The Visual Computer

Adaptive polygonization of geometrically constrained surfaces

Jung-Hong Chuang[1],
Christoph M. Hoffmann[2],
Kun-Ming Ko[1],
Wei-Chung Hwang[1]

[1] Department of Computer Science
and Information Engineering,
National Chiao Tung University, Hsinchu, Taiwan,
Republic of China
[2] Department of Computer Science, Purdue University,
West Lafayette, IN 47907, USA

Many surfaces in geometric and solid modeling, including offsets and blends, are naturally defined from given surfaces subject to geometric constraints. Surfaces that are geometrically constrained can be uniformly defined as the projection of two-dimensional manifolds (2-surfaces) in n-dimensional space, where $n>3$. This definition can be used for given surfaces that are implicit or parametric. This paper presents a robust, adaptive polygonization algorithm for evaluating and visualizing geometrically constrained surfaces. Let $\mathcal{F}$ be the constrained surface, a 2-surface in n-space, and let $\pi(\mathcal{F})$ be its projection into the subspace spanned by the first three coordinates. Our polygonization algorithm computes $\pi(\mathcal{F})$. The method works directly with the n-space representation, but performs all major computations in 3-space. Techniques for triangulation, polygon decimation, and local refinement are also presented.

Key words: Constrained surfaces – Exact representation – Polygonization – Rendering

Correspondence to: J.-H. Chuang

1 Introduction

Implicit surfaces have recently become more important in geometric and solid modeling. In part, implicit surfaces have specific advantages over the traditional parametric surfaces. For example, many complex objects can be modeled more easily with implicit surfaces, and certain geometric operations, e.g., the membership classification problem, can be handled straightforwardly when implicit surface representations are available. Moreover, a number of sophisticated techniques that use implicit surfaces have been proposed, e.g., the substitution blending surfaces of Hoffmann and Hopcroft (1987).

While many surfaces as parametric or implicit surfaces can be formulated easily in 3-D space, certain surfaces including offsets, equidistant surfaces, and spherical blends cannot. Due to the occur of high algebraic degrees, the representation and generation of such surfaces can lead to great computational complexity and numerical instability and may require expensive symbolic manipulations. As an alternative, the *dimensionality paradigm* (Hoffmann 1990) provides a way to represent such surfaces in a higher-dimensional space with more variables, but simpler equations. In the dimensionality paradigm, complex constrained surfaces are defined as the natural projection of 2-surfaces in higher-dimensional space. With this representation, complex symbolic computations and numerically delicate operations can often be avoided, so that practical implementations can be realized.

In computer graphics, many surface-rendering algorithms rely on polygonal approximations of a surface. A polygonal approximation allows one to take advantage of hardware capabilities and reduces the cost of expensive ray casting in the rendering process. However, while the polygonization of parametric surfaces has been extensively studied and utilized as a tool for the evaluation of surface intersections (Barnhill et al. 1987; Lane and Riesenfeld 1980) and for rendering, it seems that much less attention has been paid to the polygonization of implicitly defined surfaces in the literature. Bloomenthal (1988) proposes an algorithm for computing the polygonization of an implicit surface based on space subdivision using octrees. Hall and Warren (1990) propose a method based on the subdivision of tetrahedra. Allgower and Gnutzmann (1987, 1991) present a simplicial continuation algorithm for obtaining a polygonization to a component of a 2-surface in $\mathbf{R}^n$, where $n\geq 3$.

The Visual Computer (1998) 14:455–470

All three methods fundamentally rely on vertex evaluation. Rheinboldt (1987) presents an algorithm that maps a triangulation of $\mathbf{R}^p$ to a p-manifold, where $p \geq 1$, and hence induces a triangulation on the p-manifold. Lorensen and Cline (1987) and Wyvill et al. (1986) propose methods that use vertex evaluation to derive isosurfaces from volume data or from spatial scalar field data. An extensive review of surface polygonization algorithms according to topological issues can be found in Ning and Bloomenthal (1993).

To polygonize geometrically constrained surfaces, algorithms using space subdivision or simplicial continuation can be generalized to compute the polygonal approximation of a 2-surface in high-dimensional space. However, as the dimension of ambient space increases, the complexity of computing the polygonization increases exponentially. Moreover, the projection of a polygonization from n-space to 3-space can be extremely complicated when n is large. To reduce the complexity sharply, we propose an algorithm that works with the n-space representation directly, but performs all major computations in 3-space, thus realizing huge computational savings. Our algorithm marches across the constrained surface in 3-space, using a grid to detect whether a particular volume of space has already been explored. It utilizes local parametric approximations derived from the n-space representation to guide the surface expansion and to coordinate the 3-space exploration with the n-space parameters defining the surface.

In Sect. 2, we briefly review the dimensionality paradigm and polygonization techniques for implicit surfaces. In Sect. 3, the proposed polygonization algorithm is described. In Sect. 4, we describe the decimation and the refinement processes. Section 5 addresses implementation issues and gives some examples. Section 6 adds some concluding remarks.

2 Constrained surfaces and surface polygonizations

2.1 The dimensionality paradigm

Many surfaces in geometric and solid modeling, including offsetting and blending surfaces, are defined from given surfaces subject to certain geometric constraints. This definition describes the surface in natural, intuitive geometric terms. However, the representations of such surfaces in 3-space often do not belong to the class of surfaces from which they are derived, and thus should be approximated. For example, the offset of a Bézier surface is not, in general, a Bézier surface.

Special classes of constrained surfaces have been studied before; for example, offset surfaces in Rossignac and Reguichen (1986), and blending surfaces in Filip (1989). Hoffmann (1990) proposes the *dimensionality paradigm* as a straightforward, exact, and uniform method for representing these and other surfaces defined by constraints. In the dimensionality paradigm, the defining constraints are translated into equations and the surface of interest is the natural projection of a 2-surface defined in higher-dimensional space. For example, the offset of $f(u, v, w)=0$ with radius r is the natural projection into (x, y, z)-space of the 2-surface defined as follows:

$$(x-u)^2+(y-v)^2+(z-w)^2-r^2=0 \tag{1}$$

$$f(u, v, w)=0 \tag{2}$$

$$-f_v(x-u)+f_u(y-v)=0 \tag{3}$$

$$-f_w(y-v)+f_v(z-w)=0, \tag{4}$$

where Eq. 1 represents the sphere centered on the surface point (u, v, w) with radius r, and Eqs. 3 and 4 constraines points on the sphere to the offset of $f=0$.

A solution point (x, y, z, u, v, w) of the system indicates that (u, v, w) is the foot point, or projection, of (x, y, z) on $f=0$. The r-offset of $f=0$ is the natural projection of the zero set into (x, y, z)-space. In general form, a constrained surface can be represented as the natural projection of the zero set of the following system of nonlinear equations into the subspace spanned by three variables:

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ f_2(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, \ldots, x_n) &= 0. \end{aligned} \tag{5}$$

The zero set $\mathcal{F}$ of the system of Eq. 5 is assumed to be locally a smooth 2-manifold in $\mathbf{R}^n$, so m is ordinarily equal to $n-2$. However, as discussed by

Hoffmann and Vermeer (1991), in certain cases $m>n-2$ is desirable for the system of Eq. 5 to exclude degeneracies. Several uniform methods for interrogating constrained surfaces have been given.

Bajaj et al. (1988) describe a uniform method for evaluating the intersection curve of such surfaces, and Chuang and Hoffmann (1992) present a uniform method for determining the surface curvature at a given point. The dimensionality paradigm has also been used to represent exactly the spine curve of spherical blends that is subject to intricate geometric constraints (Chuang et al. 1995; Chuang and Hwang 1997).

2.2 Surface polygonization methods

For implicit algebraic surfaces, Bloomenthal (1988) proposes a polygonization algorithm that uses an octree to decompose the space surrounding the surface. At the corners of each cube, the implicit function is sampled, and surface points on the edges are calculated by binary sectioning along an edge whose vertices are evaluated with opposite signs. The polygon labeling is done by proceeding towards the positive corner and clockwise about the face to the right until the next surface vertex is reached. For the adaptive subdivision, the crack between the face of two cubes with different levels of subdivision is closed by continuing polygon labeling on the faces of the cube with a higher level of subdivision.

Rheinboldt's (1987) moving frame method proceeds by triangulating the tangent space at a surface point, and transferring the triangulation to the surface with Newton iteration. Each vertex of the triangulation, after projection to a point on the surface, becomes the center of a new triangulation of its tangent space. The algorithm resolves any overlap locally, but cannot do so globally.

Allgower's method (1991) is based on a triangulation of ambient space. At each vertex of a simplex, the implicit surface function is evaluated, and a linear approximation is constructed from the vector of component function values. Next, the faces through which the linear approximant passes are deduced. Finally, with this information, adjacent simplices are considered in the same way. Note that the number of simplices in an elementary volume grows exponentially with the number of variables.

Hall and Warren (1990) construct an adaptive polygonization by recursive subdivision of space bounded by a tetrahedron. A tetrahedron is subdivided into four tetrahedra and an octahedron by slicing off each corner of the original tetrahedron, and then the octahedron is split further into eight similar tetrahedra. Maintaining a *honeycomb*, the surface vertices are easily labeled to form a polygon.

The vertex-evaluation-based techniques can be applied to derive isosurfaces of volume data or scalar field data (Lorensen and Cline 1987; Wyvill et al. 1986). The marching cube method proposed by Lorensen and Cline (1987) has been recognized as an effective and simple method for isosurface extraction; nevertheless, this method has three notable problems. First of all, the marching cube method explores all cells, including those containing no surface of interest (Wilhelms and Van Gelder 1992). Second, the marching cube method can generate an excessive number of triangles (Hoppe et al. 1993; Müller and Stark 1993; Schroeder et al. 1992). Finally, this method may produce surfaces with holes, because of ambiguity that may be found on the common faces of adjacent cubes (Neilson and Hamann 1991; Wilhelms and Van Gelder 1990).

3 Adaptive polygonization of constrained surfaces

For curves such as surface intersections, it is easy to derive a marching scheme that produces a polygonal approximation of the intersection curve. In order to obtain a similar scheme for evaluating surfaces, we need a device that orients the exploration in space and prevents inadvertently exploring the same neighborhood several times. The simplicial continuation of Allgower (1991) and the moving-frame method of Rheinboldt (1987) are of this type and based on the following idea. Given a manifold $\mathcal{F}$ defined by the system of Eq. 5 and on it a point $p=(p_1, p_2, \ldots, p_n)$, construct a polygonal approximation of $\mathcal{F}$, beginning at p and expanding in all directions.

In the case of constrained surfaces defined by the dimensionality paradigm, the surface that is ulti-

mately of interest is the projection $\pi(\mathcal{F})$ in the (x_1, x_2, x_3)-subspace. Since this surface is in a three-dimensional space, it is advantageous to construct the polygonal approximation only for the projection $\pi(\mathcal{F})$, and we do this by tracing the surface in the 3-space with help of a grid to detect whether a volume of space has already been explored. For each grid cube that the surface passes, a polygon is determined to approximate the surface within the cube. We describe the basic surface-tracing scheme.

Let $\mathcal{F}$ be a 2-surface in $\mathbf{R}^n$ defined by the system of Eq. 5, $\pi(\mathcal{F})$ its projection into (x_1, x_2, x_3)-space. Given a regular point p in $\mathbf{R}^n$, and a cube C in (x_1, x_2, x_3)-space with one of its edges, say e, containing $\pi(p)$. The basic surface-tracing procedure starts from $\pi(p)$ on e as follows:

1. At $\pi(p)$, construct a local parametric approximant $\Phi(s, t)$ of $\mathcal{F}$ at p.
2. Determine where the projected approximant $\pi(\Phi(s, t))$ intersects the edges of the cubes sharing e, and refine these intersections to $\pi(\mathcal{F})$ on edges of these cubes.
3. Select another new intersection via a tracing scheme and repeat steps 1 and 2.
4. Derive the polygonal approximation of $\pi(\mathcal{F})$ by labeling the surface intersections on edges of each transversal cube.

3.1 Local parametric approximation

The local parametric approximation $\Phi(s, t)=(\phi_1(s, t), \phi_2(s, t),\ldots, \phi_n(s, t))$ plays an essential role here since the corresponding points in $\mathbf{R}^n$ can be recovered via the coordinate functions of $\Phi(s, t)$ once the intersections of $\pi(\Phi(s, t))$ with grid edges are found in $\mathbf{R}^3$. Moreover, having these points in $\mathbf{R}^n$, intersections of $\pi(\mathcal{F})$ and a cube's edge can be refined with the system of Eq. 5 together with equations for the edge. That is, with the local parametric approximation, we are able to trace $\pi(\mathcal{F})$ in 3-space, even though the surface is formally defined in $\mathbf{R}^n$. This is the major advantage of the method since the dimension of the grid space does not depend on the number of variables used to define $\mathcal{F}$. In contrast, the methods of Allgower (1991) and Rheinboldt (1987) necessarily perform poorly when the dimension n is large. For $n>5$, those algorithms are too slow in practice.

For a given system of Eq. 5 and a point p on it, we seek a parametrically described solution

$$\Phi(s, t)=(\phi_1(s, t), \phi_2(s, t),\ldots, \phi_n(s, t)), \quad \text{with}$$
$$\Phi(0, 0)=p. \tag{6}$$

in the vicinity of p. The solution in Eq. 6 is basically a local parametrization of the surface represented by Eq. 5 at point p in which the first three coordinate functions $x_1=\phi_1(s, t)$, $x_2=\phi_2(s, t)$, and $x_3=\phi_3(s, t)$ define a *local parametrization* of the projected surface $\pi(\mathcal{F})$ at $\pi(p)$ in (x_1, x_2, x_3)-space. When the hypersurfaces f_i intersect transversally and are not singular in the vicinity of p, the surface $\mathcal{F}$ is nonsingular at p and the parametrically defined solution of Eq. 6 exists.

The parametric approximant $\Phi(s, t)$ can be of degree one, two, or more. However, there is a trade-off between the degree of the approximant, the cube size, and the difficulty of determining the cube size and the surface-edge intersections in step 2. With increasing degree of approximant a coarser mesh can be tolerated, so that fewer approximant calculations are needed. However, determining a good cube size and the intersections with the edges of the current cube becomes more difficult. To balance these factors, in this paper we use a linear parametric approximant and an adaptive spatial subdivision. A large cube size can be given initially, followed by adaptive subdivisions whenever the refinement fails or the shape geometry or topology within the cube is too complicated.

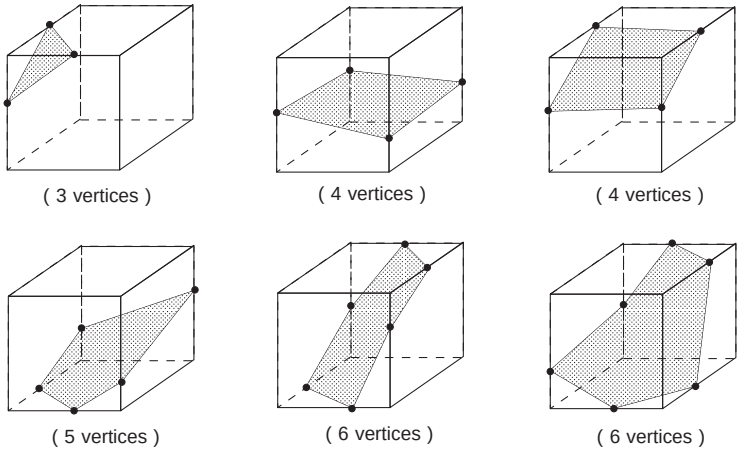
For the degree one local parametrization of $\pi(\mathcal{F})$, we find two linearly independent vectors that span the tangent space of $\mathcal{F}$ at p. That is, the approximant $\Phi(s, t)$ is obtained by solving

$$\nabla f_i(p)\cdot\mathbf{t}=0, \qquad i=1, 2,\ldots, n-2, \tag{7}$$

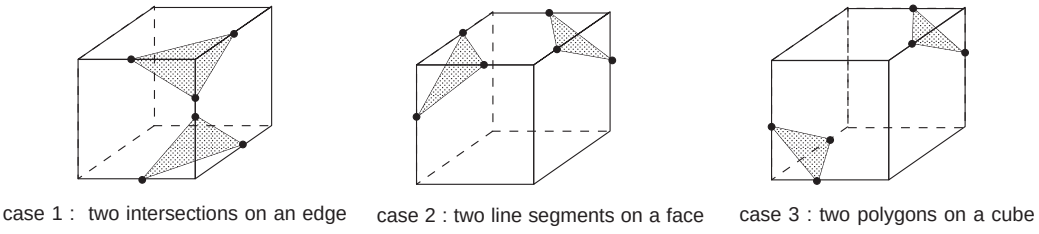
in which the solution set is of dimension two and has two linearly independent basis vectors $\mathbf{t}_1$ and $\mathbf{t}_2$, provided that p is nonsingular. Since $\mathbf{t}_1=\frac{\partial\Phi}{\partial s}(0, 0)$ and $\mathbf{t}_2=\frac{\partial\Phi}{\partial s}(0, 0)$, the linear approximant $\Phi(s, t)$ can be obtained easily.

3.2 Adaptive spatial subdivision

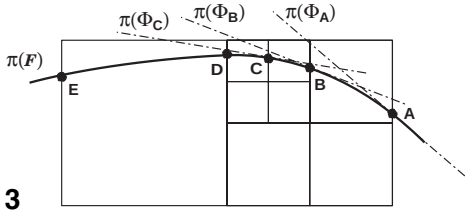
As noted before, adaptive spatial subdivision is incorporated into our surface-tracing method. This



1



2



3

Fig. 1. Successful cases of the polygon labeling

Fig. 2. Failures in polygon labeling

Fig. 3. Unnecessary subdivision caused by poor positional relationship

means that a large cube can be given initially, and the algorithm will determine adaptively how to subdivide it when a refinement fails or if the geometry or topology of the surface is too complicated within the cube.

To respond to a complicate surface topology within a cube, we check the following conditions:

1. There is at most one intersection with each edge of the cube.
2. There is at most one line segment on each face of the cube.
3. There is at most one polygon in each cube.

Violating any of these conditions in a cube would result in failures when labeling the surface-edge intersections for forming a polygon, and hence would require further subdivision on the cube. Examples of successful polygon labeling are shown in Fig. 1, and some failures are shown in Fig. 2.

Moreover, the surface geometry approximated by a polygon within a cube must be sufficiently planar. If not, further subdivisions are required. This condition is checked by computing the normal differences at adjacent polygon vertices after a polygon has been labeled successfully.

An adaptive spatial subdivision leads to a simpler and more robust structure for surface tracing. Therefore, surfaces that have complicated positional relations with the cubes or have higher curvature can be handled efficiently and robustly.

However, subdivision is not inexpensive, and therefore subdivision is delayed until necessary.
The intersection of $\pi(\Phi(s, t))$ with an edge e is assumed to be refined to a surface point on e. This simplifies choosing edges in the algorithm. If the intersection refinement fails, then the cube must be subdivided. As shown in Fig. 3, in the tracing of the surface from point A to point D, $\pi(\mathcal{F})$ does not intersect the edge that is intersected by $\pi(\Phi(s, t))$. This indicates a curvature greater than can be accommodated by the current cube size. After the first subdivision, tracing from B to D, we encounter the same problem, although locally the shape of $\pi(\mathcal{F})$ appears to be sufficiently planar. Thus, a second subdivision is needed. Thereafter, tracing can progress to D successfully. Note that more subdivisions are needed when D is close to an edge's end point.
Subdivision cannot guarantee a successful point refinement in every case. A strategy we have found to be effective for avoiding unnecessary subdivision is to delay the subdivision and try to refine the intersection point from adjacent cubes. For example, D can be traced from E as shown in Fig. 3. Thus, cube subdivision due to the failure of point refinement or polygon labeling can be delayed until every refinement originating from each adjacent cube fails.

3.3 Refinement of approximation points

Let $P_1(x_1, x_2, x_3)=0$ and $P_2(x_1, x_2, x_3)=0$ be the two face planes of a cube C whose intersection contains the cube edge e. A surface point (x_1, x_2, x_3) of $\pi(\mathcal{F})$ on e, if there is one, together with other variables, including its foot point(s) on the basis surface(s), $x_4, \ldots, x_n$ satisfies the following system of n equations in n variables:

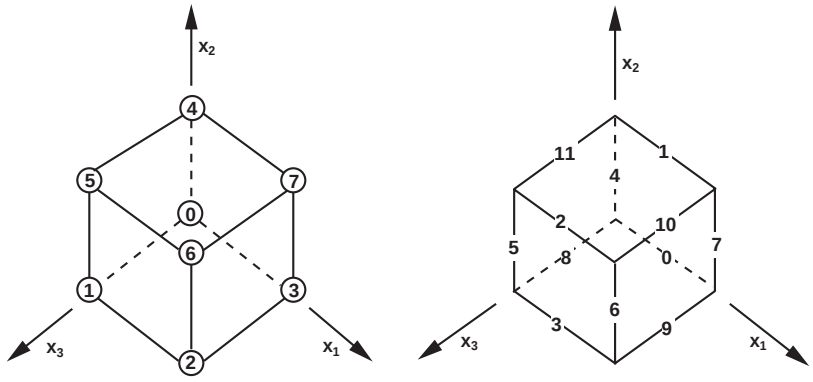
$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_{n-2}(x_1, x_2, \ldots, x_n) &= 0 \\ P_1(x_1, x_2, x_3) &= 0 \\ P_2(x_1, x_2, x_3) &= 0. \end{aligned} \tag{8}$$

We compute the intersection point of $\pi(\mathcal{F})$ on edge e by applying a Newton iteration to Eq. 8 using the intersections of $\pi(\Phi(s, t))$ with e as an initial point. That is, when $\pi(\Phi(s_0, t_0))$ is a point on edge e of C, a Newton iteration is applied to Eq. 8 with $\Phi(s_0, t_0) \in \mathbf{R}^n$ as an initial point. The natural projection of the refined point p to (x_1, x_2, x_3)-space, say $\pi(p)$, is the refined surface point of $\pi(\mathcal{F})$ on edge e.

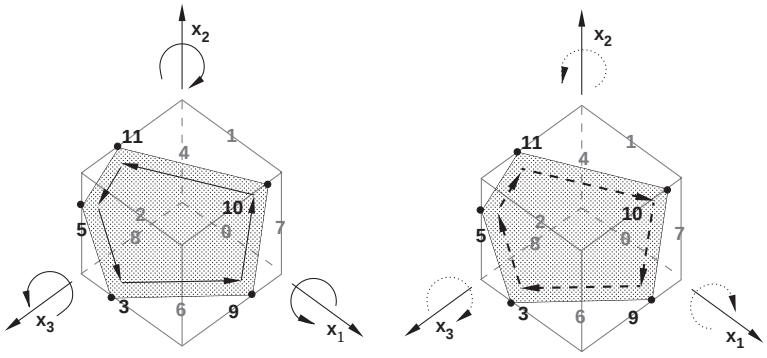
3.4 Polygon labeling and normal computations

The access of edges and cubes is a fundamental operation of the algorithm. To facilitate implementation, we need a way to index the corners and edges of a cube. Assuming the cube is in the first octant with one of its corners the origin, the vertices are indexed as shown in Fig. 4.
We enumerate edges of the same direction clockwise toward the positive axis. Edges in different directions are enumerated sequentially in the order of the x_1, x_2, and x_3 axes. The indexing of edges is shown in Fig. 4. With this indexing schema, the relationships between edges and vertices, and the subcubes can be derived easily, and operations such as polygon labeling can be facilitated.

Polygon labeling. After the intersections of $\pi(\mathcal{F})$ with edges of a cube have been found, we must check to see whether our assumptions are valid and that there are enough intersection points to form a polygon. This check is done by polygon labeling. In Bloomenthal (1988), Lorensen and Cline (1987), and Wyvill and McPheeters (1986), the polygon are labeled according to the vertex evaluation of the implicit surface function, volume data, or scalar field data. However, such vertex evaluation cannot be performed on surfaces defined in high-dimensional space: the values are vectors, not real numbers. Thus, we develop a polygon labeling strategy based on marching. We begin with an edge on which we have found a surface intersection, and then move from one face to another. For such an edge, there are two faces in the two labeling directions, described as *right spin* and *left spin*. In each direction, there are three candidate edges that might contain the next point. For an edge parallel to axis x_i, viewing into the positive direction of axis x_i, the right-spin labeling visits faces clockwise and the left-spin, counterclockwise. These candidate edges are listed in Table 1 for both labeling directions. We may start labeling



4

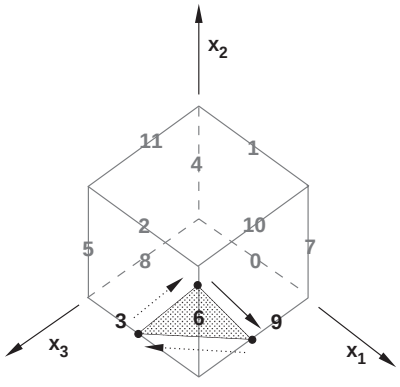


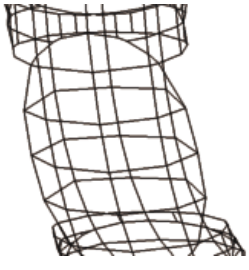
right–spin labeling

left–spin labeling

5a

5b





6

7a

7b

Fig. 4. Indexing of corners and edges

Fig. 5a, b. Polygon labeling in only one direction: **a** right-spin labeling; **b** left-spin labeling. The *solid arrows* represent connections for the right-spin system and the *dashed arrows* represent connections for the left-spin system

Fig. 6. Polygon labeling in both directions

Fig. 7a, b. Improved approximation due to difference checking on normals

Table 1. Edge reference table for polygon labeling

Row	Edge											
	0	1	2	3	4	5	6	7	8	9	10	11
Right-spin table												
1	1	2	3	0	5	6	7	4	9	10	11	8
2	4	11	6	9	8	3	10	1	0	7	2	5
3	7	10	5	8	11	2	9	0	3	6	1	4
Left-Spin Table												
1	3	0	1	2	7	4	5	6	11	8	9	10
2	8	7	10	5	0	11	2	9	4	3	6	1
3	9	4	11	6	1	8	3	10	5	0	7	2

in either direction. If the search of next intersection point reaches the starting point and our assumptions are not violated, then a polygon has been labeled successfully.

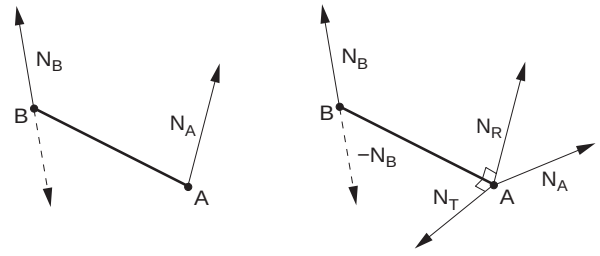
The procedure-based methods used by Bloomenthal (1988) and Wyvil and McPheeters (1986) direct the polygon labeling in only one way. With the table-driven labeling, only one table, either right spin or left spin, suffices for most cases. Two tables, however, might be needed for some of the cases shown below. Figure 5 shows examples of successful polygon labeling and the sequence in which the polygon vertices are discovered, for both directions. When the candidate edge in the third row of each table is chosen, special care must be exercised in the search of the next point since the current labeling direction may fail to identify a correct face. As shown in Fig. 6, we begin with edge 6, and connect to edge 9 by the right-spin direction. According to the right-spin system, the edge next to edge 9 is again edge 6, but we should visit edge 3 instead. To cope with this case, we switch the direction to the left-spin system in search of the edge next to edge 9. We continue using the left-spin system until an edge in the third row of the table for the left-spin system is selected. Recall that we delay subdivision due to refinement failures for efficiency reasons, as described in Sect. 3.2. It is possible to label successfully polygons that approximate surface portions of high curvature and thereby lose some details of the surface. Such cases can be detected by checking the normal differences at polygon vertices. If the normals at two adjacent vertices differ more than a user-defined angle θ, then the cube is subdivided. Figure 7a shows the result without difference checking on normals, and Figure 7b depicts the result with normal-difference checking after a successful labeling. Note that the details of the surface are better preserved and that a better approximation is obtained.

Determining surface normals. When shading a polygon, we usually need the surface normals at the vertices. The normal at a surface point $\pi(\Phi(s, t))$ can be computed as the cross product of the two tangent vectors $\partial\pi(\Phi(s, t))/\partial s$ and $\partial\pi(\Phi(s, t))/\partial t$, where $\Phi(s, t)$ is the local parametric approximation. Unfortunately, this approach cannot be used because the values of s and t are unknown at the refined surface point $\pi(p)$. Instead, the geometric relation between surface points and their foot points can be used to compute surface normals for geometrically constrained surfaces.

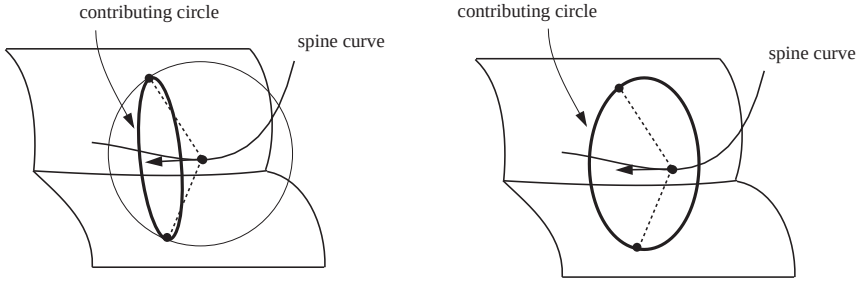
For the offset surfaces, the normal at the offset point P is along the vector $\overrightarrow{FP}$, where F is the foot point of P. For Voronoi (equidistant) surfaces, the normal at the surface point P is along the vector $\overrightarrow{FG}$ (or $\overrightarrow{GF}$), where F and G are the two foot points. For radius blending surfaces, the normal at the surface point P is along the vector $\overrightarrow{CP}$, where C is the center of the blending sphere. For normals so derived, we need to determine their orientation for a correct surface shading. The directions of surface normals must be consistent with all adjacent polygons. Suppose points A and B are connected by an edge and the direction of normal N_A at A has been fixed. If the user-defined normal-difference angle θ is small enough, we can take the normal N_A as the reference normal and determine the direction of normal N_B at point B; see Fig. 8a. Otherwise, we determine the direction of N_B with the direction of N_R as the reference normal, where $N_R = \overrightarrow{BA} \times N_T$ and $N_T = N_A \times \overrightarrow{BA}$. See also Fig. 8b.

3.5 Surface clipping for radius blends

Radius blends are popular because they have an intuitive geometric description. Nevertheless, the mathematical details have some subtle difficulties. A radius blend can be conceptually described as the envelope of a rolling sphere centering on a *spine curve* that lies on a surface bisecting two base surfaces f and g. The sphere at each point on the spine curve has radius the distance of the point from the base surfaces. The bisecting or equi-

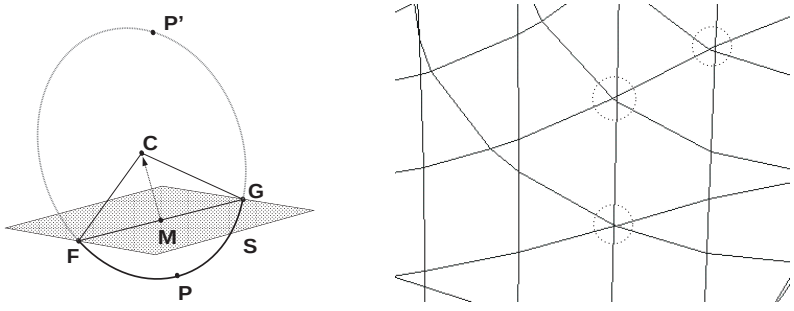


8a 8b



Spherical blend Circular blend

9a 9b



10 11

Fig. 8a, b. Determining normal directions

Fig. 9a, b. Contributing circle for the variable-radius blend: **a** spherical blend; **b** circular blend

Fig. 10. Clipping the radius blend

Fig. 11. Tiny triangles (marked by *dashed circles*) in the nonuniform polygonization

distant surface of f and g is the intersection of offsets with varying radius from each base surface in which the radius is itself a variable. The radius of the rolling sphere can be *constant* or *variable* along the spine curve. The spine curve of a constant-radius blend is the intersection of two offset surfaces with the same fixed offset while the spine curve of a variable-radius blend can be obtained by intersecting the equidistant surface with a reference surface (Hoffmann 1990) or be constrained

on the equidistant surface with certain geometry constraints (Chuang and Hwang 1997; Chuang et al. 1995).

Consider the spine curve defined by a reference surface h. The variable-radius blend of f and g is represented by the following 11 equations with 13 variables $x, y, z, u, v, w, u_1, v_1, w_1, u_2, v_2, w_2$, and r:

$$(u-u_1)^2+(v-v_1)^2+(w-w_1)^2-r^2=0 \quad (9)$$

$$f(u_1,v_1,w_1)=0 \quad (10)$$

$$-f_{v_1}(x-u_1)+f_{u_1}(y-v_1)=0 \quad (11)$$

$$-f_{w_1}(y-v_1)+f_{v_1}(z-w_1)=0 \quad (12)$$

$$(u-u_2)^2+(v-v_2)^2+(w-w_2)^2-r^2=0 \quad (13)$$

$$g(u_2,v_2,w_2)=0 \quad (14)$$

$$-g_{v_2}(x-u_2)+g_{u_2}(y-v_2)=0 \quad (15)$$

$$-g_{w_2}(y-v_2)+g_{v_2}(z-w_2)=0 \quad (16)$$

$$h(u,v,w)=0 \quad (17)$$

$$S_d:(x-u)^2+(y-v)^2+(z-w)^2-r^2=0 \quad (18)$$

$$\left(\frac{\partial S_d}{\partial u},\frac{\partial S_d}{\partial v},\frac{\partial S_d}{\partial w}\right)\cdot(N_h\times N_B)=0, \quad (19)$$

where N_h is the normal of h at (u, v, w) and N_B is the normal of the bisecting surface of f and g at (u, v, w). The cross product $N_h \times N_B$ gives the tangent direction of the spine curve at (u, v, w). Equations 9–12 and 13–16 represent the offsets of f=0 and g=0, respectively, with an unknown radius r. Since r itself is a variable, the intersection of r-offsets of f=0 and g=0 represents points equidistant to f=0 and g=0. Equation 17 represents a reference surface whose intersection with the equidistant surface forms a spine curve for the variable-radius blend between f=0 and g=0. Equation 18 represents the sphere S_d centered on the spine curve, and Eq. 19 constrains points on S_d to the envelope of the rolling spheres. The envelope of the rolling spheres is the projection of the solution set to the (x, y, z)-space. Each solution point of the system represents a spine point, foot points of the spine point on each of the base surfaces, and other auxiliary parameters.

On each rolling sphere, only a circle contributes to the envelope. The plane containing the circle must be perpendicular to the tangent of the spine curve at the center of the sphere, and must contain the foot points (Fig. 9). Moreover, with a radius blend the surface area of interest is bounded by contact curves on the base surfaces with which the rolling sphere maintains contact. That is, on each contributing circle only a segment bounded by the foot points is of interest. We introduce a mechanism that translates the bound on the circular segment into an equation.

Assume that points F and G are the foot points of the center C of the sphere on base surfaces f and g, respectively, and that M is the midpoint of $\overline{FG}$. Let $S(x, y, z)$=0 be the plane passing through F and G and normal to the vector $\overrightarrow{MC}$; as depicted in Fig. 10. The plane S separates the circle into two segments, one contains a point P on the blend and the other contains a point P' that is not on the blending surface. Assuming for simplicity that the segment of interest does not exceed a half circle, the segment of interest can be constrained by

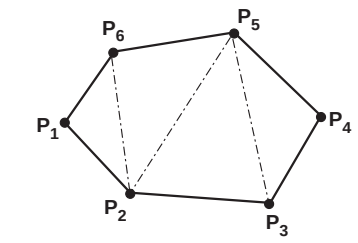
$$S(x, y, z)\, S(u, v, w)=a, \quad (20)$$

where a is a negative number.

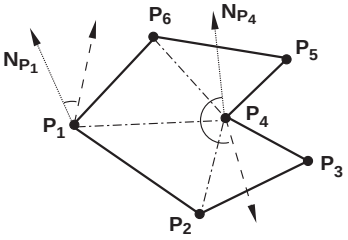
To derive the trimmed polygonal mesh for a radius blend, we first augment to the system of Eqs. 9–19 with Eq. 20, where a is an auxiliary variable. During the polygonization process we reject polygons that have vertices with nonnegative a. We clip every polygon that has vertices with both negative and nonnegative a. Finally, we label polygons with negative a as before. To clip a polygon, we label the polygon as usual, but derive the boundary points whenever a pair of vertices with both negative and nonnegative a is detected. That is, beginning with a vertex P_0 with a negative a, we do the labeling as usual until a vertex P_i, $i>0$, with a nonnegative a is reached. Instead of labeling P_i we derive boundary point P_i^* and regard it as the vertex next to P_{i-1}. Then the labeling proceeds, but without connecting edges, until a vertex P_j, $j>i$, with a negative a is found. We derive the boundary point P_j^* and connect it to P_i^*. To finish the polygon labeling, the labeling proceeds as usual until P_0 is reached.

4 Decimation and refinement of the polygonal mesh

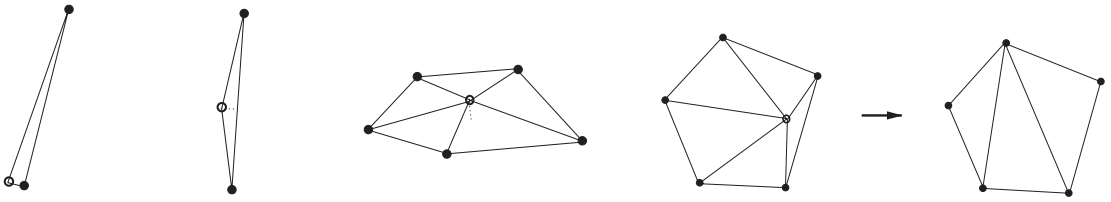
In Sect. 3, we have presented our algorithm for the adaptive polygonization of geometrically con-



12a



12b



13a 13b 13c 14

Fig. 12a, b. Triangulation of polygons: **a** convex; **b** concave

Fig. 13a–c. Vertices that should be removed

Fig. 14. Retriangulation after a vertex is removed

strained surfaces. Using the algorithm, we are able to obtain the polygonal meshes of constrained surfaces, including offset, Voronoi, and blending surfaces. A finer polygonization can be obtained by using tighter normal-difference values for polygon labeling or by using smaller grid size. However, the cost of surface tracing with spatial subdivision grows rapidly if these parameters become too tight. Moreover, the polygonization can be nonuniform due to poor positional relations between surface and the space grids. As shown in Fig. 11, such poor positional relations may result in tiny triangles. A more cost effective approach is to first obtain a crude but acceptable polygonal approximation and then apply decimation, retriangulation, and surface refinement to derive a very good quality polygonization of the original surface. The framework is as follows:

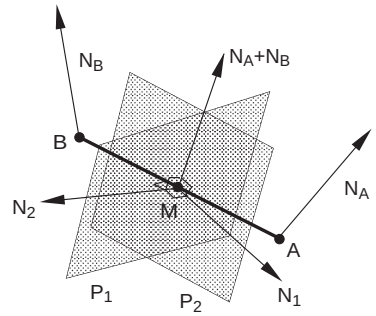
1. Compute a crude but acceptable polygonal mesh for the given constrained surface.
2. Triangulate the polygonal mesh.
3. Decimate the triangular meshes by eliminating negligible points and retriangulating the resulting polygons.
4. Refine locally the decimated triangular meshes.

Steps 3 and 4 are repeated until no new negligible points are found or no polygon requires local refinement.

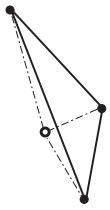
4.1 Triangulating a polygon

We can triangulate a polygon by repeatedly adding the shortest diagonal between vertices as a splitting edge, until there are only triangles. An example is shown in Fig. 12: The diagonal $\overline{P_2P_6}$ is connected first, and then $\overline{P_3P_5}$ followed by $\overline{P_2P_5}$.

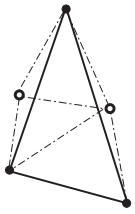
When there are concave vertices, we must make sure that an interior diagonal is selected. (Note that we have nonplanar polygons, so the definition of convex and concave vertex includes consideration



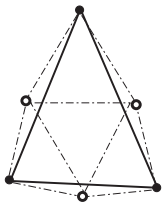
15



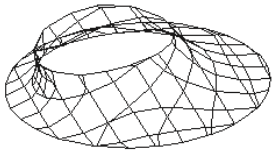
16a



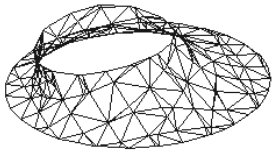
16b



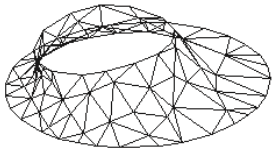
16c



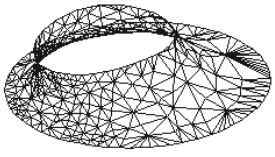
17a Original polygonal mesh (164 polygons).



17b Triangular mesh (339 triangles).



17c Triangular mesh after decimation (145 triangles).



17d Triangular mesh after refinement (604 triangles).

Fig. 15. Refining an edge

Fig. 16a–c. Refining a triangle

Fig. 17a–d. Decimation, triangulation, and local refinement

of the surface normal at the vertex.) In Fig. 12b, the shortest diagonal $\overline{P_3P_5}$ creates the inadmissible triangle $P_3P_4P_5$. To triangulate a concave polygon, we first find a concave vertex P_i that is adjacent to a convex vertex P_{i-1}. Then we connect P_i with a vertex P_j, $j \neq i$, where the angle between the surface normal at P_i and the cross product $\left(\overrightarrow{P_jP_i} \times \overrightarrow{P_{i-1}P_i}\right)$ is an acute angle and the diagonal $\overline{P_iP_j}$ is shortest. For the example shown in Fig. 12b, vertex P_4 is a concave vertex. Here, we connect the shortest valid diagonal $\overline{P_4P_2}$, and then $\overline{P_4P_1}$ followed by $\overline{P_4P_6}$.

4.2 Decimating the triangular meshes

A polygonal mesh can be reduced by removing negligible vertices or edges (Schroeder et al. 1992). That is, a polygonal vertex is removed if it is (1) too close to an adjacent vertex, (2) too close to an edge, or (3) too close to a plane (Fig. 13). When a vertex v is removed, we eliminate the edges incident to v and then triangulate the resulting polygon; see the example shown in Fig. 14. The vertex removal is repeated until no vertex can be removed for the given distance tolerances.

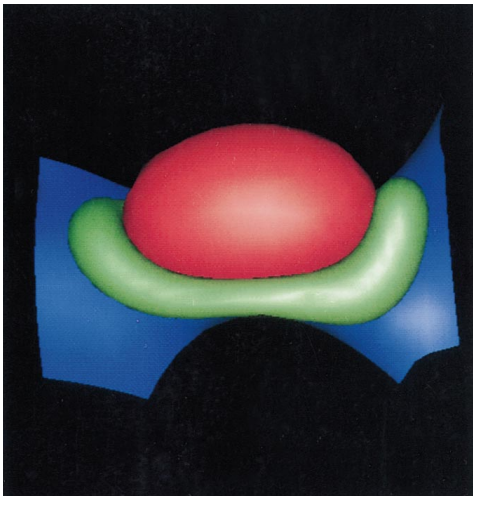
The elimination of vertices on the surface boundary is special because it could significantly change the boundary. We can preserve all the boundary

Fig. 18. The offset surface and Voronoi surface. **a** the offset surface of a bicubic parametric surface; **b** the Voronoi surface of an ellipsoid and a bicubic surface ▶

Fig. 19a–d. The variable-radius blend of an ellipsoid and a bicubic surface

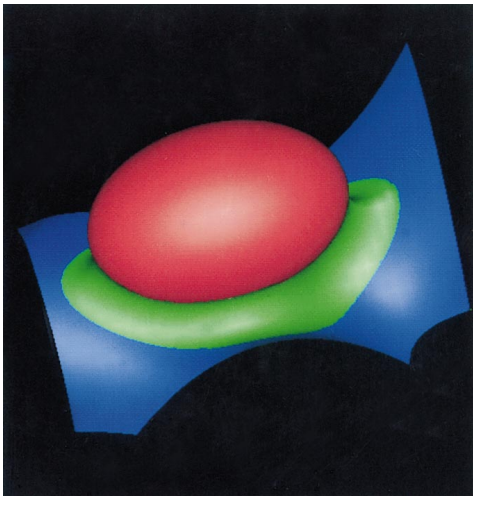
18a

18b



19a

19b



19c

19d

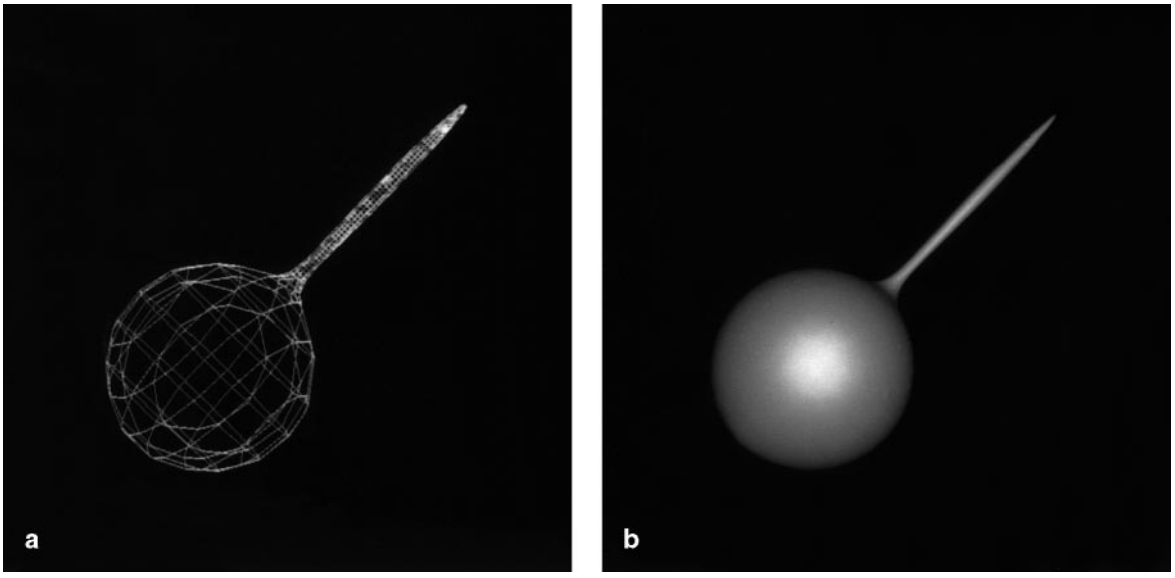


Fig. 20a, b. The implicit surface $(200\,x^2+y^2+200\,z^2-1)*(x^2+(y-2.5)^2+z^2-1)-1=0$: **a** polygonal mesh; **b** shaded image

vertices or only those vertices at which the incident edges form an acute angle. Polygons obtained by decimation can be retriangulated with the method previously described or with the loop-splitting procedure in Schroeder (1992).

4.3 Local refinement of triangular meshes

To obtain a better approximation, we next refine triangle meshes locally according to some user-defined criteria. For each edge of a triangle, we refine the edge if the surface normals at edge's endpoints differ more than a given tolerance δ. See Fig. 15 for the quantities involved in edge refinement. Let line L be the line passing through the midpoint M of edge $\overline{BA}$ and oriented in the direction (N_A+N_B). Point M is refined to the surface $\pi(\mathcal{F})$ along L with the system of Eq. 8, where the planes P_1 and P_2 contain L. The plane normals N_1 and N_2 are $N_1 = (N_A + N_B) \times \overrightarrow{BA}$ and $N_2=(N_A+N_B)\times N_1$ (Fig. 15). After edge refinement, a triangle can be replaced by two, three, or four triangles, as shown in Fig. 16. Figure 17 illustrates decimation, triangulation, and local refinement applied to the polygonal mesh of a variable-radius blend. Notice that the boundary of the radius blend is refined with a stricter tolerance and thus has greater precision.

5 Implementation and examples

The proposed algorithm has been implemented in C++ on a SGI Indigo2 High Impact with a MIPS R4400 CPU and 64 MB RAM. Many examples have been tested. Figure 18a shows the offset surface of a bicubic parametric surface, and Fig. 18b depicts the Voronoi surface of an ellipsoid and a bicubic surface. Figure 19a and b shows the variable-radius blend of an ellipsoid and a bicubic surface without surface clipping. Figure 19c and d shows the images with surface clipping. Figure 20a and b shows the polygonal mesh and its shading image of implicit surface $(200\,x^2+y^2+200\,z^2-1)*(x^2+(y-2.5)^2+z^2-1)-1=0$. The thin pin of this surface is finely polygonized with our adaptive spatial subdivision. Table 2 shows the timing data, then the polygon number and the triangle number after the polygonization and decimation, respec-

Table 2. Execution time and mesh size

Example	Mesh generation		Decimation	
	Time in seconds	Polygon number	Time in seconds	Triangle number
Offset	15.04	201	1.61	146
Voronoi	92.85	293	0.64	159
Blend	518.27	685	364.73	1479

tively. We note that with decimation and local refinement as postprocessing, the cube size for polygonization can be larger initially. Hence the expensive polygonization can be supplemented by the cheaper decimation and refinement.

6 Summary and conclusions

The appeal of geometrically constrained surfaces is their ability to represent, uniformly, surfaces that are defined from given surfaces subject to certain conditions. The given surfaces could be implicit, parametric, or again constrained surfaces. Polygonizing such surfaces is essential to working with them in an interactive design environment.

We have presented an algorithm that computes a polygonal mesh approximating a geometrically constrained surface $\mathcal{F}$ defined in $\mathbf{R}^n$, $n \geq 3$, but performs all major computations in 3-space by tracing its projection $\pi(\mathcal{F})$ with help of a grid of cubes. The approach is made possible by local parametric approximations of $\mathcal{F}$. The approximations establish the connection between the surface points in n-space and their projections into 3-space.

An adaptive spatial subdivision scheme has been proposed that, driven by a few user-supplied parameters, can refine the surface approximation completely automatically in a large initial domain. Polygon mesh refinement takes place adaptively whenever the intersection point refinement fails or when the surface geometry or topology within a cube is too complicated. Thus the user need not know, a priori, where the surface is smooth and where it has areas of great detail.

Moreover, we have developed decimation and local refinement heuristics for postprocessing. These heuristics can speed up surface approximation because they do not require a grid that is uniformly of small cube size, even in areas of low surface complexity. Thus, the need of deriving a finer polygonal mesh solely with grid subdivision is avoided. The method has been implemented and is capable of evaluating and rendering some of the most difficult surfaces arising in geometric modeling.

Acknowledgements. The work of J.-H. Chuang, K.-M. Ko and W.-C. Hwang was supported by the National Science Council (NSC) of the Republic of China under Grants NSC 81-0408-E-009-533 and NSC 85-22/3-E-009-118.

References

Allgower EL, Gnutzmann S (1987) An algorithm for piecewise linear approximation of implicitly defined two-dimensional surfaces. SIAM J Numer Anal 24:452–469

Allgower EL, Gnutzmann S (1991) Simplicial pivoting for mesh generation of implicitly defined surfaces. Comput Aided Geom Des 8:305–325

Bajaj CL, Hoffmann CM, Lynch RE, Hopcroft JEH (1988) Tracing surface intersections. Comput Aided Geom Des 5:285–307

Barnhill RE, Farin G, Piper BR (1987) Surface/surface intersection. Comput Aided Geom Des 4:3–16

Bloomenthal J (1988) Polygonization of implicit surfaces. Comput Aided Geom Des 5:341–355

Chuang JH, Hoffmann CM (1992) Curvature computations on surfaces in n-space. Math Modelling Numer Anal 26:95–112

Chuang JH, Hwang WC (1997) Variable-radius blending by constrained spine generation. Visual Comput 13:316–329

Chuang JH, Lin CH, Hwang WC (1995) Variable-radius blending of parametric surfaces. Visual Comput 11:513–525

Filip DJ (1989) Blending parametric surfaces. ACM Trans Graph 8:164–173

Hall M, Warren J (1990) Adaptive polygonization of implicitly defined surfaces. IEEE Comput Graph Appl 10:33–42

Hoffmann CM (1990) A dimensionality paradigm for surface interrogations. Comput Aided Geom Des 7:517–532

Hoffmann CM, Hopcroft JEH (1987) The potential method for blending surfaces and corners. In: Farin G (ed) Geometric modeling: algorithms and new trends. SIAM, Philadelphia, USA, pp 347–365

Hoffmann CM, Vermeer PJ (1991) Eliminating extraneous solutions in curve and surface operations. Int J Comput Geom Appl 1:47–66

Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W (1993) Mesh optimization. Proceedings of ACM SIGGRAPH'93, Anaheim, CA, USA, pp 19–26

Lane JM, Riesenfeld RF (1980) A theoretical development for the computer generation and display of piecewise polynomial surfaces. IEEE Trans Patt Anal Mach Int 2:35–46

Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. Comput Graph 21:163–169

Müller H, Stark M (1993) Adaptive generation of surfaces in volume data. Visual Comput 9:182–199

Neilson GM, Hamann B (1991) The asymptotic decider: resolving the ambiguity in marching cubes. In: Nielson GM, Rosenblum L (eds) Proceedings of Visualization '91. IEEE Computer Society Press, Washington, pp 83–90

Ning P, Bloomenthal J (1993) An evaluation of implicit surface tilers. IEEE Comput Graph Appl 13:33–41

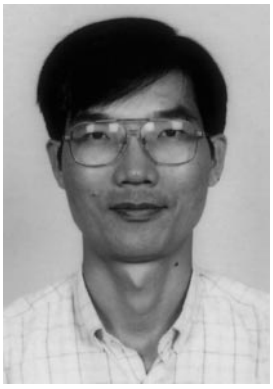
Rheinboldt WC (1987) On a moving-frame algorithm and the triangulation of equilibrium manifolds. In: Kupper T, Seydel R, Troger H (eds) Bifurcation: analysis, algorithms, applications, Brikhauser, Basel, pp 256–267

Rossignac JR, Requicha AAG (1986) Offsetting operations in solid modeling. Comput Aided Geom Des 3:129–148

Schroeder WJ, Zarge JA, Lorensen WE (1992) Decimation of triangle meshes. Comput Graph (Proceedings of Siggraph), 26:65–70

Wilhelms J, Van Gelder A (1990) Topological considerations in isosurface generation. Comput Graph 24:79–86

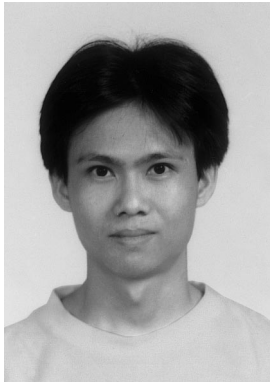
Wilhelms J, Van Gelder A (1992) Octrees for faster isosurface generation. ACM Trans Graph 11:201–227
Wyvill G, McPheeters C, Wyvill B (1986) Data structure for soft objects. Visual Comput 2:227–234



JUNG-HONG CHUANG is an Associate Professor of Computer Science and Information Engineering at National Chiao Trung University, Taiwan, ROC. His research interests include geometric and solid modeling, computer graphics, and visualization. Dr. Chuang received his BS degree in Applied Mathematics from National Chiao Tung University, Taiwan, in 1978, and MS and PhD degrees in Computer Science from Purdue University in 1987 and 1990, respectively.

CHRISTOPH M. HOFFMANN is Professor of Computer Science at Purdue University in the US. His research focuses on geometric and solid modeling and its many applications in design for manufacturing. He investigates high-level design representations and their compilation to CAD systems. This work includes research on feature-based, constraint-based generative design, and the frameworks and concepts needed to effectively implement such design paradigms, as well as on distributed CAD architectures and techniques to integrate CAD systems with downstream software and processes.

KUN-MING KO is currently a Senior Engineer at Taiwan Semiconductor Manufacturing Company (TSMC), Taiwan, ROC. His research interests include geometric modeling and computer network. Mr. Ko received his BS and MS degrees in Computer Science and Information Engineering from National Chiao Tung University in 1990 and 1992, respectively.



WEI-CHUNG HWANG is a PhD student in the Department of Computer Science and Information Engineering at National Chiao Tung University, Taiwan, ROC. His research interests include geometric modeling and computer graphics. Mr. Hwang received his BS and MS degrees in Computer Science and Information Engineering from National Chiao Tung University in 1990 and 1992, respectively.