Skin: A Constructive Approach to Modeling Free-form Shapes

Lee Markosian *Jonathan M. Cohen* *Thomas Crulli* *John Hughes*[*]

Brown University site of the NSF and Technology Center for
Computer Graphics and Scientific Visualization,
Box 1910, Providence, RI 02912

Abstract

We present a new particle-based surface representation with which a user can interactively sculpt free-form surfaces. The particles maintain mesh connectivity and operate under rules that lead them to form triangulations with properties that make them suitable for use in subdivision. A user interactively guides the particles, which we call *skin*, to grow over a given collection of polyhedral elements (or *skeletons*), yielding a smooth surface (through subdivision) that approximates the underlying skeletal shapes. Skin resembles blobby modeling in the constructive approach to modeling it supports, but allows a richer vocabulary of skeleton shapes, supports sharp creases where desired, and provides a convenient mechanism for adding multiresolution surface detail.

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling I.3.6 [Computer Graphics]: Methodology and Techniques

Additional Key Words: Free-form modeling, meshes, subdivision, multiresolution

1 Introduction

The algorithm described in this paper derives from our ongoing effort to build a free-form modeling system with a direct interface – that is, one in which the user defines shapes by sketching them directly as they would appear from a given viewpoint. In particular, we seek to extend the principles developed in the SKETCH system [26] – rapid construction of approximate shapes via direct interaction – to the problem of free-form modeling. The Teddy system presented in these proceedings [11] targets similar goals, though we seek to provide more fine-grained control over the resulting shape, including the ability to add surface detail at multiple scales. An important goal is that the interface should allow a user with skill at drawing on paper to apply this skill to model compelling 3D forms.

With that in mind, we seek to build a system in which the user models complex surfaces by first constructing simplified representations of the underlying masses that give them their form, then "oversketches" these masses with a smooth surface that approximates their collective shape. We chose this approach because of its tractability (SKETCH and Teddy, as well as our system for sketching 3D curves [6], describe techniques that could be applied to define the "underlying masses"), and because it is natural – that is, it resembles techniques commonly used by artists and recommended in books on art instruction [1, 5, 7, 8, 16, 18, 22].

This paper describes our solution to a sub-problem that arises in implementing our approach: given a collection of "masses" (or *skeletons*) represented as polygon meshes, each with an associated offset (typically small), construct a smooth surface that approximates the distance surface defined by the skeletons and their offsets. Distance surfaces, discussed in [3] and defined in section 3 below, typically have sharp creases that are not desired. Thus the final surface should smoothly approximate the distance surface, not fit it exactly. The solution should work in an interactive setting, continually updating the surface as skeletons are added, manipulated, or removed.

Our solution, the *skin* algorithm, works by constructing a subdivision surface control mesh that roughly fits the shape of the distance surface. The control mesh in turn defines a smooth surface with a similar shape. A key contribution of this paper is to provide an algorithm for generating a *suitable* control mesh for use in subdivision. Simply matching a given shape is not sufficient – the triangulation of the control mesh has a strong influence on the qualities of the resulting surface. Smaller triangles provide less "smoothing" effect, yielding a surface that more closely fits the control mesh, and – more importantly – irregular triangulations can greatly reduce the fairness of the limit surface. This can be seen in figure 1.

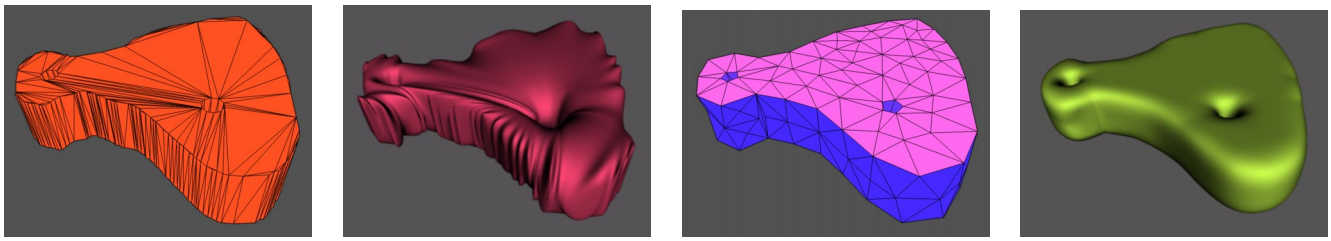
The skin algorithm is good for *rapid* construction of *approximate* free-form shapes. It is not suitable for applications that require exact control of the final shape. Skin supports fast, intuitive editing of features at multiple resolutions and produces piecewise smooth surfaces with creases or sharp corners where desired. Since the algorithm manipulates a polygon mesh, rendering can be done efficiently on common graphics hardware.

Together with these advantages, the skin algorithm has some drawbacks: it is based on an iterative algorithm that is not guaranteed to converge to a fixed triangulation – and sometimes doesn't. The exact results depend not just on the primitives used to define the surface, but also on the order of the user's operations. Finally, the behavior of the skin depends on the size of its triangles (which the user can control). It would be preferable not to expose the user to the issue of triangle size as a means of controlling surface shape. The other difficulties can be alleviated by providing tools to let the user control the skin's behavior, as we will illustrate. Because we designed skin for use in an interactive setting, we do not consider any of these problems to be severe.

2 Related work

The skin algorithm resembles the methods for iteratively evolving a triangulation described in [9, 10, 20, 23, 24]. It has much in common with Miller *et al.*'s "volumetrically deformed models" [14] in which a balloon-like surface is inflated under constraints to match a volumetrically defined implicit function. Also, because the skin al-

[*] {lem,jmc,tc,jfh}@cs.brown.edu



(a) (b) (c) (d)

Figure 1 Triangulation matters. Two triangulations of the same shape ((a) and (c)) produce different results when subdivided ((b) and (d)). The skin algorithm produces even triangulations, avoiding skinny triangles like those in (a) that lead to unexpected wiggles and bumps when subdivided.

gorithm treats vertices as particles that interact with their neighbors while being guided by an implicit function, it resembles the particle system methods described in [21, 25]. (An important difference is that skin particles have explicit connectivity information.) Skin is related to the large body of work on implicit surfaces, particularly offset and convolution surfaces [3, 17] and blobby modeling [2, 4], in the constructive approach to modeling it provides.

For multiresolution editing, we use Loop subdivision meshes [13], including the rules for defining creases and corners described in [9, 19]. Previous work on subdivision surfaces has primarily focused on specifying subdivision rules and analyzing properties of the limit surface, *given a control mesh*. Hoppe *et al.* [9] present a means to construct a good-quality control mesh for a subdivision surface that approximates a given polygon mesh (typically scanned from a real model). We are not aware of any work that discusses ways to construct suitable control meshes for subdivision surfaces *from scratch*. Recently Zorin *et al.* [27] and Pulli [15] presented systems that let the user add detail to a subdivision surface (given the control mesh). With these systems the user edits details of the subdivision surface (at a given resolution) by manipulating individual vertices directly. The skin algorithm, though originally designed to construct a good-quality control mesh in the first place, is readily extended to provide a mechanism for adding detail at finer scales more conveniently than by individual control point manipulations.

3 Terms and definitions

Our meshes are triangle meshes, represented in the usual way: each consists of a collection of vertices, edges and faces with local connectivity information. That is, each vertex stores pointers to its adjacent edges; each edge stores pointers to the two vertices that define it and to at most two adjacent faces; each face stores pointers to the three edges and vertices that define it. *Skeletons*, represented with this mesh data structure, may be non-self-intersecting closed surfaces, surfaces with boundary, polylines or even isolated points. With each skeleton is associated a positive real-valued *offset*. The skin algorithm iteratively modifies the *skin* mesh (a closed surface) to roughly fit the distance surface defined by the skeletons and their offsets. Each skeleton also has an associated *target length* that specifies the desired size of triangles the skin should generate when growing over that skeleton. We refer to vertices of the skin mesh as *particles*. It is convenient to associate with each particle a *reference length* that measures the scale of the skin mesh near that particle. We take the reference length to be the average length of the particle's adjacent edges.

Let S denote the set of skeletons. For a given skeleton $s \in S$ with associated offset r_s, and a point $\mathbf{x}$ in space, let $d_s(\mathbf{x})$ denote the signed distance from $\mathbf{x}$ to s. That is, $d_s(\mathbf{x})$ is the distance from $\mathbf{x}$ to the nearest point on s, but if s is a closed surface and $\mathbf{x}$ is inside it, the "distance" is taken to be negative. Now let $f_s(\mathbf{x}) = d_s(\mathbf{x}) - r_s$. The *distance surface* defined by a single skeleton and its offset is just the implicit surface defined by $f_s(\mathbf{x}) = 0$. The distance surface defined by the collection of skeletons and their offsets is the implicit surface defined by $F(\mathbf{x}) = 0$, where F is the continuous function given by:

$$F(\mathbf{x}) = \min\{f_s(\mathbf{x}) : s \in S\}.$$

The rules for positioning particles described in section 4.1 are designed so that the skin approximates this surface. They also allow us to evaluate the implicit function efficiently.

4 The skin algorithm

At a high level, the skin algorithm closely resembles algorithms described in [10] and [24] for iteratively evolving a triangle mesh. The primary difference is in the implementation of step (1) of the main loop:

repeat
(1) reposition particles
(2) modify the skin's connectivity
until no changes occur

Intuitively, in step (1) particles move toward the implicit surface while tending to distribute themselves more evenly and smoothing out wrinkles and bumps in the skin. In step (2), the connectivity of the skin is modified to produce triangles that are more nearly equilateral and whose size is roughly the target length specified by nearby skeletons. We now describe steps (1) and (2) in detail.

4.1 Repositioning particles

The rules for repositioning particles are designed to allow the skin to grow adaptively to conform to the changing implicit surface as skeletons are added, repositioned, removed, or their offsets are edited. In step (1) of the skin algorithm, each particle is visited in turn. The new position $\mathbf{x}'_p$ of particle p is computed as a weighted sum of its current position $\mathbf{x}_p$, the centroid $\mathbf{c}_p$ of its neighbors' positions, and a target position $\mathbf{t}_p$ chosen to bring the particle closer to the implicit surface:

$$\mathbf{x}'_p = \alpha\mathbf{x}_p + \beta\mathbf{c}_p + \gamma\mathbf{t}_p.$$

The weights α, β, and γ are non-negative and sum to one. We always take $\alpha = 0.3$, and choose β according to how smooth the skin is near p. (Then γ is taken to be $1 - \alpha - \beta$.) We estimate the smoothness of the skin near p by calculating the minimum, over each edge adjacent to p, of the dot product of the normals of the two faces adjacent to that edge. This value (call it m) lies between 1 and -1. When m is close to 1, the skin is smooth near p; when

m is close to -1, the skin contains one or more sharp edges near p. A large value for β results in a greater smoothing effect, so we choose $\beta = (1-\alpha)\beta_{scale}$, where β_{scale} is the function of m shown in figure 2. This function, chosen heuristically, seems to produce good results. It has the effect of letting the skin move briskly toward the implicit surface where the skin is reasonably smooth. Where the skin is bumpy or uneven, it slows or stops moving until it smooths out again, then resumes its motion toward the implicit surface.

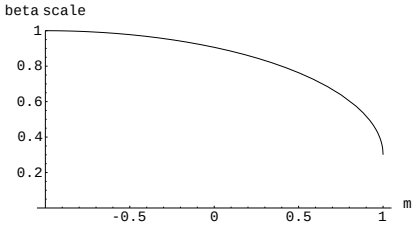


Figure 2 The coefficient β_{scale} chosen as a function of the minimum dot product m. This function is a quarter ellipse.

The target position $\mathbf{t}_p$ is chosen so that the skin tends to expand when inside the implicit surface, stay fixed when on it, and contract when outside it. We can determine whether the particle is inside, on, or outside the implicit surface according to whether $F(\mathbf{x}_p)$ is negative, zero, or positive, respectively. Also, by definition, the magnitude of this value specifies a lower bound on the distance from the particle to the nearest point on the implicit surface. We thus set the target position to be a displacement along the particle's surface normal $\vec{\mathbf{n}}_p$ by an amount d determined by $F(\mathbf{x}_p)$:

$$\mathbf{t}_p = \mathbf{x}_p + d\vec{\mathbf{n}}_p.$$

We can take $d = -F(\mathbf{x}_p)$ unless this value is large (that is, the particle is far from the implicit surface). To prevent the particle from taking excessively large steps, we restrict the magnitude of d to be no more than the reference length of the particle.

Because β is always nonzero, skin particles tend not to arrive exactly at the implicit surface. (Where it is concave they lie outside it, and where it is convex they lie inside it.) As mentioned earlier, we prefer this behavior, since the distance surface defined by the skeletons typically has unwanted sharp creases. For this reason, we can think of the implicit function as guiding rather than defining the skin's final shape.

4.2 Evaluating the implicit function

The skin algorithm is intended to work with skeletons that may be finely tessellated (as when representing muscle masses, such as those in figure 15). For the algorithm to be usable in an interactive setting, it is important to evaluate the implicit function efficiently. The particles do this by exploiting locality in two ways. To explain this, let's first assume there is only a single skeleton, s. Each particle p tracks a point $\mathbf{y}_p$ on s that is locally closest to p.[1] We refer to $\mathbf{y}_p$ as p's *track point*.[2] After updating its position $\mathbf{x}_p$, p updates its track point using the LOCAL-SEARCH procedure (figure 3), passing in a face f containing the track point:

$$\mathbf{y}_p \leftarrow \text{LOCAL-SEARCH}(f, \mathbf{x}_p).$$

Evaluating $f_s(\mathbf{x}_p)$ is now simple: For a non-closed skeleton surface, we evaluate the distance d from $\mathbf{x}_p$ to $\mathbf{y}_p$, and subtract r_s. For a closed surface, we negate d if the particle is inside it, and then subtract r_s. (Each skeleton surface is checked in a pre-process step to determine whether it is closed.) For closed surfaces, we can determine when a particle is inside provided that the particle has actually tracked the globally closest point $\mathbf{y}_p$ on the skeleton. For then the plane containing $\mathbf{y}_p$ that is perpendicular to the vector from $\mathbf{y}_p$ to $\mathbf{x}_p$ divides space into two parts. The skeleton locally near $\mathbf{y}_p$ lies all in one part and the particle lies in the other. An examination of the face normals around $\mathbf{y}_p$ determines whether p is inside or outside the surface.

[1] By locally closest, we mean that all points on s in a neighborhood of $\mathbf{y}_p$ are farther away.

[2] In fact, p stores a pointer to a face of s containing $\mathbf{y}_p$.

```
LOCAL-SEARCH(f, x)
    y ← closest point to x on f
    if y is on boundary of f
        foreach adjacent face f_i containing y
            if closest point to x on f_i ≠ y
                return LOCAL-SEARCH(f_i, x)
    return y
```

Figure 3 The LOCAL-SEARCH algorithm, starting from face f on a mesh M, traverses faces of M, always moving closer to point $\mathbf{x}$, to reach a point $\mathbf{y}$ on M that is locally closest to $\mathbf{x}$.

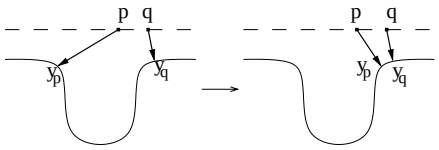


Figure 4 A particle p can use its neighbor's track point $\mathbf{y}_q$ to get out of a local minimum.

Of course, the LOCAL-SEARCH procedure may return a point on the skeleton that is only *locally* closest to p (see figure 4). To overcome this, particles exploit a second kind of locality: they share information with their immediate neighbors. Thus, to evaluate $f_s(\mathbf{x}_p)$, p runs the LOCAL-SEARCH procedure starting from its own track point, and then again starting from the track points of each of its neighbors in turn. The particle selects the result yielding the smallest distance, and stores this as its new track point.

When there are multiple skeletons, p can switch to a track point on a different skeleton if doing so yields a smaller *value* for the implicit function. It can switch to a different track point on the same skeleton, though, only if doing so yields a smaller *distance*. The distinction between these cases occurs only when the skeleton is a closed surface. In this case, the particle should first find the closest point on the skeleton, then evaluate the implicit function from the signed distance to this point. Figure 5 shows examples of these two cases.

When a new skeleton is added, it is sufficient for a single particle (lying inside the skeleton's distance surface) to begin tracking its closest point on the skeleton. That particle will then recruit its neighbors to track points on the new skeleton, and in the next iteration they will recruit their neighbors, and so on.

It sometimes happens that a particle has no track point: for example, when a skeleton is removed, particles tracking points on it forget their track points. In this case particles simply take $\beta = 1-\alpha$, which causes them to move towards the centroids of their neighbors. The region of skin that covered that skeleton will wither away, until all of its particles are either destroyed in edge collapses (described in the next section) or acquire new track points on other skeletons from their neighbors.

There are two other cases in which particles forget their track points. If a particle is outside of the implicit surface but its surface normal points toward its track point, the particle would, according to the rules described above, tend to back away from the track point,

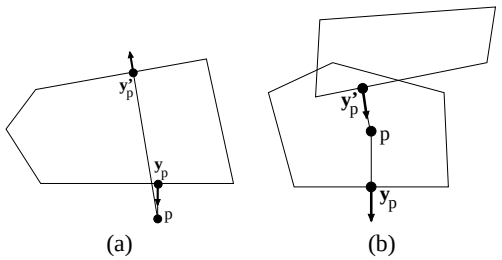


Figure 5 In (a), p is deciding between tracking $\mathbf{y}_p$ or $\mathbf{y}'_p$. Since both points lie on the same skeleton, it will choose $\mathbf{y}_p$. Note that the surface normal at $\mathbf{y}'_p$ is facing away from p, so tracking $\mathbf{y}'_p$ would yield a negative value for the implicit function, while $\mathbf{y}_p$ would yield a positive value. In (b), $\mathbf{y}'_p$ is closer. However, $\mathbf{y}_p$ is on a different skeleton, and will yield a lower implicit function value because its surface normal is facing away from p. p should therefore choose to track $\mathbf{y}_p$.

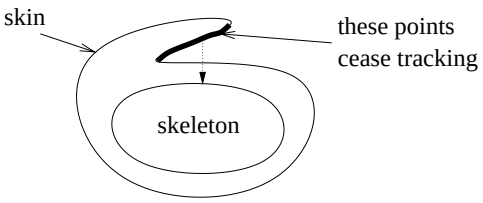


Figure 6 When the skin folds over so that an exterior particle's normal points towards the skeleton, the particle ignores the track point to avoid "backing away."

moving farther and farther from the implicit surface (see figure 6). To prevent this, particles in this case simply forget their track points and take $\beta = 1-\alpha$ as above. The second case occurs when a particle is adjacent to a *stressed* edge – that is, an edge whose two adjacent faces form an angle of less than 60 degrees. We discuss this case further in section 5.

4.3 Modifying the mesh connectivity

In step (2) of the skin algorithm we modify the skin's connectivity by performing the edge operations shown in figure 7. These operations were described in [10, 24]; conditions under which the operations are valid are discussed in [10].

We swap an edge when doing so increases the minimum angle within its adjacent faces. As noted by Welch and Witkin [24], repeated application of this swap operation (always increasing the minimum angle) computes a constrained Delaunay triangulation. That is, it maximizes the minimum angle over all the triangles of the mesh.

Each edge has an associated target length. We split an edge if it is longer than 1.5 times its target length, and collapse the edge if it is less than half its target length. These numbers are chosen in part so that a collapse operation will not immediately be invoked on an edge that has just been split. We compute the target length for each edge as follows.

As mentioned in section 3, each skeleton has an associated target length that the user can set. This specifies the desired size of triangles for portions of skin growing over that skeleton. We also store a target length value in each particle. In the repositioning step, each particle sets its target length to a weighted sum of its current value, the average target length of the particle's neighbors, and the target length of the skeleton currently tracked by the particle. (We use weights of 0.4, 0.4 and 0.2, respectively.) The target length of an edge is taken to be the average of the target lengths of its two adjacent particles.

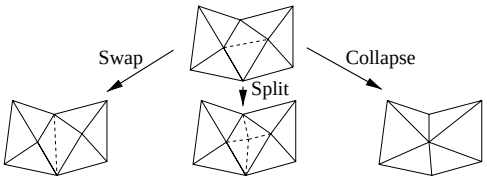


Figure 7 We *swap* to improve the minimum angle of a triangle. A *split* introduces a new vertex by splitting a long edge. A *collapse* removes a vertex by destroying a short edge.

Splitting and collapsing edges as described tends to produce edges of approximately the desired length. Interleaving steps (1) and (2) of the skin algorithm further tends to equalize edge lengths, allowing particles to redistribute themselves more evenly after edge operations are performed. In fact, we control the number of split and collapse operations that can occur between repositioning steps by sorting edges by the ratio of their actual length to their target length, then considering split operations just on the last 5% of edges and collapse operations just on the first 5% of edges. Any edge that is too long (or too short) is still guaranteed to be split (or collapsed) eventually.

Each edge operation affects the valence of nearby vertices. Since Loop subdivision performs best on meshes with a majority of valence-six vertices, we modify the rules slightly to favor the formation of such vertices. For example, a proposed edge swap is evaluated according to: (1) how much it will increase or decrease the minimum angle interior to its adjacent faces, and (2) the extent to which it will increase (or decrease) the number of valence-six vertices around it. The other operations are "handicapped" in a similar way. Such handicapping yields a small but noticeable improvement in the quality of the resulting subdivision surfaces.

4.4 Geometric constraints – creases

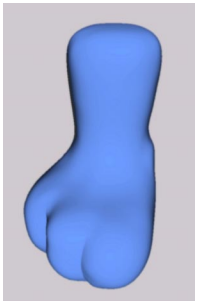


Figure 8 Indentations were created in the foot model using crease curves.

The user might want to constrain the skin's triangulation to include a sequence of edges aligned along some curve. Such an ability is required to model piecewise-smooth surfaces that include sharp "creases." Many organic forms, including human figures, can be described as having such features. To support this, we let the user first create a curve interactively by drawing onto the skin surface. We then split the triangles of the skin through which the curve passes to produce a sequence of edges aligned along this curve. Finally, the skin retriangulates in the neighborhood of the curve to maintain a good triangulation subject to the constraint that edges remain aligned along the curve. The user can then interact with the curve to change the shape of the surface. The foot model in figure 8 was created by drawing curves to model the creases between the toes, and then interactively pushing each curve into the foot to create the indentations. In this way, the user can create features that are not present in the implicit surface. Because the rules for repositioning vertices do not force them to interpolate the implicit surface, the region of skin around the crease blends smoothly to meet it.

For particles and edges lying on a crease, we modify the skin algorithm as follows. First, edges lying along a crease are not allowed to swap. Second, a particle p lying on a crease curve C is constrained

to stay on C. If p lies on an endpoint of C, its position is constrained to remain at that endpoint. Otherwise, it computes its target position $\mathbf{t}_p$ by taking a convex combination of its current position $\mathbf{x}_p$ and the midpoint $\mathbf{c}_p$ between its two neighboring particles on the crease curve. Then, instead of moving under the influence of the implicit function, p sets its new position $\mathbf{x}'_p$ to be the point on the curve that is closest to $\mathbf{t}_p$:

$$\mathbf{x}'_p = \underset{\mathbf{x}\in C}{\operatorname{argmin}} |\mathbf{t}_p - \mathbf{x}|.$$

We use a global search to compute this point, but a number of other techniques, such as gradient descent, could be applied depending on the underlying curve representation.

4.5 Termination

In the repositioning step, we actually assign a particle its new position only if it is appreciably different from the current position. Specifically, the new position must differ from the current one by more than .01 times the particle's reference length. This prevents the skin from continuing to make visually unnoticeable adjustments to the particle's positions after the surface is essentially stable.

Still, we have observed two cases in which the algorithm does not terminate. The first can occur when the topology of the skeletons' distance surface differs from that of the skin. The rules for modifying the skin's connectivity do not change its surface topology. If the user grows a skin over two skeletons that are initially close together, and then pulls them far apart, the skin does not spontaneously separate into two pieces. (In section 7 we describe how the user can explicitly invoke this type of topological change.) Instead, it remains joined by a thin strand, along which particles continue to move as edges are repeatedly split and collapsed. If the strand is cut, the skin heals itself and stabilizes.

The other case can occur when adjacent regions of skin have very different target lengths. In this case particles may continuously "swim" out from the region with small triangles toward the one with larger ones. This happens when a particle between the regions drifts toward the centroid of its neighbors in the region with larger triangles. As this happens, the particle's shorter edges are stretched, leading them to split and create new particles that repeat the process. We do not currently provide a solution to this, other than letting the user "freeze" the skin in that region.

Another problem involves premature termination. This can occur when the target length associated with a skeleton is large relative to one of its dimensions. In that case, the region of skin attempting to grow over the skeleton may necessarily have sharp edges. The tendency for particles to move to their neighbors' centroids may then outweigh their tendency to move toward the implicit surface, and the skin stops. An effective remedy is for the user to reduce the target length associated with the skeleton.

5 Preventing self-intersection

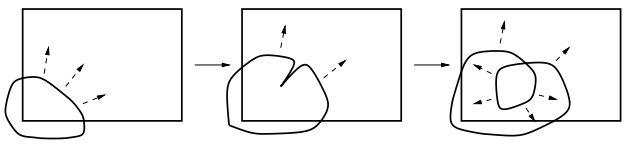


Figure 9 Uneven expansion of a skin can lead to self-intersection.

One problem with the skin algorithm as stated is shown in figure 9. During rapid expansion, one region of the skin may grow at a different rate from a neighboring region. The skin begins to buckle, and soon two adjacent regions of skin interpenetrate, forming a "bubble" that expands to produce a second, unwanted layer of skin.

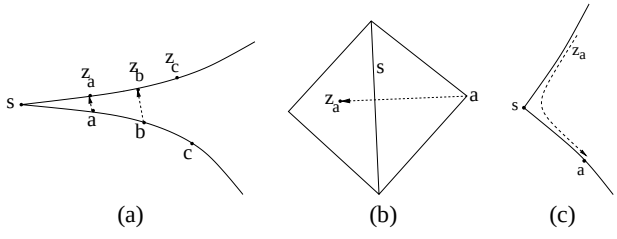


Figure 10 In (a), a and b track their penetration points $\mathbf{z}_a$ and $\mathbf{z}_b$. c is too far away from $\mathbf{z}_c$, so it has no penetration point. In (b), the stressed edge s tells adjacent particle a to track $\mathbf{z}_a$. In (c), the surface has smoothed out, so a's LOCAL-SEARCH returns to a.

In this section we discuss how we can prevent such self-intersection efficiently by reusing the particles' abilities to track a locally closest point on some mesh, and by sharing information as in section 4.2. A key observation is that local interpenetration typically occurs between two portions of surface separated by a chain of stressed edges. Our strategy is to initiate a non-penetration behavior only as needed, starting at stressed edges and propagating along opposing surface regions. We illustrate the idea in figure 10.

In figure 10(a), we see the skin surface, in cross section, starting to buckle along the stressed edge s. Particle a, which shares a face with s, receives notification from s to begin tracking its closest point $\mathbf{z}_a$ on the opposite face, shown in figure 10(b). We call $\mathbf{z}_a$ the *penetration point* of a.

As particles are repositioned, each updates its penetration point using the LOCAL-SEARCH procedure starting from its previous penetration point, as well as each of its neighbors' penetration points, if they have them. If a local search starting from any of these points returns a point that is not the particle's location, but is at a distance of less than 1.5 times the particle's reference length, the particle stores this point as its new penetration point. Particles are in this way always attempting to borrow penetration points from their neighbors.

Thus, b acquires a penetration point $\mathbf{z}_b$ by using the LOCAL-SEARCH starting from $\mathbf{z}_a$. When c does the same – borrowing from b – the result $\mathbf{z}_c$ is too distant (more than 1.5 times c's reference length) so c "forgets" the penetration point.

Whenever a particle with a valid penetration point passes through the surface containing its penetration point, it suspends the normal repositioning rules and moves back to the correct side of the surface. The result is that a crease temporarily forms in the skin but then resolves itself (figure 11). As the surface flattens out, particles track their penetration points back to themselves (figure 10(c)), upon which they forget their penetration point and revert to the normal repositioning rules.

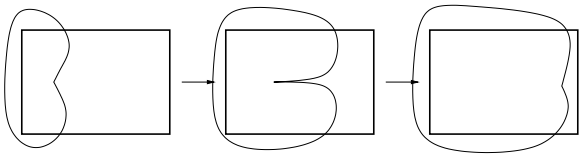


Figure 11 With the non-penetration behavior, a crease forms along the stressed edge, and then resolves itself as the skin grows.

6 Multiresolution editing

The approach to modeling we have targeted – sculpting complex surfaces by constructing the underlying masses that define their shape – naturally allows the user to work in a coarse-to-fine manner. That is, the user can begin with a coarse cylindrical shape to represent a torso, and then refine the surface, adding masses to define the shapes of individual muscles, bones, and tendons. The subdivision surface framework is naturally compatible with such an approach. With each level of refinement, the surface can effectively "resolve" detail at finer scales. The systems described in [15, 27] allow the user to add detail to a subdivision surface at multiple scales by manipulating vertices individually. We now explain how to extend the skin algorithm with little modification to support *subdivision skeletons*. A subdivision skeleton is like a regular skeleton except that it is active only at specific levels of subdivision. When the skin is subdivided, its vertices act as *subdivision particles* that track points on the subdivision skeletons active at that level.

Intuitively, subdivision particles that lie inside the implicit surface of a skeleton should move in the normal way (toward the implicit surface). A particle far from any subdivision skeleton should remain at the position it was initially assigned through subdivision, which we call its *base point*. In some range near the skeleton, a particle should smoothly blend between its base point and the displacement induced by the skeleton in the region of skin near the particle.

We achieve this as follows. Each particle evaluates the implicit function F as before, sharing information with its neighbors and tracking a point on some skeleton active at the current level of subdivision. Each particle has an associated "cutoff distance" M, equal to 3 times its reference length, beyond which it is unaffected by any skeleton. As before (section 4.1), its new position is computed from a weighted average of its current position, the centroid of its neighbors, and its target position:

$$\mathbf{x}'_p = \alpha\mathbf{x}_p + \beta\mathbf{c}_p + \gamma\mathbf{t}_p.$$

$F(\mathbf{x}_p)$	α	γ	$\mathbf{t}_p$
0	.3	.4	$\mathbf{x}_p$
$\geq M$	0	1	$\mathbf{b}_p$

We now choose the target position and weights α, β, and γ according to the value of the implicit function: when $F(\mathbf{x}_p) < 0$, the particle sets its target as before and uses weights $\alpha = 0.3$, $\beta = 0.3$, and $\gamma = 0.4$. When the implicit function is 0 or M we choose values for α and γ according to the above table, setting $\beta = 1 - \alpha - \gamma$. For values of $F(\mathbf{x}_p)$ between 0 and M we choose the target position and weights by interpolating the corresponding values listed in the table.

7 Topology changes

At each iteration, the skin evolves through local operations, using local information. Global properties of the skin surface, such as its genus, are not taken into account (or modified). Consequently, we do not automatically detect topological changes to the implicit surface. Figure 12 shows an example of a skin growing over a toroidal skeleton. When the new skeleton is first added, it changes the genus of the implicit surface. The skin grows around the torus and through itself in an attempt to fit the implicit surface. We could perhaps detect this interpenetration and prevent it. However, since it is global and not local, the method given in section 5 will not work.

Instead, we allow the user to explicitly change the genus of the skin by joining two sections of surface or by cutting a region of skin into two parts. Because the skin will evolve a poor triangulation into a good one, we can perform these operations without worrying about leaving a good triangulation.

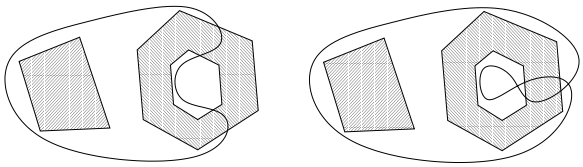


Figure 12 The skin does not detect the topology of the isosurface, which is toroidal, and hence will interpenetrate unless the user intervenes.

To cut the skin, we split the edges and faces that cross a user-specified cutting plane, and stitch up the resulting holes with a simple triangulation. (If desired, we can place crease curves along the cut to preserve its shape.) To join two portions of skin surface, we remove a face on each portion to create two holes, then connect the holes with a bridge of triangles. In either case, when editing the topology of the skin it's important to edit the skeletons accordingly so that the implicit surface matches the skin's new topology.

8 Software framework

We developed the skin algorithm within a larger system consisting of a collection of C++ libraries that provide extensive support for 3D modeling and interaction. The mesh class consists of a collection of vertices, edges, and faces (generically called simplices) that store connectivity information as described in section 3. To implement skin, we first introduced a separate class, Tessellator, that operates on a mesh's simplices according to some procedure, typically by editing connectivity and vertex positions. We made a small modification to the existing simplex classes to allow a tessellator to store arbitrary data directly on them.

The skin algorithm is implemented by a skin tessellator, which is a type of 2D tessellator – that is, it operates on a region of surface. Other types of tessellators include point tessellators and curve tessellators, which are 0D and 1D tessellators, respectively. Each Tessellator receives a callback every frame in which it iterates over its simplices and edits them according to its procedure. We enforce the rule that if multiple tessellators attempt to edit the same simplex, the tessellator with the lowest dimension wins. For example, we represent a curve constraint with a 1D tessellator that acts on its vertices and edges to approximate the curve. A skin tessellator operating on the surrounding surface is not allowed to edit the vertices and edges of the curve.

A vertex is considered active only if it has moved within the last 32 frames. Edges and faces are considered active if some contained vertex is active. We also activate simplices when the user performs an operation that could cause particles to move, such as changing a skeleton's offset distance or target length, or interactively moving a constraint curve. Tessellators operate only on simplices that are active. Since users typically edit just a local region of a surface at a time, this can significantly increase the interactivity of the system – particularly when the user edits a subdivision skin, which may have a large number of inactive vertices.

9 Discussion

As stated in the introduction, skin was designed to be used in an interactive free-form modeling setting. Since our ultimate aim is to create high-quality models, we believe our results should be judged by what types of models we can create within this framework. Although we have not designed the final interface for our system, we do have a preliminary implementation that lets the user create primitives either using SKETCH [26] or by importing them from other modeling packages. The user can interactively create a ball of skin

and grow it over a primitive. Editing operations include instructing the skin to grow over a new skeleton, adjusting a skeleton's offset distance or target length, drawing and editing crease curves, and changing the genus of the skin surface.

With this system we have created the models shown in figures 8, 13, 14, and 15. The torso model was created by growing skin over the skeleton shown in the top image of figure 15. The skeletons were created by a user with no artistic training using SKETCH. Some skeletons were created by growing skin over a sketched primitive. The skin surface was edited at both one and two levels of subdivision. For example, the muscles of the neck were added as subdivision skeletons. The foot and hand models demonstrate how skin can be used to create cartoonish effects. Both models are expressive, yet were built by growing skin over simple skeletons. The face model is more complex. Note the use of crease curves to delineate the mouth, nose, and eye sockets. This model would be difficult to create using a surface patch representation. The skeletons, created by a skilled computer artist using a conventional modeling system, are shown in the top image of figure 13.

10 Future work

Future work will proceed in two directions. We are continuing to design an end-to-end interface to allow a skilled artist to create expressive 3D scenes. This will include facilities for defining underlying skeletons and stick figures, along the lines of [6, 11, 26], an interface for creating and guiding the skin, an interface for editing the surface at fine levels of detail, and an interface for assigning and customizing procedural textures for rendering, some of which are described in these proceedings [12].

We have also begun to investigate modifying the skin algorithm to perform mesh optimization, remeshing for subdivision connectivity, and mesh decimation.

11 Acknowledgments

We thank Michael LeGrand for creating the skeletons for the face model. Thanks also to Andy van Dam and the Graphics Group, and to our sponsors: the NSF Graphics and Visualization Center, Advanced Network and Services, Alias/Wavefront, Autodesk, IBM, Intel, Microsoft, National Tele-Immersion Initiative, Sun Microsystems, and TACO. Lee Markosian received support for his graduate education from Intel through the Intel Foundation Ph.D. Fellowship program.

References

[1] Greg Albert, editor. *Basic Figure Drawing Techniques*. North Light Books, Cincinnati, Ohio, 1994.

[2] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[3] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In *SIGGRAPH 91 Conference Proceedings*, pp. 251–257. ACM SIGGRAPH, July 1991.

[4] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, pp. 109–116, March 1990.

[5] George B. Bridgman. *Constructive Anatomy*. Dover Publications, Inc., New York, 1973.

[6] Jonathan M. Cohen, Lee Markosian, Robert C. Zeleznik, John F. Hughes, and Ronen Barzel. An interface for sketching 3d curves. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, pp. 17–21, 1999.

[7] Louise Gordon. *How to Draw the Human Figure*. Penguin Books, New York, 1979.

[8] Burne Hogarth. *Dynamic Anatomy*. Watson-Guptill Publications, New York, paperback edition, 1990.

[9] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *SIGGRAPH 94 Conference Proceedings*, pp. 295–302. ACM SIGGRAPH, July 1994.

[10] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH 93 Conference Proceedings*, pp. 19–26. ACM SIGGRAPH, August 1993.

[11] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *SIGGRAPH 99 Conference Proceedings*. ACM SIGGRAPH, August 1999.

[12] Michael A. Kowalski, Lee Markosian, J.D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John F. Hughes. Art-based rendering of fur, grass, and trees. In *SIGGRAPH 99 Conference Proceedings*. ACM SIGGRAPH, August 1999.

[13] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[14] J. V. Miller, D. E. Breen, W. E. Lorensen, R. M. O'Bara, and M. J. Wozny. Geometrically deformed models: A method for extracting closed geometric models from volume data. In *SIGGRAPH 91 Conference Proceedings*, pp. 217–226. ACM SIGGRAPH, July 1991.

[15] Kari Pulli and Michael Lounsbery. Hierarchical editing and rendering of subdivision surfaces. Technical report, University of Washington, 1997.

[16] Walt Reed, editor. *The Figure: An Approach to Drawing and Construction*. North Light Books, Cincinnati, Ohio, 2nd edition, 1984.

[17] A.A.G. Requicha. Toward a theory of geometric tolerancing. *International Journal of Robotics Research*, 2(4):45–49, 1983.

[18] Fritz Schider. *An Atlas of Anatomy for Artists*. Dover Publications, Inc., New York, 3rd american edition, 1957.

[19] J. Schweitzer. *Analysis and application of subdivision surfaces*. PhD thesis, University of Washington, 1996.

[20] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *SIGGRAPH 97 Conference Proceedings*, pp. 279–286. ACM SIGGRAPH, August 1997.

[21] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In *SIGGRAPH 92 Conference Proceedings*, pp. 185–194. ACM SIGGRAPH, July 1992.

[22] Hal Tollison. *Cartooning*. Walter Foster Publishing, Inc., Laguna Hills, CA, 1989.

[23] C.W.A.M van Overveld and B. Wyvill. Shrinkwrap: an adaptive algorithm for polygonizing an implicit surface. Technical report, University of Calgary, 1993.

[24] William Welch and Andrew Witkin. Free–Form shape design using triangulated surfaces. In *SIGGRAPH 94 Conference Proceedings*, pp. 247–256. ACM SIGGRAPH, July 1994.

[25] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *SIGGRAPH 94 Conference Proceedings*, pp. 269–278. ACM SIGGRAPH, July 1994.

[26] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. SKETCH: An interface for sketching 3D scenes. In *SIGGRAPH 96 Conference Proceedings*, pp. 163–170. ACM SIGGRAPH, August 1996.

[27] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH 97 Conference Proceedings*, pp. 259–268. ACM SIGGRAPH, August 1997.

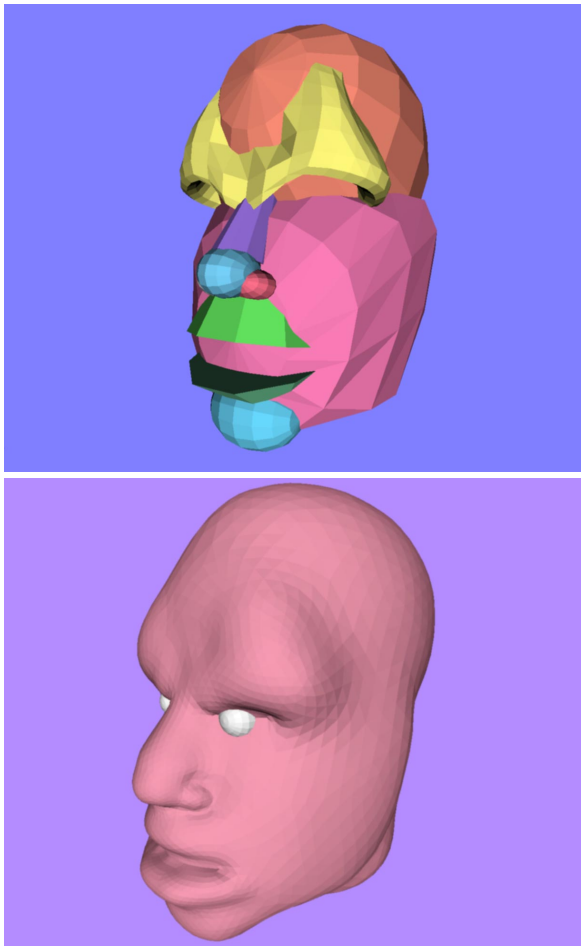


Figure 13 This face model consists of a skin surface with about 3500 particles. The skeleton, shown in the upper image, was created using a commercial modeling package. Notice the fine detail around the nose and eyes, which was created using creases in the base mesh, and subdivision skeletons.

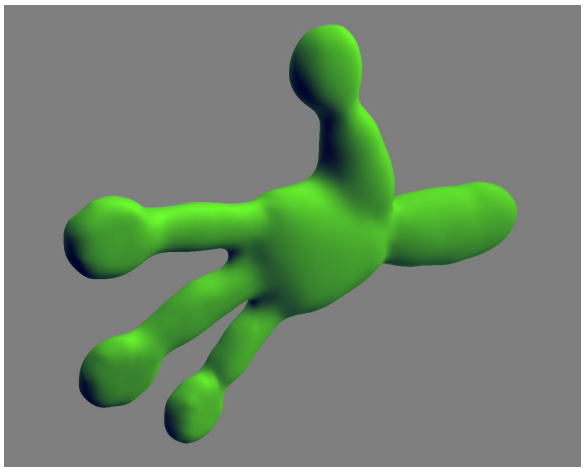


Figure 14 This frog hand is based on an image from a children's book. It was made from 10 skeleton meshes. Notice the smooth webbing between the fingers.

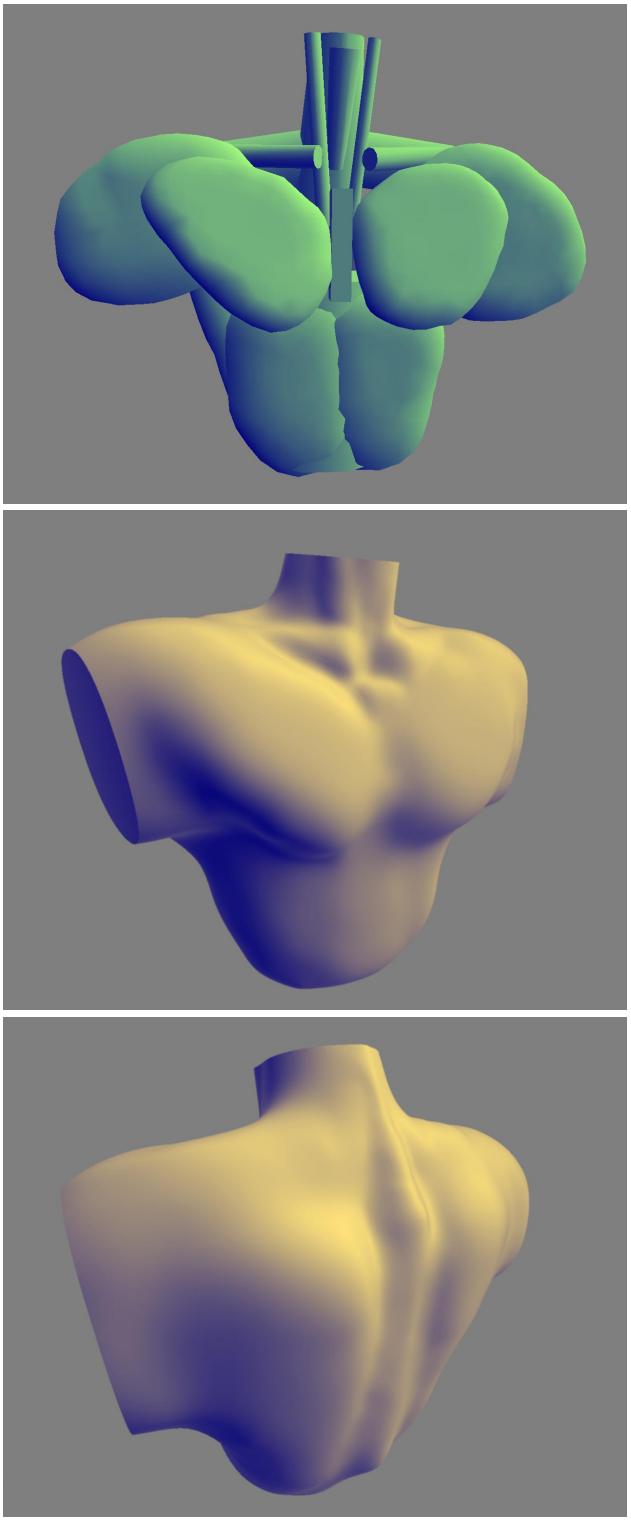


Figure 15 The torso model was edited at 2 levels of subdivision. For example, the vertical neck muscles and the spine were added as subdivision skeletons. Notice the indications of underlying muscles, especially around the neck.