Subdivision Surfaces in Character Animation

Tony DeRose   Michael Kass   Tien Truong

Pixar Animation Studios

Figure 1: Geri.

Abstract

The creation of believable and endearing characters in computer graphics presents a number of technical challenges, including the modeling, animation and rendering of complex shapes such as heads, hands, and clothing. Traditionally, these shapes have been modeled with NURBS surfaces despite the severe topological restrictions that NURBS impose. In order to move beyond these restrictions, we have recently introduced subdivision surfaces into our production environment. Subdivision surfaces are not new, but their use in high-end CG production has been limited.

Here we describe a series of developments that were required in order for subdivision surfaces to meet the demands of high-end production. First, we devised a practical technique for constructing provably smooth variable-radius fillets and blends. Second, we developed methods for using subdivision surfaces in clothing simulation including a new algorithm for efficient collision detection. Third, we developed a method for constructing smooth scalar fields on subdivision surfaces, thereby enabling the use of a wider class of programmable shaders. These developments, which were used extensively in our recently completed short film *Geri's game*, have become a highly valued feature of our production environment.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.3 [Computer Graphics]: Picture/Image Generation.

1 Motivation

The most common way to model complex smooth surfaces such as those encountered in human character animation is by using a patchwork of trimmed NURBS. Trimmed NURBS are used primarily because they are readily available in existing commercial systems such as Alias-Wavefront and SoftImage. They do, however, suffer from at least two difficulties:

1. Trimming is expensive and prone to numerical error.

2. It is difficult to maintain smoothness, or even approximate smoothness, at the seams of the patchwork as the model is

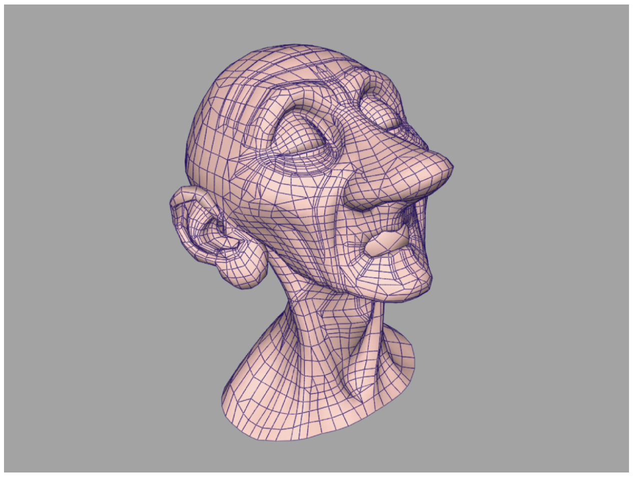

Figure 2: The control mesh for Geri's head, created by digitizing a full-scale model sculpted out of clay.

animated. As a case in point, considerable manual effort was required to hide the seams in the face of Woody, a principal character in *Toy Story*.

Subdivision surfaces have the potential to overcome both of these problems: they do not require trimming, and smoothness of the model is automatically guaranteed, even as the model animates.

The use of subdivision in animation systems is not new, but for a variety of reasons (several of which we address in this paper), their use has not been widespread. In the mid 1980s for instance, Symbolics was possibly the first to use subdivision in their animation system as a means of creating detailed polyhedra. The LightWave 3D modeling and animation system from NewTek also uses subdivision in a similar fashion.

This paper describes a number of issues that arose when we added a variant of Catmull-Clark [2] subdivision surfaces to our animation and rendering systems, Marionette and RenderMan [17], respectively. The resulting extensions were used heavily in the creation of Geri (Figure 1), a human character in our recently completed short film *Geri's game*. Specifically, subdivision surfaces were used to model the skin of Geri's head (see Figure 2), his hands, and his clothing, including his jacket, pants, shirt, tie, and shoes.

In contrast to previous systems such as those mentioned above, that use subdivision as a means to embellish polygonal models, our system uses subdivision as a means to define piecewise smooth surfaces. Since our system reasons about the limit surface itself, polygonal artifacts are never present, no matter how the surface animates or how closely it is viewed.

The use of subdivision surfaces posed new challenges throughout the production process, from modeling and animation to rendering. In modeling, subdivision surfaces free the designer from worrying about the topological restrictions that haunt NURBS modelers, but they simultaneously prevent the use of special tools that have been developed over the years to add features such as variable radius fillets to NURBS models. In Section 3, we describe an approach for introducing similar capabilities into subdivision surface models. The basic idea is to generalize the infinitely sharp creases of Hoppe *et. al.* [10] to obtain semi-sharp creases – that is, creases whose sharpness can vary from zero (meaning smooth) to infinite.

Once models have been constructed with subdivision surfaces, the problems of animation are generally easier than with corresponding NURBS surfaces because subdivision surface models are seamless, so the surface is guaranteed to remain smooth as the model is animated. Using subdivision surfaces for physically-based

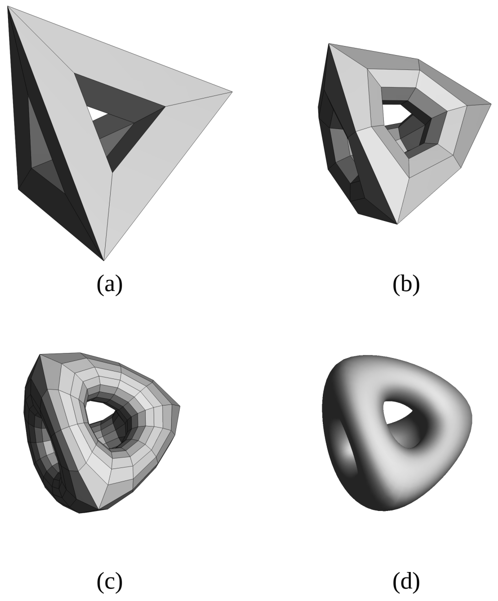

Figure 3: Recursive subdivision of a topologically complicated mesh: (a) the control mesh; (b) after one subdivision step; (c) after two subdivision steps; (d) the limit surface.

animation of clothing, however, poses its own difficulties which we address in Section 4. First, it is necessary to express the energy function of the clothing on subdivision meshes in such a way that the resulting motion does not inappropriately reveal the structure of the subdivision control mesh. Second, in order for a physical simulator to make use of subdivision surfaces it must compute collisions very efficiently. While collisions of NURBS surfaces have been studied in great detail, little work has been done previously with subdivision surfaces.

Having modeled and animated subdivision surfaces, some formidable challenges remain before they can be rendered. The topological freedom that makes subdivision surfaces so attractive for modeling and animation means that they generally do not admit parametrizations suitable for texture mapping. Solid textures [12, 13] and projection textures [9] can address some production needs, but Section 5.1 shows that it is possible to go a good deal further by using programmable shaders in combination with smooth scalar fields defined over the surface.

The combination of semi-sharp creases for modeling, an appropriate and efficient interface to physical simulation for animation, and the availability of scalar fields for shading and rendering have made subdivision surfaces an extremely effective tool in our production environment.

2 Background

A single NURBS surface, like any other parametric surface, is limited to representing surfaces which are topologically equivalent to a sheet, a cylinder or a torus. This is a fundamental limitation for any surface that imposes a global planar parameterization. A single subdivision surface, by contrast, can represent surfaces of arbitrary topology. The basic idea is to construct a surface from an arbitrary polyhedron by repeatedly subdividing each of the faces, as illustrated in Figure 3. If the subdivision is done appropriately, the limit of this subdivision process will be a smooth surface.

Catmull and Clark [2] introduced one of the first subdivision schemes. Their method begins with an arbitrary polyhedron called the control mesh. The control mesh, denoted M^0 (see Figure 3(a)), is subdivided to produce the mesh M^1 (shown in Figure 3(b)) by splitting each face into a collection of quadrilateral subfaces. A face having n edges is split into n quadrilaterals. The vertices of M^1 are computed using certain weighted averages as detailed below. The same subdivision procedure is used again on M^1 to produce the mesh M^2 shown in Figure 3(c). The subdivision surface is defined to be the limit of the sequence of meshes $M^0, M^1, \ldots$ created by repeated application of the subdivision procedure.

To describe the weighted averages used by Catmull and Clark it is convenient to observe that each vertex of M^{i+1} can be associated with either a face, an edge, or a vertex of M^i; these are called face, edge, and vertex points, respectively. This association is indicated in Figure 4 for the situation around a vertex v^0 of M^0. As indicated in the figure, we use f's to denote face points, e's to denote edge points, and v's to denote vertex points. Face points are positioned at the centroid of the vertices of the corresponding face. An edge point e_j^{i+1}, as indicated in Figure 4 is computed as

$$e_j^{i+1} = \frac{v^i + e_j^i + f_{j-1}^{i+1} + f_j^{i+1}}{4}, \tag{1}$$

where subscripts are taken modulo the valence of the central vertex v^0. (The valence of a vertex is the number of edges incident to it.) Finally, a vertex point v^i is computed as

$$v^{i+1} = \frac{n-2}{n} v^i + \frac{1}{n^2} \sum_j e_j^i + \frac{1}{n^2} \sum_j f_j^{i+1} \tag{2}$$

Vertices of valence 4 are called ordinary; others are called extraordinary.

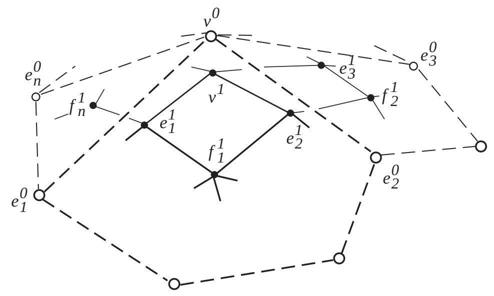

Figure 4: The situation around a vertex v^0 of valence n.

These averaging rules — also called subdivision rules, masks, or stencils — are such that the limit surface can be shown to be tangent plane smooth no matter where the control vertices are placed [14, 19].[1]

Whereas Catmull-Clark subdivision is based on quadrilaterals, Loop's surfaces [11] and the Butterfly scheme [6] are based on triangles. We chose to base our work on Catmull-Clark surfaces for two reasons:

1. They strictly generalize uniform tensor product cubic B-splines, making them easier to use in conjunction with existing in-house and commercial software systems such as Alias-Wavefront and SoftImage.

2. Quadrilaterals are often better than triangles at capturing the symmetries of natural and man-made objects. Tube-like surfaces — such as arms, legs, and fingers — for example, can be modeled much more naturally with quadrilaterals.

[1]Technical caveat for the purist: The surface is guaranteed to be smooth except for control vertex positions in a set of measure zero.

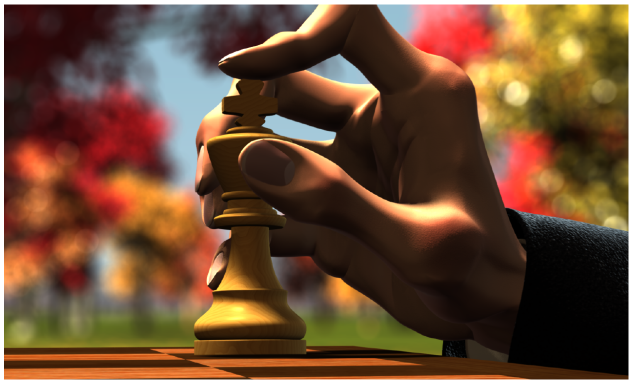

Figure 5: Geri's hand as a piecewise smooth Catmull-Clark surface. Infinitely sharp creases are used between the skin and the finger nails.

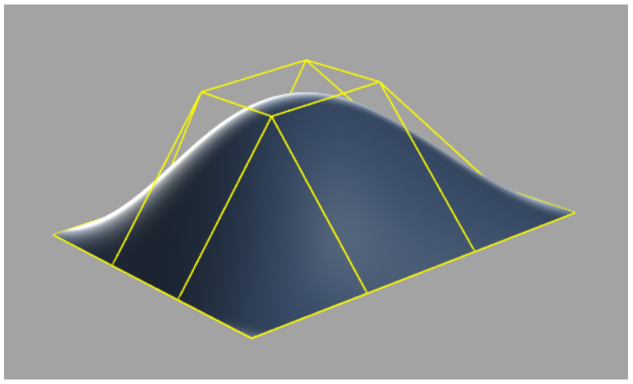

Figure 6: A surface where boundary edges are tagged as sharp and boundary vertices of valence two are tagged as corners. The control mesh is yellow and the limit surface is cyan.

Following Hoppe *et. al.* [10] it is possible to modify the subdivision rules to create piecewise smooth surfaces containing infinitely sharp features such as creases and corners. This is illustrated in Figure 5 which shows a close-up shot of Geri's hand. Infinitely sharp creases were used to separate the skin of the hand from the finger nails. Sharp creases can be modeled by marking a subset of the edges of the control mesh as sharp and then using specially designed rules in the neighborhood of sharp edges. Appendix A describes the necessary special rules and when to use them.

Again following Hoppe *et. al.*, we deal with boundaries of the control mesh by tagging the boundary edges as sharp. We have also found it convenient to tag boundary vertices of valence 2 as corners, even though they would normally be treated as crease vertices since they are incident to two sharp edges. We do this to mimic the behavior of endpoint interpolating tensor product uniform cubic B-spline surfaces, as illustrated in Figure 6.

3 Modeling fillets and blends

As mentioned in Section 1 and shown in Figure 5, infinitely sharp creases are very convenient for representing piecewise-smooth surfaces. However, real-world surfaces are never infinitely sharp. The corner of a tabletop, for instance, is smooth when viewed sufficiently closely. For animation purposes it is often desirable to capture such tightly curved shapes.

To this end we have developed a generalization of the Catmull-

Clark scheme to admit semi-sharp creases – that is, creases of controllable sharpness, a simple example of which is shown in Figure 7.

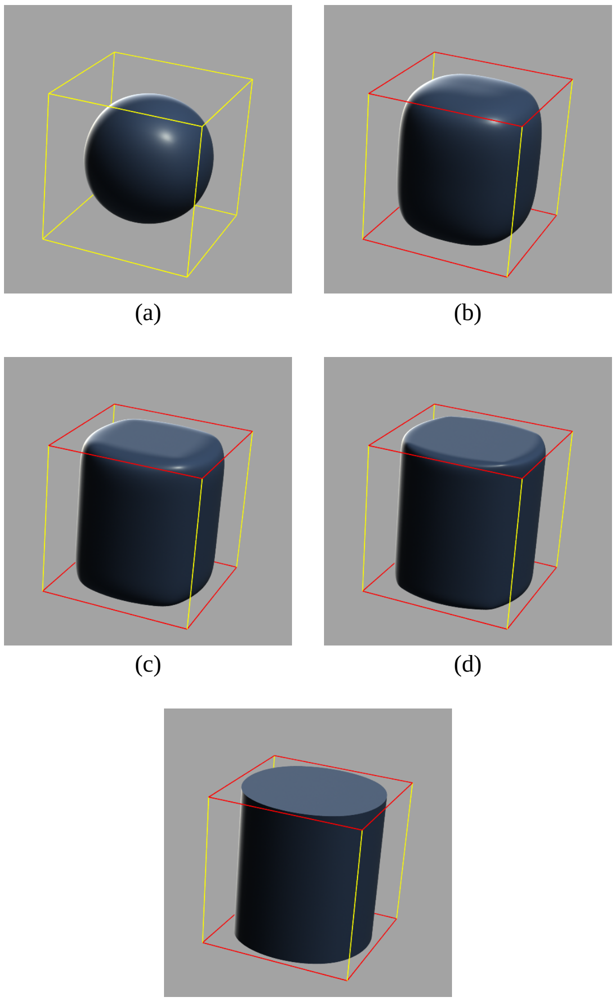

Figure 7: An example of a semi-sharp crease. The control mesh for each of these surfaces is the unit cube, drawn in wireframe, where crease edges are red and smooth edges are yellow. In (a) the crease sharpness is 0, meaning that all edges are smooth. The sharpnesses for (b), (c), (d), and (e) are 1, 2, 3, and infinite, respectively.

One approach to achieve semi-sharp creases is to develop subdivision rules whose weights are parametrized by the sharpness s of the crease. This approach is difficult because it can be quite hard to discover rules that lead to the desired smoothness properties of the limit surfaces. One of the roadblocks is that subdivision rules around a crease break a symmetry possessed by the smooth rules: typical smooth rules (such as the Catmull-Clark rules) are invariant under cyclic reindexing, meaning that discrete Fourier transforms can be used to prove properties for vertices of arbitrary valence (cf. Zorin [19]). In the absence of this invariance, each valence must currently be considered separately, as was done by Schweitzer [15]. Another difficulty is that such an approach is likely to lead to a zoo of rules depending on the number and configuration of creases through a vertex. For instance, a vertex with two semi-sharp creases passing through it would use a different set of rules than a vertex with just one crease through it.

Our approach is to use a very simple process we call hybrid subdivision. The general idea is to use one set of rules for a finite but arbitrary number of subdivision steps, followed by another set of rules that are applied to the limit. Smoothness therefore depends only on the second set of rules. Hybrid subdivision can be used to obtain semi-sharp creases by using infinitely sharp rules during the first few subdivision steps, followed by use of the smooth rules for subsequent subdivision steps. Intuitively this leads to surfaces that are sharp at coarse scales, but smooth at finer scales.

Now the details. To set the stage for the general situation where the sharpness can vary along a crease, we consider two illustrative special cases.

Case 1: A constant integer sharpness s crease: We subdivide s times using the infinitely sharp rules, then switch to the smooth rules. In other words, an edge of sharpness $s > 0$ is subdivided using the sharp edge rule. The two subedges created each have sharpness $s - 1$. A sharpness $s = 0$ edge is considered smooth, and it stays smooth for remaining subdivisions. In the limit where $s \to \infty$ the sharp rules are used for all steps, leading to an infinitely sharp crease. An example of integer sharpness creases is shown in Figure 7. A more complicated example where two creases of different sharpnesses intersect is shown in Figure 8.

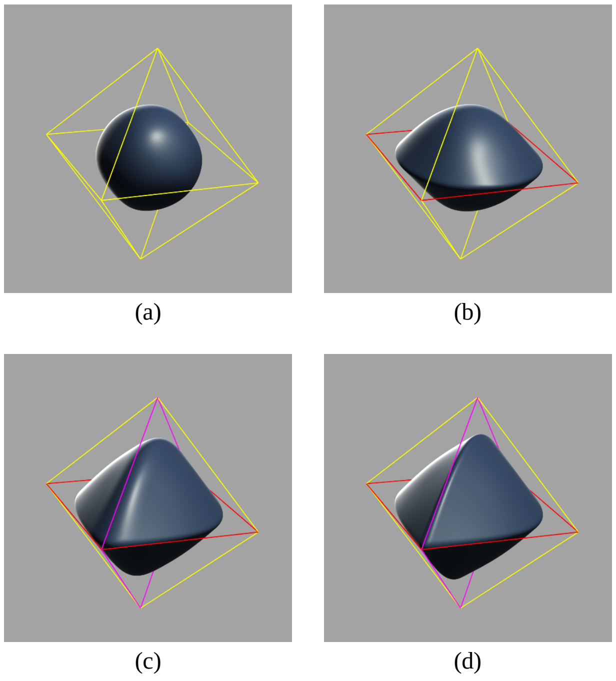

Figure 8: A pair of crossing semi-sharp creases. The control mesh for all surfaces is the octahedron drawn in wire frame. Yellow denotes smooth edges, red denotes the edges of the first crease, and magenta denotes the edges of the second crease. In (a) the crease sharpnesses are both zero; in (b), (c), and (d) the sharpness of the red crease is 4. The sharpness of the magenta crease in (b), (c), and (d) is 0, 2, and 4, respectively.

Case 2: A constant, but not necessarily integer sharpness s: the main idea here is to interpolate between adjacent integer sharpnesses. Let $s{\downarrow}$ and $s{\uparrow}$ denote the floor and ceiling of s, respectively. Imagine creating two versions of the crease: the first obtained by subdividing $s{\downarrow}$ times using the sharp rules, then subdividing one additional time using the smooth rules. Call the vertices of this first version $v{\downarrow}_0, v{\downarrow}_1, \ldots$. The second version, the vertices of which we denote by $v{\uparrow}_0, v{\uparrow}_1, \ldots$, is created by subdividing $s{\uparrow}$ times using the sharp rules. We take the $s{\uparrow}$-times subdivided semi-sharp crease to

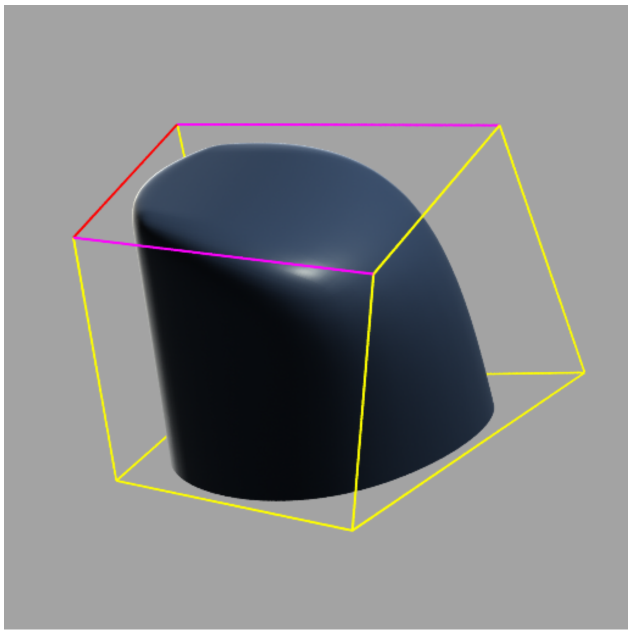
Figure 9: A simple example of a variable sharpness crease. The edges of the bottom face of the cubical control mesh are infinitely sharp. Three edges of the top face form a single variable sharpness crease with edge sharpnesses set to 2 (the two magenta edges), and 4 (the red edge).

have vertex positions $v_i^{s\uparrow}$ computed via simple linear interpolation:

$$v_i^{s\uparrow} = (1-\sigma)v{\downarrow}_i + \sigma v{\uparrow}_i \qquad (3)$$

where $\sigma = (s - s{\downarrow})/(s{\uparrow} - s{\downarrow})$. Subsequent subdivisions are done using the smooth rules. In the case where all creases have the same non-integer sharpness s, the surface produced by the above process is identical to the one obtained by linearly interpolating between the integer sharpness limit surfaces corresponding to $s{\downarrow}$ and $s{\uparrow}$. Typically, however, crease sharpnesses will not all be equal, meaning that the limit surface is not a simple blend of integer sharpness surfaces.

The more general situation where crease sharpness is non-integer and varies along a crease is presented in Appendix B. Figure 9 depicts a simple example. A more complex use of variable sharpness is shown in Figure 10.

4 Supporting cloth dynamics

The use of simulated physics to animate clothing has been widely discussed in the literature (cf. [1, 5, 16]). Here, we address the issues that arise when interfacing a physical simulator to a set of geometric models constructed out of subdivision surfaces. It is not our intent in this section to detail our cloth simulation system fully – that would require an entire paper of its own. Our goal is rather to highlight issues related to the use of subdivision surfaces to model both kinematic and dynamic objects.

In Section 4.1 we define the behavior of the cloth material by constructing an energy functional on the subdivision control mesh. If the material properties such as the stiffness of the cloth vary over the surface, one or more scalar fields (see Section 5.1) must be defined to modulate the local energy contributions. In Section 4.2 we describe an algorithm for rapidly identifying potential collisions involving the cloth and/or kinematic obstacles. Rapid collision detection is crucial to achieving acceptable performance.

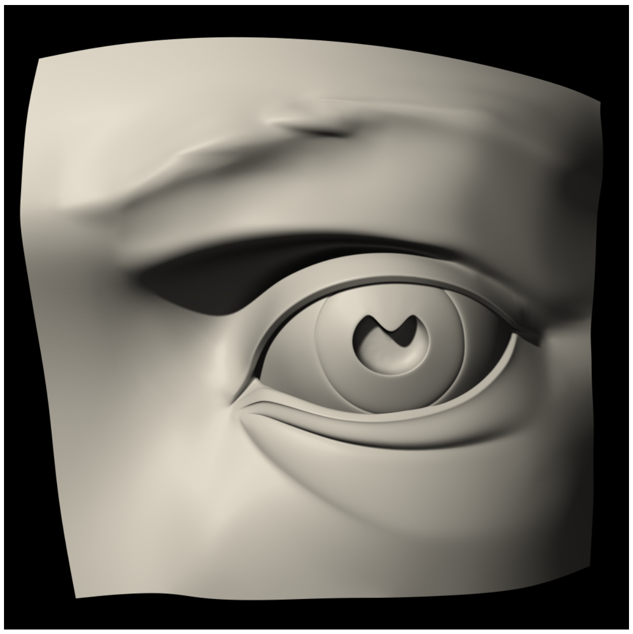
Figure 10: A more complex example of variable sharpness creases. This model, inspired by an Edouard Lanteri sculpture, contains numerous variable sharpness creases to reduce the size of the control mesh. The control mesh for the model made without variable sharpness creases required 840 faces; with variable sharpness creases the face count dropped to 627. Model courtesy of Jason Bickerstaff.

4.1 Energy functional

For physical simulation, the basic properties of a material are generally specified by defining an energy functional to represent the attraction or resistance of the material to various possible deformations. Typically, the energy is either specified as a surface integral or as a discrete sum of terms which are functions of the positions of surface samples or control vertices. The first type of specification typically gives rise to a finite-element approach, while the second is associated more with finite-difference methods.

Finite-element approaches are possible with subdivision surfaces, and in fact some relevant surface integrals can be computed analytically [8]. In general, however, finite-element surface integrals must be estimated through numerical quadrature, and this gives rise to a collection of special cases around extraordinary points. We chose to avoid these special cases by adopting a finite-difference approach, approximating the clothing with a mass-spring model [18] in which all the mass is concentrated at the control points.

Away from extraordinary points, Catmull-Clark meshes under subdivision become regular quadrilateral grids. This makes them ideally suited for representing woven fabrics which are also generally described locally by a gridded structure. In constructing the energy functions for clothing simulation, we use the edges of the subdivision mesh to correspond with the warp and weft directions of the simulated woven fabrics.

Since most popular fabrics stretch very little along the warp or weft directions, we introduce relatively strong fixed rest-length springs along each edge of the mesh. More precisely, for each edge from p_1 to p_2, we add an energy term $k_s E_s(p_1, p_2)$ where

$$E_s(p_1, p_2) = \frac{1}{2}\left(\frac{|p_1 - p_2|}{|p_1^* - p_2^*|} - 1\right)^2. \qquad (4)$$

Here, p_1^* and p_2^* are the rest positions of the two vertices, and k_s is the corresponding spring constant.

With only fixed-length springs along the mesh edges, the simulated clothing can undergo arbitrary skew without penalty. One way to prevent the skew is to introduce fixed-length springs along the diagonals. The problem with this approach is that strong diagonal springs make the mesh too stiff, and weak diagonal springs allow the mesh to skew excessively. We chose to address this problem by introducing an energy term which is proportional to the product of the energies of two diagonal fixed-length springs. If p_1 and p_2 are vertices along one diagonal of a quadrilateral mesh face and p_3 and p_4 are vertices along the other diagonal, the energy is given by $k_d E_d(p_1, p_2, p_3, p_4)$ where k_d is a scalar parameter that functions analagously to a spring constant, and where

$$E_d(p_1, p_2, p_3, p_4) = E_s(p_1, p_2) E_s(p_3, p_4). \tag{5}$$

The energy $E_d(p_1, p_2, p_3, p_4)$ reaches its minimum at zero when either of the diagonals of the quadrilateral face are of the original rest length. Thus the material can fold freely along either diagonal, while resisting skew to a degree determined by k_d. We sometimes use weak springs along the diagonals to keep the material from wrinkling too much.

With the fixed-length springs along the edges and the diagonal contributions to the energy, the simulated material, unlike real cloth, can bend without penalty. To add greater realism to the simulated cloth, we introduce an energy term that establishes a resistance to bending along virtual threads. Virtual threads are defined as a sequence of vertices. They follow grid lines in regular regions of the mesh, and when a thread passes through an extraordinary vertex of valence n, it continues by exiting along the edge $\lfloor n/2 \rfloor$-edges away in the clockwise direction. If p_1, p_2, and p_3 are three points along a virtual thread, the anti-bending component of the energy is given by $k_p E_p(p_1, p_2, p_3)$ where

$$E_p(p_1, p_2, p_3) = \frac{1}{2} * [C(p_1, p_2, p_3) - C(p_1^*, p_2^*, p_3^*)]^2 \tag{6}$$

$$C(p_1, p_2, p_3) = \left| \frac{p_3 - p_2}{|p_3^* - p_2^*|} - \frac{p_2 - p_1}{|p_2^* - p_1^*|} \right| \tag{7}$$

and p_1^*, p_2^*, and p_3^* are the rest positions of the three points.

By adjusting k_s, k_d and k_p both globally and locally, we have been able to simulate a reasonably wide variety of cloth behavior. In the production of *Geri's game*, we found that Geri's jacket looked a great deal more realistic when we modulated k_p over the surface of the jacket in order to provide more stiffness on the shoulder pads, on the lapels, and in an area under the armpits which is often reinforced in real jackets. Methods for specifying scalar fields like k_p over a subdivision surface are discussed in more detail in section 5.1.

4.2 Collisions

The simplest approach to detecting collisions in a physical simulation is to test each geometric element (i.e. point, edge, face) against each other geometric element for a possible collision. With N geometric elements, this would take N^2 time, which is prohibitive for large N. To achieve practical running times for large simulations, the number of possible collisions must be culled as rapidly as possible using some type of spatial data structure. While this can be done in a variety of different ways, there are two basic strategies: we can distribute the elements into a two-dimensional surface-based data structure, or we can distribute them into a three-dimensional volume-based data structure. Using a two-dimensional structure has several advantages if the surface connectivity does not change. First, the hierarchy can be fixed, and need not be regenerated each time the geometry is moved. Second, the storage can all be statically allocated. Third, there is never any need to rebalance the tree. Finally, very short edges in the surface need not give rise to deep branches in the tree, as they would using a volume-based method.

It is a simple matter to construct a suitable surface-based data structure for a NURBS surface. One method is to subdivide the (s,t) parameter plane recursively into an quadtree. Since each node in the quadtree represents a subsquare of the parameter plane, a bounding box for the surface restricted to the subsquare can be constructed. An efficient method for constructing the hierarchy of boxes is to compute bounding boxes for the children using the convex hull property; parent bounding boxes can then be computed in a bottom up fashion by unioning child boxes. Having constructed the quadtree, we can find all patches within ε of a point p as follows. We start at the root of the quadtree and compare the bounding box of the root node with a box of size 2ε centered on p. If there is no intersection, then there are no patches within ε of p. If there is an intersection, then we repeat the test on each of the children and recurse. The recursion terminates at the leaf nodes of the quadtree, where bounding boxes of individual subpatches are tested against the box around p.

Subdivision meshes have a natural hierarchy for levels finer than the original unsubdivided mesh, but this hierarchy is insufficient because even the unsubdivided mesh may have too many faces to test exhaustively. Since there is there is no global (s,t) plane from which to derive a hierarchy, we instead construct a hierarchy by "unsubdividing" or "coarsening" the mesh: We begin by forming leaf nodes of the hierarchy, each of which corresponds to a face of the subdivision surface control mesh. We then hierarchically merge faces level by level until we finish with a single merged face corresponding to the entire subdivision surface.

The process of merging faces proceeds as follows. In order to create the ℓth level in the hierarchy, we first mark all non-boundary edges in the $\ell - 1$st level as candidates for merging. Then until all candidates at the ℓth level have been exhausted, we pick a candidate edge e, and remove it from the mesh, thereby creating a "superface" f^* by merging the two faces f_1 and f_2 that shared e. The hierarchy is extended by creating a new node to represent f^* and making its children be the nodes corresponding to f_1 and f_2. If f^* were to participate immediately in another merge, the hierarchy could become poorly balanced. To ensure against that possibility, we next remove all edges of f^* from the candidate list. When all the candidate edges at one level have been exhausted, we begin the next level by marking non-boundary edges as candidates once again. Hierarchy construction halts when only a single superface remains in the mesh.

The coarsening hierarchy is constructed once in a preprocessing phase. During each iteration of the simulation, control vertex positions change, so the bounding boxes stored in the hierarchy must be updated. Updating the boxes is again a bottom up process: the current control vertex positions are used to update the bounding boxes at the leaves of the hierarchy. We do this efficiently by storing with each leaf in the hierarchy a set of pointers to the vertices used to construct its bounding box. Bounding boxes are then unioned up the hierarchy. A point can be "tested against" a hierarchy to find all faces within ε of the point by starting at the root of the hierarchy and recursively testing bounding boxes, just as is done with the NURBS quadtree.

We build a coarsening hierarchy for each of the cloth meshes, as well as for each of the kinematic obstacles. To determine collisions between a cloth mesh and a kinematic obstacle, we test each vertex of the cloth mesh against the hierarchy for the obstacle. To determine collisions between a cloth mesh and itself, we test each vertex of the mesh against the hierarchy for the same mesh.

5 Rendering subdivision surfaces

In this section, we introduce the idea of smoothly varying scalar fields defined over subdivision surfaces and show how they can be used to apply parametric textures to subdivision surfaces. We then describe a collection of implementation issues that arose when subdivision surfaces and scalar fields were added to RenderMan.

5.1 Texturing using scalar fields

NURBS surfaces are textured using four principal methods: parametric texture mapping, procedural texture, 3D paint [9], and solid texture [12, 13]. It is straightforward to apply 3D paint and solid texturing to virtually any type of primitive, so these techniques can readily be applied to texture subdivision surfaces. It is less clear, however, how to apply parametric texture mapping, and more generally, procedural texturing to subdivision surfaces since, unlike NURBS, they are not defined parametrically.

With regard to texture mapping, subdivision surfaces are more akin to polygonal models since neither possesses a global (s,t) parameter plane. The now-standard method of texture mapping a polygonal model is to assign texture coordinates to each of the vertices. If the faces of the polygon consist only of triangles and quadrilaterals, the texture coordinates can be interpolated across the face of the polygon during scan conversion using linear or bilinear interpolation. Faces with more than four sides pose a greater challenge. One approach is to pre-process the model by splitting such faces into a collection of triangles and/or quadrilaterals, using some averaging scheme to invent texture coordinates at newly introduced vertices. One difficulty with this approach is that the texture coordinates are not differentiable across edges of the original or pre-processed mesh. As illustrated in Figures 11(a) and (b), these discontinuities can appear as visual artifacts in the texture, especially as the model is animated.

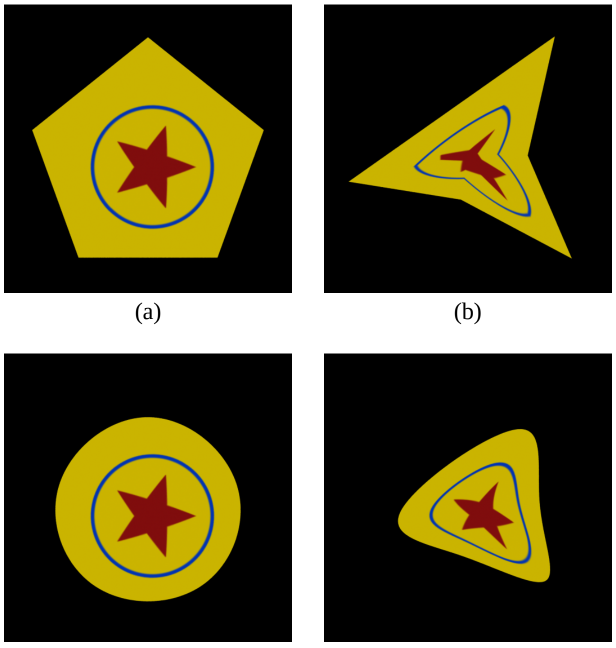

Figure 11: (a) A texture mapped regular pentagon comprised of 5 triangles; (b) the pentagonal model with its vertices moved; (c) A subdivision surface whose control mesh is the same 5 triangles in (a), and where boundary edges are marked as creases; (d) the subdivision surface with its vertices positioned as in (b).

Fortunately, the situation for subdivision surfaces is profoundly better than for polygonal models. As we prove in Appendix C, smoothly varying texture coordinates result if the texture coordinates (s,t) assigned to the control vertices are subdivided using the same subdivision rules as used for the geometric coordinates (x,y,z). (In other words, control point positions and subdivision can be thought of as taking place in a 5-space consisting of (x,y,z,s,t) coordinates.) This is illustrated in Figure 11(c), where the surface is treated as a Catmull-Clark surface with infinitely sharp boundary edges. A more complicated example of parametric texture on a subdivision surface is shown in Figure 12.

As is generally the case in real productions, we used a combination of texturing methods to create Geri: the flesh tones on his head and hands were 3D-painted, solid textures were used to add fine detail to his skin and jacket, and we used procedural texturing (described more fully below) for the seams of his jacket.

The texture coordinates s and t mentioned above are each instances of a scalar field; that is, a scalar-valued function that varies over the surface. A scalar field f is defined on the surface by assigning a value f_v to each of the control vertices v. The proof sketch in Appendix C shows that the function $f(p)$ created through subdivision (where p is a point on the limit surface) varies smoothly wherever the subdivision surface itself is smooth.

Scalar fields can be used for more than just parametric texture mapping — they can be used more generally as arbitrary parameters to procedural shaders. An example of this occurs on Geri's jacket. A scalar field is defined on the jacket that takes on large values for points on the surface near a seam, and small values elsewhere. The procedural jacket shader uses the value of the this field to add the apparent seams to the jacket. We use other scalar fields to darken Geri's nostril and ear cavities, and to modulate various physical parameters of the cloth in the cloth simulator.

We assign scalar field values to the vertices of the control mesh in a variety of ways, including direct manual assignment. In some cases, we find it convenient to specify the value of the field directly at a small number of control points, and then determine the rest by interpolation using Laplacian smoothing. In other cases, we specify the scalar field values by painting an intensity map on one or more rendered images of the surface. We then use a least squares solver to determine the field values that best reproduce the painted intensities.

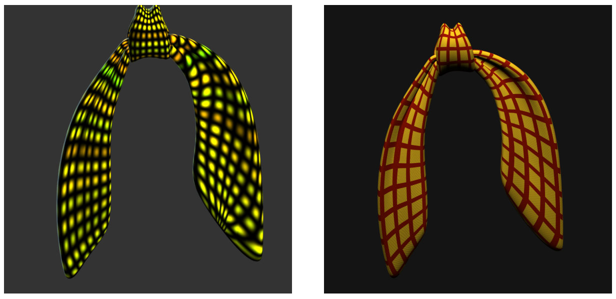

Figure 12: Gridded textures mapped onto a bandanna modeled using two subdivision surfaces. One surface is used for the knot, the other for the two flaps. In (a) texture coordinates are assigned uniformly on the right flap and nonuniformly using smoothing on the left to reduce distortion. In (b) smoothing is used on both sides and a more realistic texture is applied.

5.2 Implementation issues

We have implemented subdivision surfaces, specifically semi-sharp Catmull-Clark surfaces, as a new geometric primitive in RenderMan.

Our renderer, built upon the REYES architecture [4], demands that all primitives be convertible into grids of micropolygons (i.e. half-pixel wide quadrilaterals). Consequently, each type of primitive must be capable of splitting itself into a collection of subpatches, bounding itself (for culling and bucketing purposes), and dicing itself into a grid of micropolygons.

Each face of a Catmull-Clark control mesh can be associated with a patch on the surface, so the first step in rendering a Catmull-Clark surface is to split it in into a collection of individual patches. The control mesh for each patch consists of a face of the control mesh together with neighboring faces and their vertices. To bound each patch, we use the knowledge that a Catmull-Clark surface lies within the convex hull of its control mesh. We therefore take the bounding box of the mesh points to be the bounding box for the patch. Once bounded, the primitive is tested to determine if it is diceable; it is not diceable if dicing would produce a grid with too many micropolygons or a wide range of micropolygon sizes. If the patch is not diceable, then we split each patch by performing a subdivision step to create four new subpatch primitives. If the patch is diceable, it is repeatedly subdivided until it generates a grid with the required number of micropolygons. Finally, we move each of the grid points to its limit position using the method described in Halstead *et. al.* [8].

An important property of Catmull-Clark surfaces is that they give rise to bicubic B-splines patches for all faces except those in the neighborhood of extraordinary points or sharp features. Therefore, at each level of splitting, it is often possible to identify one or more subpatches as B-spline patches. As splitting proceeds, more of the surface can be covered with B-spline patches. Exploiting this fact has three advantages. First, the fixed 4×4 size of a B-spline patch allows for efficiency in memory usage because there is no need to store information about vertex connectivity. Second, the fact that a B-spline patch, unlike a Catmull-Clark patch, can be split independently in either parametric direction makes it possible to reduce the total amount of splitting. Third, efficient and well understood forward differencing algorithms are available to dice B-spline patches [7].

We quickly learned that an advantage of semi-sharp creases over infinitely sharp creases is that the former gives smoothly varying normals across the crease, while the latter does not. This implies that if the surface is displaced in the normal direction in a creased area, it will tear at an infinitely sharp crease but not at a semi-sharp one.

6 Conclusion

Our experience using subdivision surfaces in production has been extremely positive. The use of subdivision surfaces allows our model builders to arrange control points in a way that is natural to capture geometric features of the model (see Figure 2), without concern for maintaining a regular gridded structure as required by NURBS models. This freedom has two principal consequences. First, it dramatically reduces the time needed to plan and build an initial model. Second, and perhaps more importantly, it allows the initial model to be refined locally. Local refinement is not possible with a NURBS surface, since an entire control point row, or column, or both must be added to preserve the gridded structure. Additionally, extreme care must be taken either to hide the seams between NURBS patches, or to constrain control points near the seam to create at least the illusion of smoothness.

By developing semi-sharp creases and scalar fields for shading, we have removed two of the important obstacles to the use of subdivision surfaces in production. By developing an efficient data structure for culling collisions with subdivisions, we have made subdivision surfaces well suited to physical simulation. By developing a cloth energy function that takes advantage of Catmull-Clark mesh structure, we have made subdivision surfaces the surfaces of choice for our clothing simulations. Finally, by introducing Catmull-Clark subdivision surfaces into our RenderMan implementation, we have shown that subdivision surfaces are capable of meeting the demands of high-end rendering.

A Infinitely Sharp Creases

Hoppe *et. al.* [10] introduced infinitely sharp features such as creases and corners into Loop's surfaces by modifying the subdivision rules in the neighborhood of a sharp feature. The same can be done for Catmull-Clark surfaces, as we now describe.

Face points are always positioned at face centroids, independent of which edges are tagged as sharp. Referring to Figure 4, suppose the edge $v^i\ e^i_j$ has been tagged as sharp. The corresponding edge point is placed at the edge midpoint:

$$e^{i+1}_j = \frac{v^i + e^i_j}{2}. \tag{8}$$

The rule to use when placing vertex points depends on the number of sharp edges incident at the vertex. A vertex with one sharp edge is called a dart and is placed using the smooth vertex rule from Equation 2. A vertex v^i with two incident sharp edges is called a crease vertex. If these sharp edges are $e^i_j v^i$ and $v^i e^i_k$, the vertex point v^{i+1} is positioned using the crease vertex rule:

$$v^{i+1} = \frac{e^i_j + 6v^i + e^i_k}{8}. \tag{9}$$

The sharp edge and crease vertex rules are such that an isolated crease converges to a uniform cubic B-spline curve lying on the limit surface. A vertex v^i with three or more incident sharp edges is called a corner; the corresonding vertex point is positioned using the corner rule

$$v^{i+1} = v^i \tag{10}$$

meaning that corners do not move during subdivision. See Hoppe *et. al.* [10] and Schweitzer [15] for a more complete discussion and rationale for these choices.

Hoppe *et. al.* found it necessary in proving smoothness properties of the limit surfaces in their Loop-based scheme to make further distinctions between so-called regular and irregular vertices, and they introduced additional rules to subdivide them. It may be necessary to do something similar to prove smoothness of our Catmull-Clark based method, but empirically we have noticed no anamolies using the simple strategy above.

B General semi-sharp creases

Here we consider the general case where a crease sharpness is allowed to be non-integer, and to vary along the crease. The following procedure is relatively simple and strictly generalizes the two special cases discussed in Section 3.

We specify a crease by a sequence of edges $e_1, e_2, \ldots$ in the control mesh, where each edge e_i has an associated sharpness $e_i.s$. We associate a sharpness per edge rather than one per vertex since there is no single sharpness that can be assigned to a vertex where two or more creases cross.[2]

[2]In our implementation we do not allow two creases to share an edge.

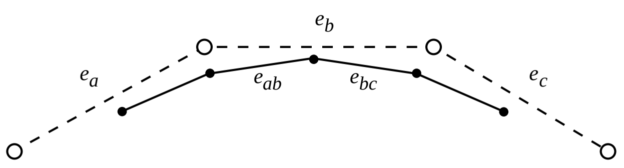

Figure 13: Subedge labeling.

During subdivision, face points are always placed at face centroids. The rules used when placing edge and vertex points are determined by examining edge sharpnesses as follows:

- An edge point corresponding to a smooth edge (i.e, $e.s = 0$) is computed using the smooth edge rule (Equation 1).
- An edge point corresponding to an edge of sharpness $e.s >= 1$ is computed using the sharp edge rule (Equation 8).
- An edge point corresponding to an edge of sharpness $e.s < 1$ is computed using a blend between smooth and sharp edge rules: specifically, let v_{smooth} and v_{sharp} be the edge points computed using the smooth and sharp edge rules, respectively. The edge point is placed at

$$(1 - e.s)v_{smooth} + e.sv_{sharp}. \tag{11}$$

- A vertex point corresponding to a vertex adjacent to zero or one sharp edges is computed using the smooth vertex rule (Equation 2).
- A vertex point corresponding to a vertex v adjacent to three or more sharp edge is computed using the corner rule (Equation 10).
- A vertex point corresponding to a vertex v adjacent to two sharp edges is computed using the crease vertex rule (Equation 9) if $v.s \geq 1$, or a linear blend between the crease vertex and corner masks if $v.s < 1$, where $v.s$ is the average of the incidence edge sharpnesses.

When a crease edge is subdivided, the sharpnesses of the resulting subedges is determined using Chaikin's curve subdivision algorithm [3]. Specifically, if e_a, e_b, e_c denote three adjacent edges of a crease, then the subedges e_{ab} and e_{bc} as shown in Figure 13 have sharpnesses

$$\begin{aligned} e_{ab}.s &= \max(\frac{e_a.s + 3e_b.s}{4} - 1, 0) \\ e_{bc}.s &= \max(\frac{3e_b.s + e_c.s}{4} - 1, 0) \end{aligned}$$

A 1 is subtracted after performing Chaikin's averaging to account for the fact that the subedges (e_{ab}, e_{bc}) are at a finer level than their parent edges (e_a, e_b, e_c). A maximum with zero is taken to keep the sharpnesses non-negative. If either e_a or e_b is infinitely sharp, then e_{ab} is; if either e_b or e_c is infinitely sharp, then e_{bc} is. This relatively simple procedure generalizes cases 1 and 2 described in Section 3. Examples are shown in Figures 9 and 10.

C Smoothness of scalar fields

In this appendix we wish to sketch a proof that a scalar field f is smooth as a function on a subdivision surface wherever the surface itself is smooth. To say that a function on a smooth surface S is smooth to first order at a point p on the surface is to say that there exists a parametrization $S(s,t)$ for the surface in the neighborhood of p such that $S(0,0) = p$, and such that the function $f(s,t)$ is differentiable and the derivative varies continuously in the neighborhood of $(0,0)$.

The characteristic map, introduced by Reif [14] and extended by Zorin [19], provides such a parametrization: the characteristic map allows a subdivision surface S in three space in the neighborhood of a point p on the surface to be written as

$$S(s,t) = (x(s,t), y(s,t), z(s,t)) \tag{12}$$

where $S(0,0) = p$ and where each of $x(s,t)$, $y(s,t)$, and $z(s,t)$ is once differentiable if the surface is smooth at p. Since scalar fields are subdivided according to the same rules as the x, y, and z coordinates of the control points, the function $f(s,t)$ must also be smooth.

Acknowledgments

The authors would like to thank Ed Catmull for creating the *Geri's game* project, Jan Pinkava for creating Geri and for writing and directing the film, Karen Dufilho for producing it, Dave Haumann and Leo Hourvitz for leading the technical crew, Paul Aichele for building Geri's head, Jason Bickerstaff for modeling most of the rest of Geri and for Figure 10, and Guido Quaroni for Figure 12. Finally, we'd like to thank the entire crew of *Geri's game* for making our work look so good.

References

[1] David E. Breen, Donald H. House, and Michael J. Wozny. Predicting the drape of woven cloth using interacting particles. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 365–372. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[2] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[3] G. Chaikin. An algorithm for high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

[4] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 95–102, July 1987.

[5] Martin Courshesnes, Pascal Volino, and Nadia Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 137–144. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[6] Nira Dyn, David Leven, and John Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, April 1990.

[7] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Prentice-Hall, 1990.

[8] Mark Halstead, Michael Kass, and Tony DeRose. Efficient, fair interpolation using Catmull-Clark surfaces. *Computer Graphics*, 27(3):35–44, August 1993.

[9] Pat Hanrahan and Paul E. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 215–223, August 1990.

[10] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics*, 28(3):295–302, July 1994.

[11] Charles T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, August 1987.

[12] Darwyn R. Peachey. Solid texturing of complex surfaces. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 279–286, July 1985.

[13] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 287–296, July 1985.

[14] Ulrich Reif. A unified approach to subdivision algorithms. Mathematisches Institute A 92-16, Universitaet Stuttgart, 1992.

[15] Jean E. Schweitzer. *Analysis and Application of Subdivision Surfaces*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1996.

[16] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 205–214, July 1987.

[17] Steve Upstill. *The RenderMan Companion*. Addison-Wesley, 1990.

[18] Andrew Witkin, David Baraff, and Michael Kass. An introduction to physically based modeling. SIGGRAPH Course Notes, Course No. 32, 1994.

[19] Denis Zorin. *Stationary Subdivision and Multiresolution Surface Representations*. PhD thesis, Caltech, Pasadena, 1997.