 2
16. OSAMA TOLBA, JULIE DORSEY, L. M. Sketching with projective 2D strokes. In *ACM Symposium on User Interface Software and Technology* (1999), pp. 149–157. 2
17. PETROVIC, L., FUJITO, B., WILLIAMS, L., AND FINKELSTEIN, A. Shadows for cel animation. In *Proceedings of SIGGRAPH 2000* (July 2000), pp. 511–516. 2
18. SCHNEIDER, R., AND KOBBELT, L. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design 18*, 4 (May 2001), 359–379. 2
19. TOLBA, O., DORSEY, J., AND MCMILLAN, L. A projective drawing system. In *2001 ACM Symposium on Interactive 3D Graphics* (March 2001), pp. 25–34. 2
20. TURK, G., AND O’BRIEN, J. Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH 99* (August 1999), pp. 335–342. 1, 2, 4
21. WYVILL, B., GUY, A., AND GALIN, E. Extending the CSG tree - warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (June 1999), 149–158. 2
22. ZELEZNIK, R. C., AND FORSBERG, A. Unicam — 2D gestural camera controls for 3D environments. In *1999 ACM Symposium on Interactive 3D Graphics* (April 1999), ACM SIGGRAPH, pp. 169–174. 3
23. ZELEZNIK, R. C., HERNDON, K., AND HUGHES, J. Sketch: An Interface for Sketching 3D Scenes. In *Proceedings of SIGGRAPH 96* (August 1996), pp. 163–170. 1, 2