BUTTERWORTH HEINEMANN

Computer-Aided Design, Vol. 27, No. 1, pp. 16–26, 1995
Copyright © 1995 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0010–4485/95 $10.00+0.00

Research

Tessellating trimmed NURBS surfaces

Leslie A Piegl and Arnaud M Richard

An algorithm for obtaining a piecewise planar approximation of a trimmed NURBS surface is presented. Given a model space tolerance ε, the algorithm triangulates the parameter space domain of the trimmed surface such that the 3D planar approximation, obtained by mapping 2D triangles onto the surface, deviates from the trimmed surface by no more than ε. The number of triangles computed in parameter space depends on the bounds of the second derivatives. A detailed discussion of the algorithm and a practical error analysis of the tessellation are provided.

Keywords: NURBS, tessellation, visualization

Trimmed surfaces have a fundamental role in computer-aided design[1–4]. Most complex objects are generated by some sort of trimming/scissoring process, i.e. unwanted parts of the rectangular patch are trimmed away. Trimmed patches are also the result of Boolean operations on solid objects bounded by NURBS surfaces. In the computer-aided design pipeline, the trimmed patch undergoes a number of processes such as rendering for visualization[5–8], cutter path generation, area computation, or rapid prototyping, also known as 'solid hard copy'[9]. Probably one of the easiest ways to accomplish all this is by approximating the trimmed patch by triangular facets to within a user given tolerance. This method has a number of advantages:

- It is not sensitive to the complexity of the trimmed patch, i.e. the number of holes and trimmings along the boundaries do not complicate the geometry processing routines.
- It results in a unique database, i.e. the same triangular irregular network (TIN) can be used to render the patch as well as to compute, say, cutter paths or silhouette lines.
- Algorithms that operate on triangles are far easier and numerically more stable than those dealing with freeform geometry.
- The piecewise triangular approximation is a parameterization independent representation of the trimmed surface, i.e. very general algorithms can be written to process its geometry. (Of course, the triangulation of the trimmed surface depends on the parameterization. However, all triangulations are equivalent to within the user specified tolerance.)

The two major disadvantages are that

- adequate representation of a trimmed patch with high curvature areas requires large numbers of triangles,
- the triangulation, if not done properly, can result in triangles of different sizes, and, in particular, in long and skinny triangles which, in turn, can cause numerical problems.

In this paper we present a parameter space-driven tessellation method. The algorithm triangulates the trimmed parametric region such that the triangles mapped onto the surface form a piecewise triangular approximation to within a user specified tolerance. The parameter space is not split into regions representing Bézier patches; rather it is triangulated as a whole. In the second section a short definition of trimmed NURBS surfaces is given. The third section provides the details of each part of the algorithm. The fourth section presents several examples and numerical tests, and a conclusions section closes the paper.

TRIMMED NURBS SURFACES

A trimmed NURBS surface consists of two things: (a) a tensor product NURBS surface (see *Figure 1*), and (b) a set of properly ordered trimming curves lying within the parameter rectangle of the surface (see *Figure 2*). A degree (p, q) NURBS surface has the form

$$S(u, v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} P_{i,j} N_{i,p}(u) N_{j,q}(v)}{\sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} N_{i,p}(u) N_{j,q}(v)} \qquad (1)$$

where $w_{i,j}$ are the weights, $P_{i,j}$ are the control points, and

Department of Computer Science & Engineering, University of South Florida, ENG 118, 4202 Fowler Avenue, Tampa, FL 33620, USA
A M Richard is on leave from Renault Direction de la Recherche, France
Paper received: 9 September 1993. Revised: 1 December 1994

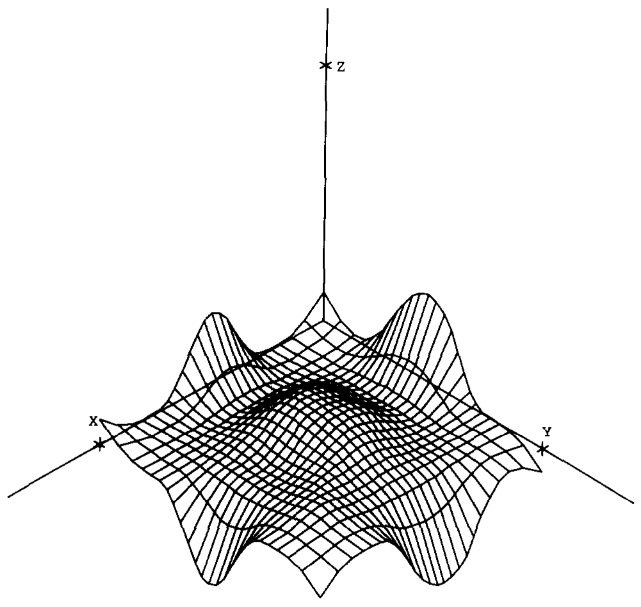


Figure 1 Bicubic NURBS surface

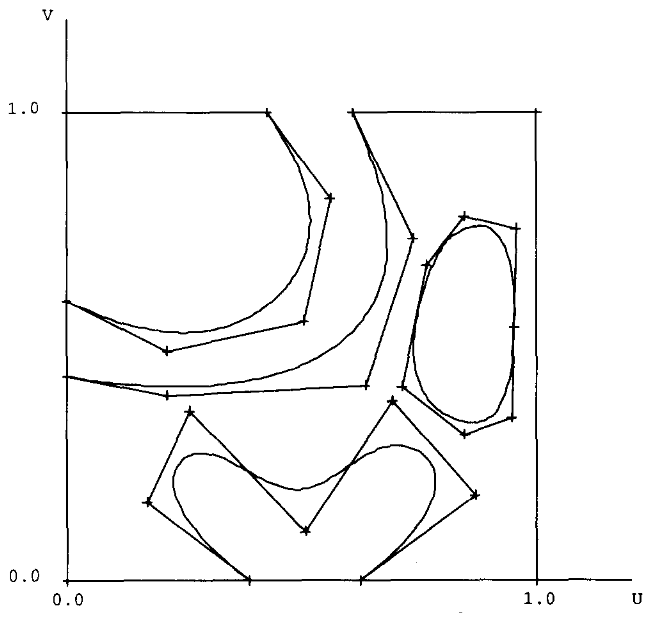


Figure 2 Trimmed domain using cubic NURBS curve

$N_{i,p}(u)$ and $N_{j,q}(v)$ are the degree p and q B-splines, respectively, defined over the knot vectors

$$U=\{\underbrace{a,a,\ldots,a}_{p+1},\ u_{p+1},\ldots,\ u_{r-p-1},\ \underbrace{b,b,\ldots,b}_{p+1}\}$$

$$V=\{\underbrace{c,c,\ldots,c}_{q+1},\ v_{q+1},\ldots,\ v_{s-q-1},\ \underbrace{d,d,\ldots,d}_{q+1}\}$$

The trimming curves can be of any form. However, when dealing with NURBS entities, it is desirable to represent them in NURBS form. Assume that N such curves are given defined as

$$C_k(t)=(u_k(t),\ v_k(t))=\sum_{i=0}^{n} P_i^k N_{i,l}(t) \qquad k=1,2,\ldots,N \qquad (2)$$

with the knot vectors

$$T_k=\{\underbrace{e^k,e^k,\ldots,e^k}_{l+1},\ t_{l+1}^k,\ldots,\ t_{m_k-l-1}^k,\ \underbrace{f^k,f^k,\ldots,f^k}_{l+1}\}$$
$$k=1,2,\ldots,N$$

The curves $C_k(t)$ are all properly oriented forming $M<N$ numbers of loops. (A loop forms a boundary of the trimmed region such that, when marching along the piecewise curve as indicated by its direction, the valid surface material is always on the same side. *Figure 2* has $M=3$ loops.) The trimmed surface boundaries are then obtained by mapping the 2D trimming curves onto the surface. That is,

$$S(u_k(t),\ v_k(t)) \qquad k=1,2,\ldots,N$$

are surface curves bounding the trimmed surface. For further details on NURBS, see Reference 10.

TESSELLATION

The tessellation method presented in this paper consists of the following steps:

- *Step 1:* compute the longest edge size in the parameter domain.
- *Step 2:* Obtain a polygonal approximation of trimming curves.
- *Step 3:* Select points inside the valid region.
- *Step 4:* Triangulate the trimmed region.
- *Step 5:* Map the triangles onto the surface and build a 3D triangular database for further processing.

In the following sections we elaborate on the details of each step.

Computing the longest edge size

The main idea, which has long been known to both the graphics[11,12] and CAD communities[9,13,14], is simply this: obtain an edge size λ such that, when triangulating the domain with triangles of sides less than λ, the 3D triangles, obtained by mapping the 2D ones onto the surface, deviate from the surface by less than ε, where ε is a user specified tolerance. The general theory of obtaining such a bound is well understood and can be summarized as follows. A triangle $\langle A,B,C\rangle$ in the parameter space domain with edge length less than λ is given. Let $T(u,v)$ be a linearly parameterized triangle in 3D obtained by mapping $\langle A,B,C\rangle$ onto a triangular facet satisfying $T(A)=S(A)$, $T(B)=S(B)$ and $T(C)=S(C)$. The deviation between the triangular facet and the triangular surface patch satisfies[9]

$$\sup_{(u,v)\in\langle A,B,C\rangle} \|S(u,v)-T(u,v)\| \le \frac{2}{9}\lambda^2(D_1^{ABC}+2D_2^{ABC}+D_3^{ABC}) \qquad (3)$$

where

$$D_1^{ABC} = \sup_{(u,v)\in\langle A,B,C\rangle} \|S_{uu}(u,v)\| \tag{4}$$

$$D_2^{ABC} = \sup_{(u,v)\in\langle A,B,C\rangle} \|S_{uv}(u,v)\| \tag{5}$$

$$D_3^{ABC} = \sup_{(u,v)\in\langle A,B,C\rangle} \|S_{vv}(u,v)\| \tag{6}$$

That is, to obtain an edge length which is valid for every triangular facet, the upper bounds of the second derivatives, D_1, D_2 and D_3, computed over the entire patch, are needed. After these bounds are found, the edge length is

$$\lambda = 3\left(\frac{\varepsilon}{2(D_1+2D_2+D_3)}\right)^{1/2} \tag{7}$$

The bulk of the problem is to determine the bounds on the second derivatives for both rational and nonrational B-spline surfaces. Methods published so far require either constrained optimization and form conversion[13], or form conversion and knowledge of properties of special functions such as Chebyshev polynomials[9]. None of the published methods seem to offer a good solution for rational surfaces. The method we present here is very simple and numerically stable.

Let us consider nonrational surfaces first. It is well known that the second derivatives of a B-spline surface are simply B-spline surfaces. More precisely, the second derivative in the u direction is computed as

$$S_{uu}(u,v) = \sum_{i=0}^{n-2}\sum_{j=0}^{m} N_{i,p-2}(u)N_{j,q}(v)P_{i,j}^{(2,0)} \tag{8}$$

where

$$P_{i,j}^{(2,0)} = \frac{p(p-1)}{u_{i+p+1}-u_{i+2}}\left(\frac{P_{i+2,j}-P_{i+1,j}}{u_{i+p+2}-u_{i+2}} - \frac{P_{i+1,j}-P_{i,j}}{u_{i+p+1}-u_{i+1}}\right)$$
$$i=0,1,\ldots,n-2;\ j=0,1,\ldots,m \tag{9}$$

defined over the knot vectors

$$U^{(2)} = \{\underbrace{a,a,\ldots,a}_{p-1}, u_{p+1},\ldots,u_{r-p-1},\underbrace{b,b,\ldots,b}_{p-1}\}$$

$$V^{(0)} = V$$

Similarly, the second derivative in the v direction is obtained as

$$S_{vv}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m-2} N_{i,p}(u)N_{j,q-2}(v)P_{i,j}^{(0,2)} \tag{10}$$

Table 1 Change of magnitudes of maximum second derivatives during refinement

Refinement	$S_{uu}^{\max}$	$S_{vv}^{\max}$	$S_{uv}^{\max}$
0	110.6746	97.8726	49.6903
1	102.4967	93.2605	49.6903
2	102.3495	92.1086	49.6903
3	101.8226	92.0643	49.6903
4	101.6908	91.9850	49.6903

where

$$P_{i,j}^{(0,2)} = \frac{q(q-1)}{v_{j+q+1}-v_{j+2}}\left(\frac{P_{i,j+2}-P_{i,j+1}}{v_{j+q+2}-v_{j+2}} - \frac{P_{i,j+1}-P_{i,j}}{v_{j+q+1}-v_{j+1}}\right)$$
$$i=0,1,\ldots,n;\ j=0,1,\ldots,m-2 \tag{11}$$

defined over the knot vectors

$$U^{(0)} = U$$

$$V^{(2)} = \{\underbrace{c,c,\ldots,c}_{q-1}, v_{q+1},\ldots,v_{s-q-1},\underbrace{d,d,\ldots,d}_{q-1}\}$$

Finally, the mixed partials are

$$S_{uv}(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} N_{i,p-1}(u)N_{j,q-1}(v)P_{i,j}^{(1,1)} \tag{12}$$

where

$$P_{i,j}^{(1,1)} = \frac{pq}{v_{j+q+1}-v_{j+1}}\left(\frac{P_{i+1,j+1}-P_{i,j+1}-P_{i+1,j}+P_{i,j}}{u_{i+p+1}-u_{i+1}}\right)$$
$$i=0,1,\ldots,n-1;\ j=0,1,\ldots,m-1 \tag{13}$$

defined over the knot vectors

$$U^{(1)} = \{\underbrace{a,a,\ldots,a}_{p}, u_{p+1},\ldots,u_{r-p-1},\underbrace{b,b,\ldots,b}_{p}\}$$

$$V^{(1)} = \{\underbrace{c,c,\ldots,c}_{q}, v_{p+1},\ldots,v_{s-q-1},\underbrace{d,d,\ldots,d}_{q}\}$$

Because the basis functions $N_{i,p}(u)$ and $N_{j,q}(v)$ are nonnegative and form a partition of unity, the bounds on the derivatives are simply

$$\sup_{(u,v)\in[a,b]\times[c,d]} \|S_{uu}(u,v)\| \leqslant \max_{\substack{0\leqslant i\leqslant n-2\\ 0\leqslant j\leqslant m}} \|P_{i,j}^{(2,0)}\| \tag{14}$$

$$\sup_{(u,v)\in[a,b]\times[c,d]} \|S_{vv}(u,v)\| \leqslant \max_{\substack{0\leqslant i\leqslant n\\ 0\leqslant j\leqslant m-2}} \|P_{i,j}^{(0,2)}\| \tag{15}$$

$$\sup_{(u,v)\in[a,b]\times[c,d]} \|S_{uv}(u,v)\| \leqslant \max_{\substack{0\leqslant i\leqslant n-1\\ 0\leqslant j\leqslant m-1}} \|P_{i,j}^{(1,1)}\| \tag{16}$$

Upper bounds are obtained by computing the maxima of the position vectors (control points) of the derivative surfaces S_{uu}, S_{vv} and S_{uv}. These bounds can then be sharpened to any required accuracy via knot refinement. *Table 1* shows the convergence properties of the refinement process applied to the test surface shown in *Figure 1*. Note that $S_{uv}^{\max}$ did not change at all. It can happen that the maximum is at the corner of the patch in which case refinement is completely unnecessary.

One might wonder how the accuracy of the derivative bounds affects the edge size of triangles. It is well known that knot refinement is not terribly fast and requires large amounts of memory. Consequently, computing bounds on derivatives to a high accuracy using knot refinement can be time consuming. Although knot refinement can be modified (localized) in a number of different ways, it turns out, however, that it is really ucessary. The reader

Table 2 Edge length and number of triangles as functions of refinement ($\varepsilon=0.1$)

Refinement	λ	Number of triangles
0	0.027031	2049
1	0.027611	2011
2	0.027672	2007
3	0.027699	2003
4	0.027709	2003

Table 3 Edge length and number of triangles as functions of refinement ($\varepsilon=0.01$)

Refinement	λ	Number of triangles
0	0.008548	18917
1	0.008731	18211
2	0.008751	18184
3	0.008759	18156
4	0.008762	18152

can easily verify that in general the maximum derivatives are much larger than the tolerance ε. Consequently, the term

$$\frac{\varepsilon}{2(D_1+2D_2+D_3)}$$

in Equation 7 is not affected much by the refinement of the derivative surfaces. *Tables 2* and *3* show the relationship between the levels of refinement, the edge length and the number of triangles computed using the surface in *Figure 1*. Although the numerical data depends on the surface geometry (low and high curvature areas), practical experience shows that the level of refinement is not that significant. In the examples above, four refinements provided 2.2% improvement for $\varepsilon=0.1$, and 4.0% for $\varepsilon=0.01$. Both improvements are pretty insignificant considering how much time the algorithm spent refining the surface. Our practical experience shows that a maximum of one or two refinements is quite adequate.

Let us now turn our attention to rational surfaces. Denoting the numerator in Equation 1 by $A(u, v)$ and the denominator by $w(u, v)$, the second derivatives are obtained as

$$S_{uu}(u, v)=\frac{A_{uu}(u, v)-2w_u(u, v)S_u(u, v)-w_{uu}(u, v)S(u, v)}{w(u, v)} \tag{17}$$

$$S_{vv}(u, v)=\frac{A_{vv}(u, v)-2w_v(u, v)S_v(u, v)-w_{vv}(u, v)S(u, v)}{w(u, v)} \tag{18}$$

$$S_{uv}(u, v) = \frac{A_{uv}(u, v)-w_{uv}(u, v)S(u, v)-w_u(u, v)S_v(u, v)-w_v(u, v)S_u(u, v)}{w(u, v)} \tag{19}$$

Computing the maxima of these derivatives is not particularly easy. We chose a different route. The idea is to work in 4D space and consider the surface as nonrational. More precisely, the surface of Equation 1 can be written as a nonrational surface in 4D space as follows:

$$S^w(u, v)=\sum_{i=0}^{n}\sum_{j=0}^{m} P_{i,j}^w N_{i,p}(u)N_{j,q}(v) \tag{20}$$

where $P_{i,j}^w=(w_{i,j}x_{i,j}, w_{i,j}y_{i,j}, w_{i,j}z_{i,j}, w_{i,j})$. The surface of Equation 1 is obtained by mapping $S^w(u, v)$ onto the $w=1$ plane. Now, given a tolerance ε, the surface $S^w(u, v)$ is to be tessellated to within a tolerance ε^w so that, when the tessellation in 4D is mapped to 3D, the surface of Equation 1 is tessellated to within the given tolerance ε. The 4D tolerance is obtained as follows[15]:

$$\varepsilon^w=\min_{i,j} w_{i,j}\frac{\varepsilon}{1+\max_{i,j}\|P_{i,j}\|} \tag{21}$$

Given this tolerance, the maximum edge length is computed as

$$\lambda^w=3\left(\frac{\varepsilon^w}{2(D_1^w+2D_2^w+D_3^w)}\right)^{1/2} \tag{22}$$

where D_1^w, D_2^w and D_3^w are bounds of the 4D derivatives computed just as the 3D ones are computed except that the computation takes place in 4D space. For example,

$$D_1^w=\sup_{(u,v)\in[a,b]\times[c,d]}\|S_{uu}^w(u, v)\|$$

where

$$S_{uu}^w(u, v)=\sum_{i=0}^{n-2}\sum_{j=0}^{m} N_{i,p-2}(u)N_{j,q}(v)P_{i,j}^{w(2,0)}$$

with

$$P_{i,j}^{w(2,0)}=\frac{p(p-1)}{u_{i+p+1}-u_{i+2}}\left(\frac{P_{i+2,j}^w-P_{i+1,j}^w}{u_{i+p+2}-u_{i+2}}-\frac{P_{i+1,j}^w-P_{i,j}^w}{u_{i+p+1}-u_{i+1}}\right)$$

The only extra cost is the additional 4D coordinate that needs to be considered when computing the derivative surfaces and the maximum derivatives.

Polygonal approximation of trimming curves

In the previous section we established a maximum edge length λ such that triangles in the parameter domain with sides less than λ map onto 3D triangles that are within ε distance from the surface. The task now is to select points within the domain as well as along the trimming curves so that when triangulating the domain no edge is longer than λ. We elected to select points inside the domain uniformly using a scanline-type algorithm (see below). To ensure that the maximum edge length in each triangle is less than λ, we chose the step length as

$$STEP=\frac{2^{1/2}}{2}\lambda$$

or

$$STEP^{w} = \frac{2^{1/2}}{2} \lambda^{w}$$

for rational surfaces (for simplicity, the rest of the paper uses STEP only to denote rational and nonrational step lengths). This ensures that, if the edge happens to be along the diagonal of a uniform point distribution, then it is less than λ. Now, selecting points along trimming curves can be done similarly to the approach presented for surfaces above, i.e. by establishing a maximum parameter increment Δt, based on the bound on the first derivative, for every trimming curve $C_k(t)$, $k=1, \ldots, N$, such that

$$\|C_k(t) - C_k(t+\Delta t)\| \leqslant \lambda$$

After implementing this method, it turned out to be inefficient. It provided a very uneven point distribution and the number of points generated was greater than expected. The method we implemented disregards the parameterization. It divides the trimming curves into two groups: (a) straight lines (either the bounds of the surface domain or straight line B-spline trimming curves), and (b) curved boundaries. For straight lines, the points are selected by simple division, that is, if the length of the line segment is L, then the number of subdivision points along the segment is

$$n = \left\lfloor \frac{L}{STEP} \right\rfloor$$

where $\lfloor x \rfloor$ denotes the integer part of x.

Curve boundaries are approximated by the following recursive algorithm:

```
$t_l = T_0$; $t_r = T_m$; HALFSTEP = STEP/2.0;
**Approximate**($t_l$,$t_r$)
{
  **while**(1)
  {
    $t = (t_l + t_r)/2.0$;
    $d_l = \|C(t) - C(t_l)\|$;
    $d_r = \|C(t) - C(t_r)\|$;
    **if** ($d_l >$ STEP **and** $d_r >$ STEP)
    {
      **Create_Point**($C(t)$);
      **Approximate**($t_l$,$t$);
      **Approximate**($t$,$t_r$);
      return;
    }
    **else if** ($d_l \leqslant$ STEP **and** $d_r \leqslant$ STEP)
    {
      **if** ($d_l \geqslant$ HALFSTEP **and** $d_r \geqslant$ HALFSTEP)
      {
        **Create_Point**($C(t)$); return;
      }
      **else if** ($d_l <$ HALFSTEP) $t_l = t$; **else** $t_r = t$;
    }
    **else if** ($d_l <$ STEP) $t_l = t$; **else** $t_r = t$;
  }
}
```

This routine recursively subdivides the curve until each edge satisfies the condition

$$HALFSTEP \leqslant \|C(t_i) - C(t_{i+1})\| \leqslant STEP$$

That is, points are not allowed to cluster together or to lie too far apart. There can be two problems with the above routine:

- The trimming curve is smaller than STEP. In that case the curve is simply discarded.
- The curve comes too close to itself at $t = (t_l + t_r)/2.0$ and hence the routine can stop without further dividing the remaining curve segments. This situation can be detected and corrected by keeping track of the parameter values.

Figure 3 illustrates the concept of the above algorithm and *Figure 4a* shows the result of point selection along the trimming curves shown in *Figure 2*.

Selection of points inside trimmed region

The selection of points inside the trimmed region is done via a simple scanline-type algorithm used in raster graphics to fill polygons[16]. Once the trimming curves are approximated, points inside the trimmed region are selected in a similar way to pixel selection in polygon fill algorithms. The method is outlined as follows (see *Figure 4b*):

(1) Establish the number of scanlines, that is, compute

$$NSL = \left\lfloor \frac{v_{\max} - v_{\min}}{STEP} \right\rfloor + 1$$

(2) Sort each edge into NSL number of v buckets by checking whether the edge intersects the current scanline. If the edge intersects more than one scanline, put it in the lowest bucket. For each bucket, sort the edges by increasing u value.

(3) Initialize an *active edge table* (AET) to be empty. Also set $v_{\text{incr}} = (v_{\max} - v_{\min})/NSL$ and $v = v_{\text{incr}}$.

(4) While $v < v_{\max}$ do

(4.1) Move from the current v bucket to the AET those edges whose $v_{\min} \leqslant v$ maintaining the AET sort order on u.

(4.2) Compute the intersection points of the v scanline with the active edges going from left to right. Set the 'in' and 'out' flags as edges are crossed.

(4.3) Compute

$$\left\lfloor \frac{d}{STEP} \right\rfloor$$

interior points within each 'in' segment, where d is the length of the segment (an 'in' segment has 'in–out' flag pairs whereas an 'out' segment's flag pair is 'out–in').

(4.4) Increment v, i.e. $v = v + v_{\text{incr}}$.

(4.5) Remove edges from the AET whose $v_{\max} \leqslant v$.

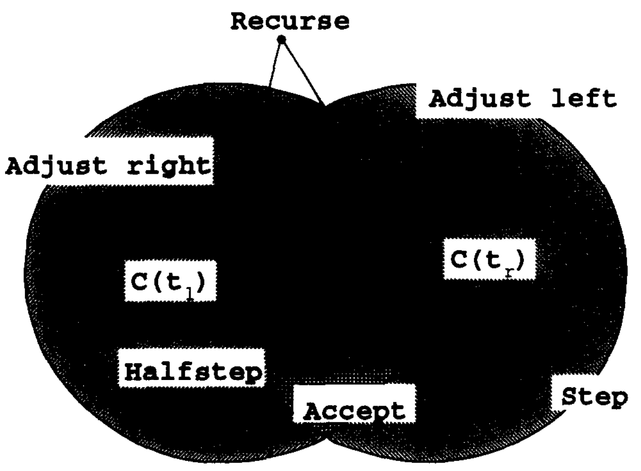


Figure 3 Concept of curve approximation

This routine does not consider the first and the last scanlines. This is correct because these lines can contain only vertex points and/or edges. Since the edges have been processed already and the vertices are kept as data points, these scanlines can be omitted. *Figure 4c* shows the result of point selection inside the valid domain of *Figure 2*.

Note that this method selects about $2^{1/2}$ times more points in each direction than a random or an adaptive one which would be based on λ. There is a tradeoff between simplicity and speed *versus* storage. The scanline method is very simple and fast; however, it produces more triangles than absolutely necessary.

Triangulating the trimmed region

For triangulating the trimmed region we used our algorithms that we developed for digital terrain modelling applications[17] and for triangulating multiply connected polygonal domains[18]. Since these algorithms are rather involved, we provide only the outline of the main ideas. For further details the reader is encouraged to study these methods.

The algorithm is tied to a specific data structure to store points as well as the boundary edges of the trimming curves. This structure is obtained by putting a uniform grid over the points and processing each point and edge into grid cells. Each cell then has a list of points lying inside the cell, and a list of edges intersecting the cell. The algorithm then proceeds in three steps:

- *Range searching:* Given an edge, find a point to form a valid triangle. Since trimming edges are to be preserved, the modified circumcircle criterion[19] is applied in the Delaunay triangulation. That is, if a circumcircle of a Delaunay triangle contains a point other than the vertices of the triangle, then every interior point of the triangle must be separated from this point by an edge of a trimming curve. To satisfy this criterion, for each edge, the algorithm finds a point for which the angle at this point is the largest and the sides of the triangle do not intersect any of the trimming edges.
- *Shelling:* Put triangles together so that completeness and correctness are maintained. At any stage of the triangulation, list the edges bounding the current triangulation in an edge list in, say, counterclockwise order. For each edge, form a modified Delaunay triangle, erase the current edge and add the two new edges in proper order to the edge list. Account for boundary edges and for situations where the newly computed triangle touches existing one(s).
- *Outut generation:* Create a data structure that stores triangles lying within the trimmed region and incorporates neighbouring information for further processing. We chose a point list based data structure that stores internal and boundary points separately. For each point, all points surrounding this point are listed in clockwise order. Internal point lists are circular, whereas boundary ones are not. Simple database browsing routines can be written to answer queries such as 'given an edge, what triangles share this edge?'. The advantages of this data structure are (a) it is very compact and requires only modest storage, (b) it can be built on the fly (as the triangulation part of the program adds new triangles, the point lists are updated with almost no effort), and (c) triangles outside the trimmed region can be discarded on the fly by introducing a few flags and flagging when the triangulation exits the valid domain and enters it again.

References 17 and 18 show walk-through examples to guide the reader in understanding the method. *Figure 4d* shows the triangulation of the domain depicted in *Figure 2*. Note that the method can triangulate multiply connected domains with no extra effort. There is no need to obtain the Boolean union of simple domains[3].

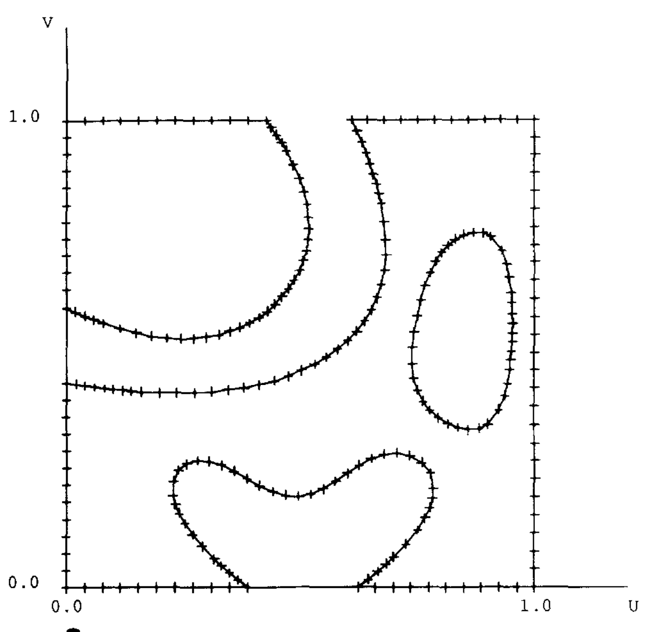
Mapping triangles onto the trimmed surface

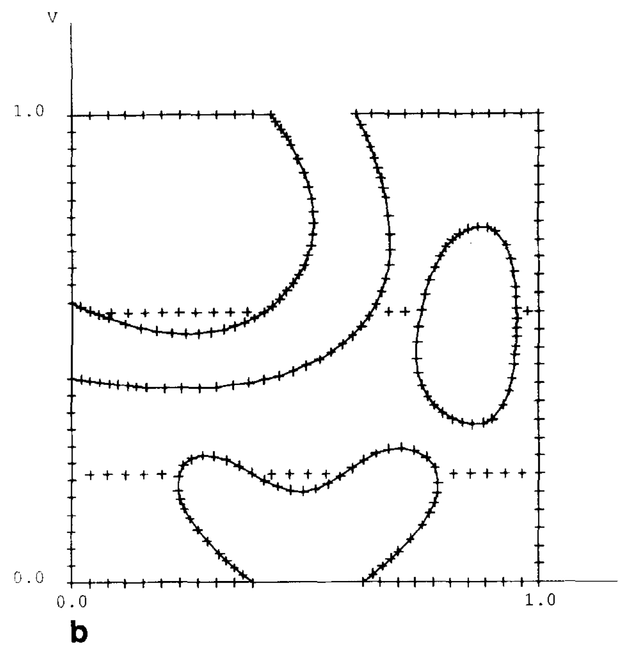
Once the triangulation of the trimmed domain is completed, the domain triangles are mapped onto the surface forming a tessellation. Although this is a straightforward map, a data structure has to be maintained so that the triangles can be passed onto a postprocessor such as a contouring program or a shader based on polygonal objects. The output data structure (the point lists) that we obtained from the triangulation module can still be used because the polygonized surface is topologically equivalent to the plane, and the neighbouring information is still maintained. *Figure 4e* illustrates the result of tessellating the trimmed surface shown in *Figures 1* and *2*.

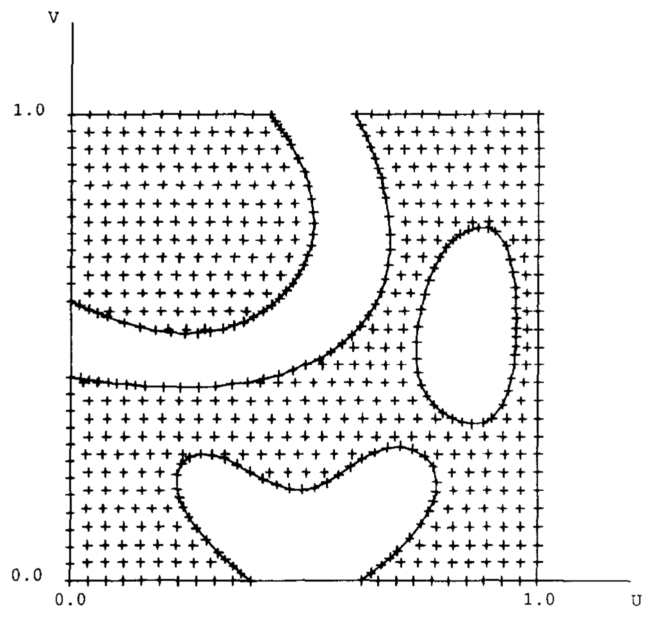
Efficiency considerations

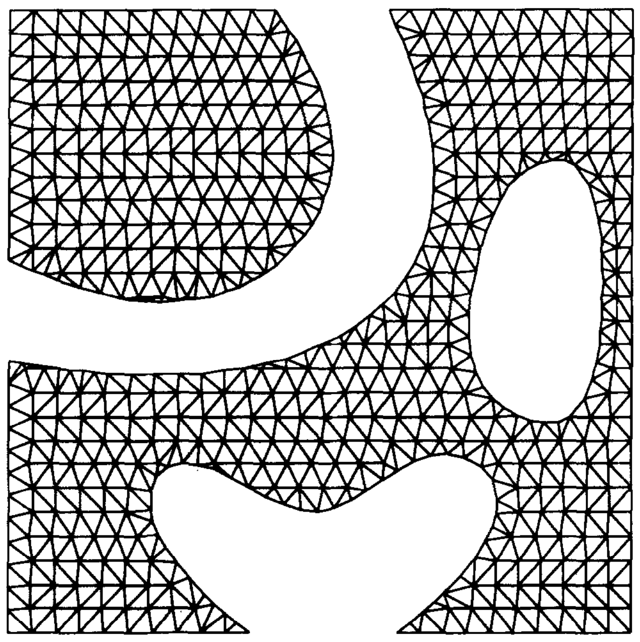
There are several factors that affect efficiency. We elaborate here on the following ones:

- *Why general triangulation routine should be used:* The uniform selection of points raises the question as to why not triangulate the region using some special code that takes advantage of the distribution of the points. There are two main reasons why we elected to use a general algorithm: (a) we think that the

a


b


c


d


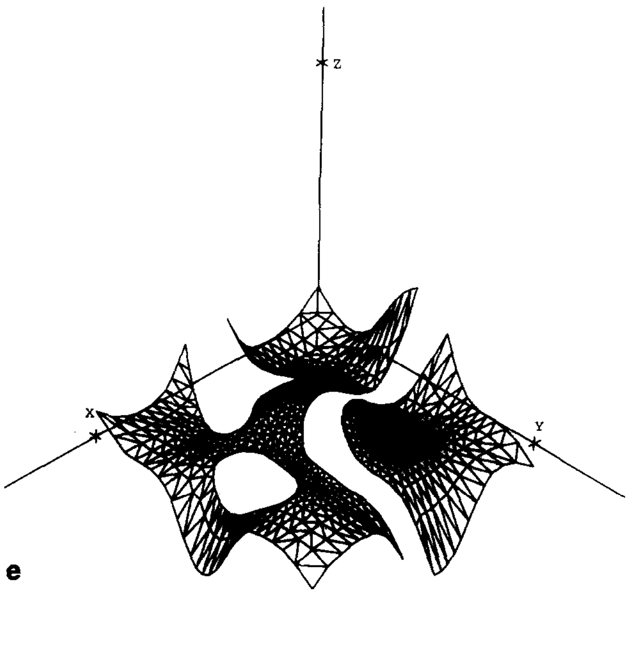
e


Figure 4 Point selection; (a) point selection along trimming curves, (b) scan-line method of selecting points inside trimmed region, (c) result of point selection inside valid domain, (d) triangulation of trimmed region of *Figure 2*, (e) tessellation of trimmed surface

special code would not be much simpler and faster than our triangulation code, and (b) the general triangulation algorithm allows any kind of point selection. If, for some reason, the user decides to use random distribution, only a small portion of the code needs to be modified.

- *Adaptive versus direct triangulation:* The method we outlined above is a direct triangulation that produces large numbers of triangles to guarantee that the maximum deviation *anywhere* along the surface is

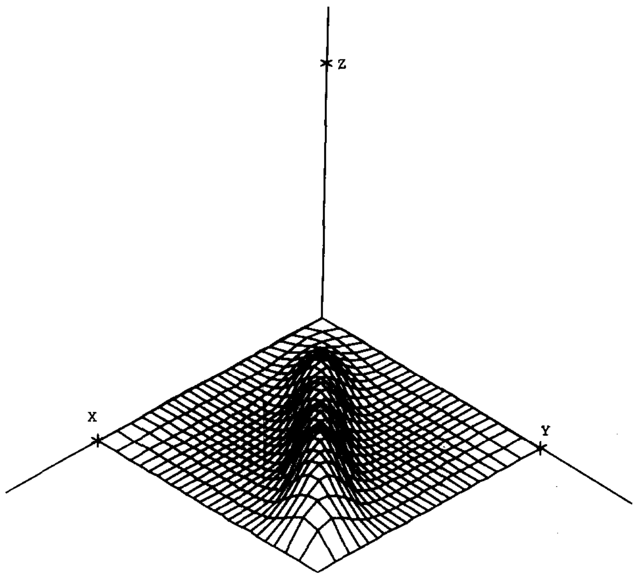


Figure 5 Nonrational bicubic test surface

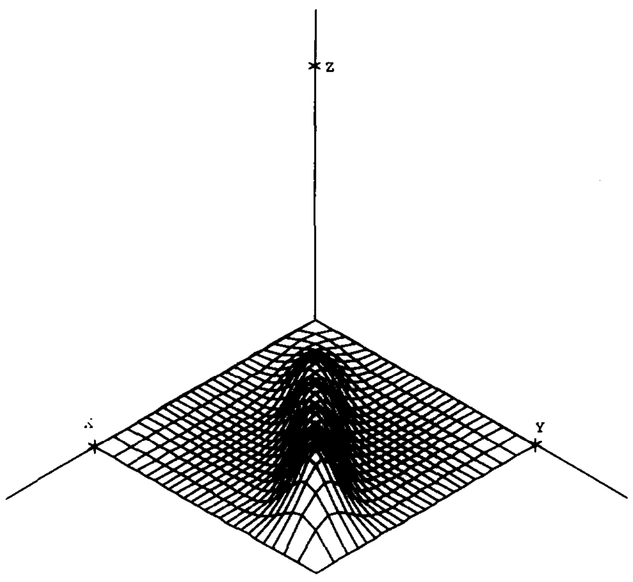


Figure 7 Rational bicubic test surface

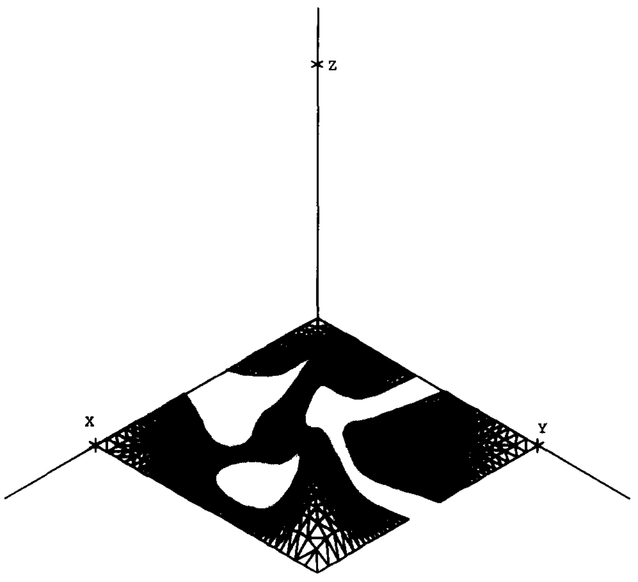


Figure 6 Tessellation of surface in *Figure 5* with $\varepsilon = 0.1$

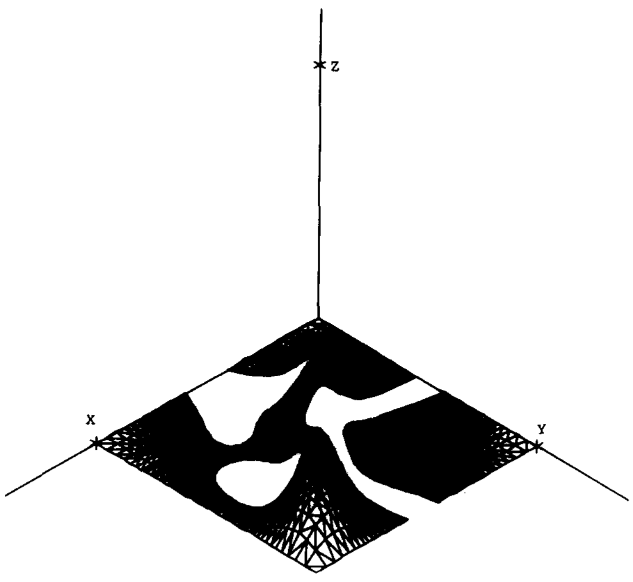


Figure 8 Tessellation of surface in *Figure 7* with $\varepsilon = 0.3$

Table 4 Numerical data for *Figure 6*

ε	N_Δ	$error_S$	$error_C$
0.2	1265	0.0117	0.0023
0.1	2430	0.0043	0.0011
0.05	4613	0.0035	0.0006
0.025	9116	0.0018	0.0004

Table 6 Numerical data for *Figure 10*

ε	N_Δ	$error_S$	$error_C$
0.2	2020	0.0122	0.0019
0.1	3877	0.0065	0.0009
0.5	7628	0.0025	0.0005
0.0025	14916	0.0014	0.0002

Table 5 Numerical data for *Figure 8*

ε	ε^w	N_Δ	$error_S$	$error_C$
0.4	0.165	2214	0.0038	0.0019
0.3	0.124	2853	0.0056	0.0011
0.2	0.080	4195	0.0038	0.0006
0.1	0.040	8276	0.0016	0.0004

Table 7 Numerical data for *Figure 12*

ε	ε^w	N_Δ	$error_S$	$error_C$
0.5	0.200	14423	0.0033	0.0003
0.4	0.160	17929	0.0029	0.0002
0.3	0.120	23846	0.0025	0.0001
0.2	0.080	35435	0.0015	0.0001

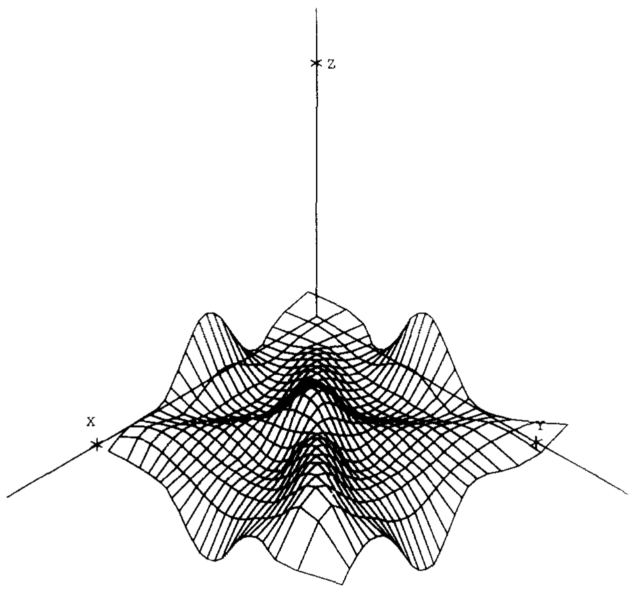


Figure 9 Nonrational bicubic test surface

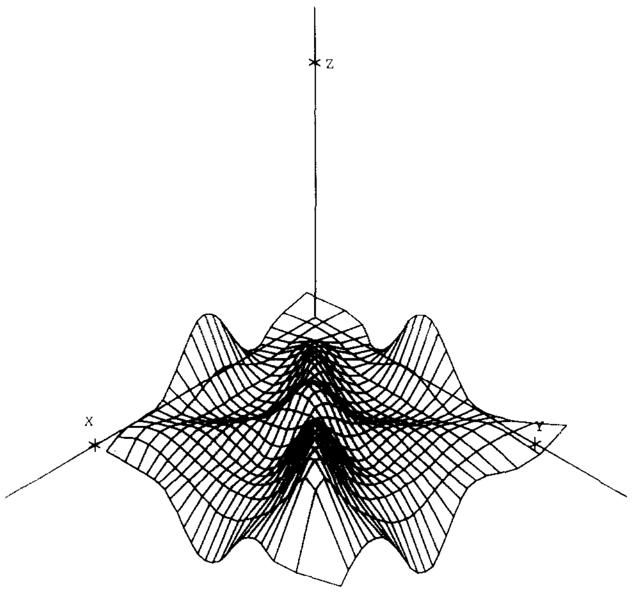


Figure 11 Rational bicubic test surface

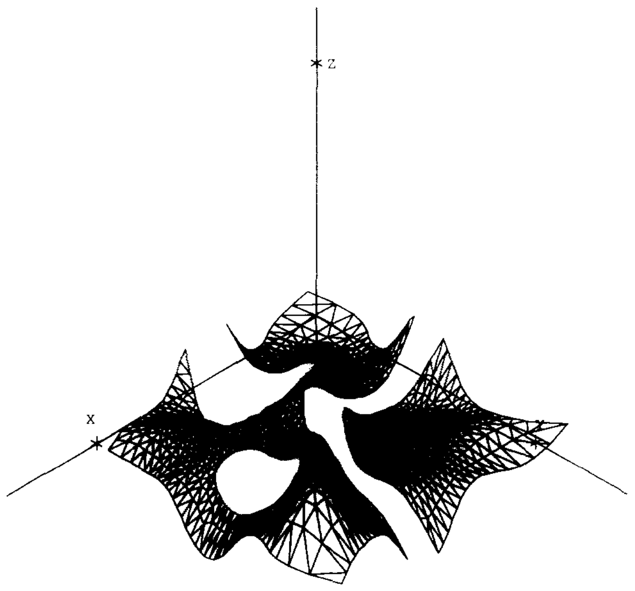


Figure 10 Tessellation of surface in *Figure 9* with $\varepsilon = 0.2$

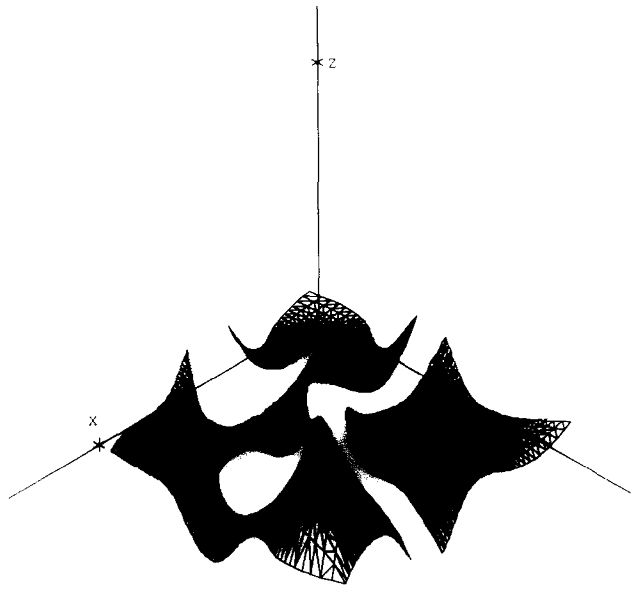
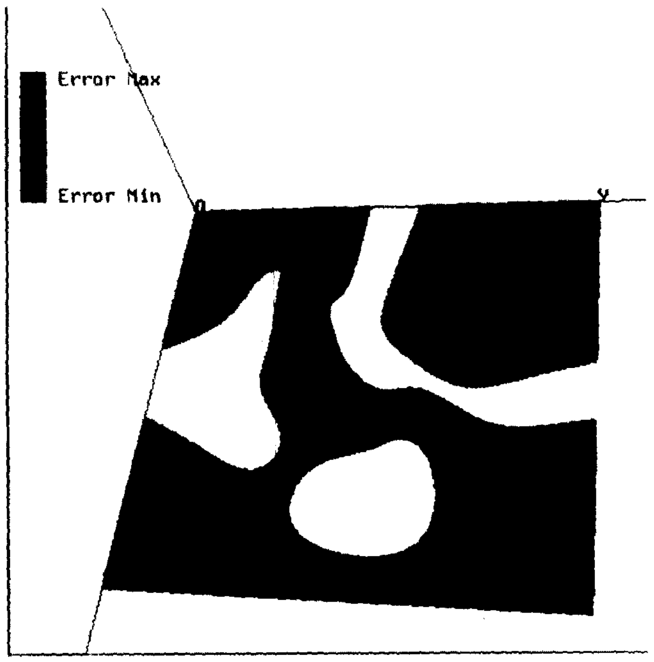


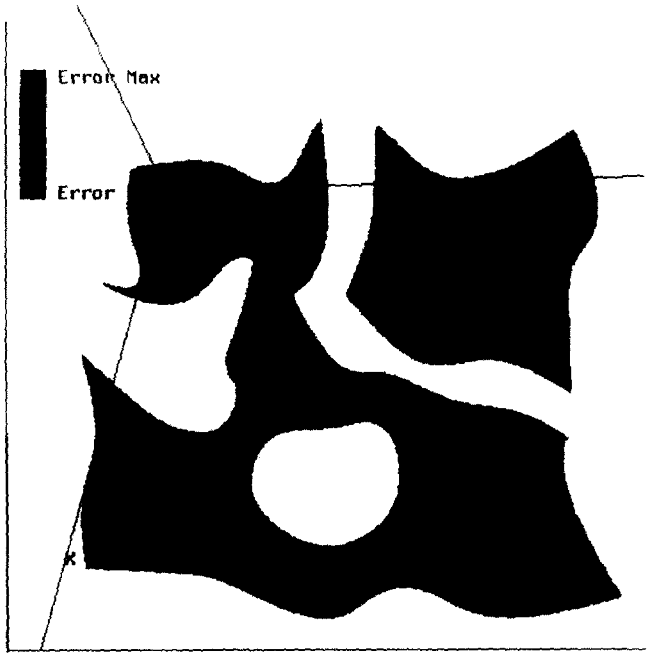
Figure 12 Tessellation of surface in *Figure 11* with $\varepsilon = 0.5$

less than a tolerance. Obviously, this method puts more triangles in low curvature areas than necessary to achieve an acceptable approximation. The two things that can be done are (a) do the triangulation adaptively, i.e. get an initial triangulation and refine it only in high curvature areas, and (b) postprocess the direct triangulation, i.e. throw away unnecessary triangles in low curvature areas. Both of these methods result in an optimal triangulation as far as the *number* of triangles is concerned. Unfortunately, obtaining a small number of triangles is only a minor problem. The major concern is numerical stability. It is well known that routines based on triangles are sensitive to the type of triangle. For example, long and skinny triangles create numerical problems. Also, if large triangles are mixed with small ones, many postprocessing routines perform badly. An example is contour smoothing: cut the triangular network with a plane and smooth the polygon into a curve. Since the data points can be distributed very unevenly, most of the interpolation routines would give rather unpleasant results (some of them would create unwanted wiggles or even loops). The method we propose generates reasonable evenly sized triangles although at the cost of data storage.

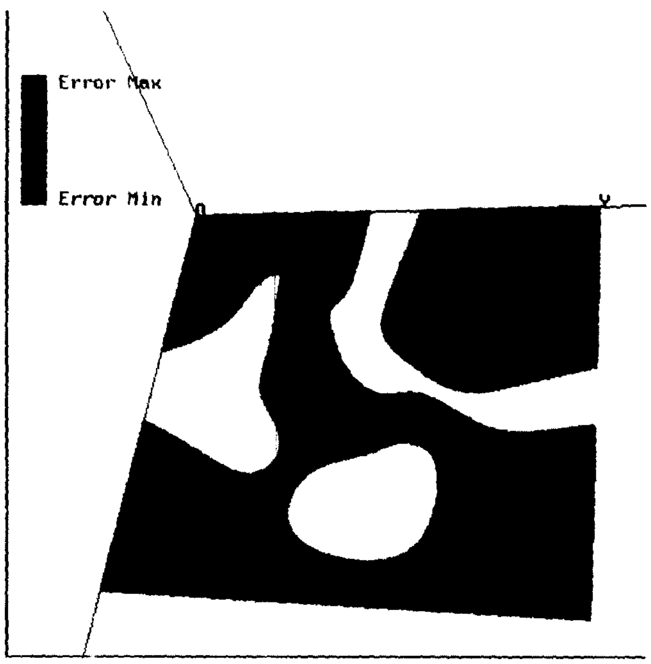
- *Which data structure should be used:* We used two separate data structures to select points inside the valid region (scanline method) and to triangulate that region (grid method). The grid structure could have been used to do both. The only problem is that we do not know in advance how many points will lie inside the trimmed region. Although this can be (over)estimated to create a grid structure, it would, however, slow down the triangulation. The active edge list based method has proven to be fast and robust.



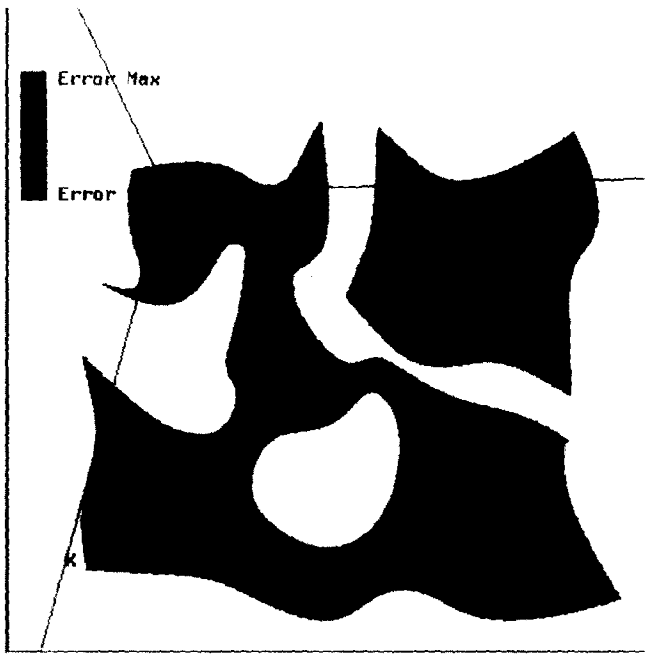
Colour Plate 1 Colour-coded error distribution for *Figure 6*



Colour Plate 3 Colour-coded error distribution for *Figure 10*



Colour Plate 2 Colour-coded error distribution for *Figure 8*



Colour Plate 4 Colour-coded error distribution for *Figure 12*

NUMERICAL TESTS AND EXAMPLES

We have tested our algorithm using a number of surfaces containing large as well as small curvature areas. The same trimmed region was used for all the test surfaces, i.e. the one shown in *Figure 2*, and the control points were chosen such that they fit into the unit cube. *Figure 5* shows a nonrational surface containing flat as well as bumpy areas. *Figure 6* shows the result of the tessellations using the tolerance 0.1. *Table 4* summarizes the numerical results where ε is a user chosen tolerance, N_Δ denotes the number of triangles, $error_S$ is the maximum error inside the trimmed region and $error_C$ is the maximum error along the trimming curves.

Figure 7 shows the same surface as in *Figure 5* except that a weight of $w_{7,7}=2.0$ is applied to turn the surface into a rational one. *Figure 8* shows the tessellation for $\varepsilon=0.3$ and *Table 5* summarizes the numerical results.

In *Figure 9* a more complicated surface is shown with varying curvatures along the boundaries as well as inside the surface. *Figure 10* illustrates the tessellation for $\varepsilon=0.2$ and *Table 6* gives the numerical results.

A final example is given in *Figure 11* which was obtained by applying weights to the control points in

Figure 9 ($w_{11} = w_{33} = w_{66} = w_{77} = w_{88} = 3.0$). The tessellation result is depicted in *Figure 12* for $\varepsilon = 0.5$. *Table 7* summarizes the numerical results.

Examining *Tables 4–7*, one thing is quite apparent: the theory behind the selection of edge length is too restrictive. Although it *guarantees* the result, the actual error is one (nonrational) or two (rational) magnitudes smaller than the user specified tolerance. It might happen that this is necessary for certain situations. However, the price we pay for it is quite significant both in storage and in computation time.

Colour Plates 1–4 show the error distributions for *Figures 6, 8, 10* and *12*, respectively. To see any error whatsoever, we normalized the distribution to the maximum error over the surface. That is, red represents the maximum error computed across the surface and blue represents zero error.

CONCLUSIONS

We presented an algorithm for tessellating trimmed NURBS surfaces in the parameter domain. Based on bounds on the second derivatives, the method estimates the longest edge in the domain to obtain a triangulation to within a user specified tolerance. The number of triangles computed is higher than the one an adaptive algorithm would produce. However, numerical stability and speed justify the results. The error analysis shows that on average the maximum error is a magnitude smaller than the user specified tolerance. However, to ensure that high curvature areas (that might or might not be cut out during trimming) are adequately dealt with, extra triangles are necessary. An analysis on the bounds of surface derivatives shows that simple 1-step or 2-step refinement of the derivative surface is adequate to obtain bounds resulting in the smaller number of triangles.

ACKNOWLEDGEMENTS

The work reported in this paper was supported by the US National Science Foundation under grant CCR-9217768 awarded to the University of South Florida, USA, and by CISI Ingénierie, France.

REFERENCES

1 Peng, Q S 'Volume modeling for sculptured objects' *PhD Thesis* University of East Anglia, UK (1983)

2 Miller, J R 'Sculptured surfaces in solid models: issues and alternative approaches' *Comput. Graph. & Applic.* Vol 6 No 12 (1986) pp 37–48

3 Casale, M S 'Free-form solid modeling with trimmed surface patches' *Comput. Graph. & Applic.* Vol 7 No 1 (1987) pp 33–43

4 Farouki, R T 'Trimmed surface algorithms for the evaluation and interrogation of solid boundary representations' *IBM J. Res. & Develop.* Vol 31 No 3 (1987) pp 314–334

5 Shantz, M and Chang, S-L 'Rendering trimmed NURBS with adaptive forward differencing' *Comput. Graph.* Vol 22 No 4 (1988) pp 189–198

6 Rockwood, A, Heaton, K and Davis, T 'Real-time rendering of trimmed surfaces' *Comput. Graph.* Vol 23 No 3 (1989) pp 107–116

7 Schafer, L J and Carlton, E H 'NURBS surfaces: tessellation and trimming' (in manuscript) (Fujitsu America, San Jose, CA, USA (1989))

8 Luken, W L and Cheng, F 'Rendering trimmed NURBS surfaces' (in manuscript)

9 Sheng, X and Hirsch, B E 'Triangulation of trimmed surfaces in parametric space' *Comput.-Aided Des.* Vol 24 No 8 (1992) pp 437–444

10 Piegl, L 'On NURBS: a survey' *Comput. Graph. & Applic.* Vol 11 No 1 (1991) pp 57–71

11 Prenter, P M *Splines and Variational Methods* John Wiley, USA (1975)

12 Lane, J and Carpenter, L 'A generalized scan line algorithm for the computer display of parametrically defined surfaces' *Comput. Graph. & Image Proc.* Vol 11 pp 290–297

13 Filip, D, Magedson, R and Markot, R 'Surface algorithms using bounds on derivatives' *Comput. Aided Geom. Des.* Vol 3 No 4 (1986) pp 295–311

14 Dolenc, A and Mäkelä, I 'Optimized triangulation of parametric surfaces' in *The Mathematics of Surfaces IV* Clarendon Press, UK (1991) pp 1–13

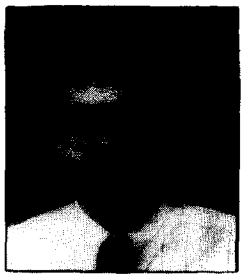
15 Tiller, W 'Knot-removal algorithms for NURBS curves and surfaces' *Comput.-Aided Des.* Vol 24 No 8 (1992) pp 445–453

16 Foley, J D, van Dam, A, Feiner, S K and Hughes, J F *Computer Graphics — Principles and Practice* Addison-Wesley, USA (1990)

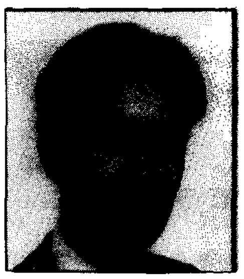
17 Fang, T-P and Piegl, L A 'Delaunay triangulation using a uniform grid' *Comput. Graph. & Applic.* Vol 13 No 3 (1993) pp 36–47

18 Piegl, L A and Richard, A M 'Algorithm and data structure for triangulating multiply connected polygonal domains' *Comput. & Graph.* Vol 17 No 5 (1993) pp 563–574

19 Cline, A K and Renka, R J 'A constrained two-dimensional triangulation and the solution of closest node problems in the presence of barriers' *SIAM J. Numer. Anal.* Vol 27 No 5 (1990) pp 1305–1321



Les A Piegl is a professor in the Department of Computer Science and Engineering, University of South Florida, USA. His research interests are in CAD/CAM, geometric modelling, user interface design, data structures and algorithms, and computer graphics. He has spent many years researching and implementing NURBS routines in academia as well as in industry.



Arnaud M Richard holds a civil engineering degree from the Ecole des Mines de Paris, France. He then graduated from the Institut Supérieur d'Informatique et d'Automatique in Sophia-Antipolis with a degree in computer science and systems control. He worked in the Visual Simulation Group of Renault for two years before visiting the Computer Science and Engineering Department at the University of South Florida, USA. His research interests are in geometric algorithms, 3D graphics, CAD/CAM and visualization. He is currently an engineer with Giravions Dorand Industries.