Interactive Visualization of Triangular Bézier Surfaces

Subodh Kumar
Department of Computer Science
Johns Hopkins University
Baltimore, MD, 21218. USA.
subodh@cs.jhu.edu
http://www.cs.jhu.edu/~subodh

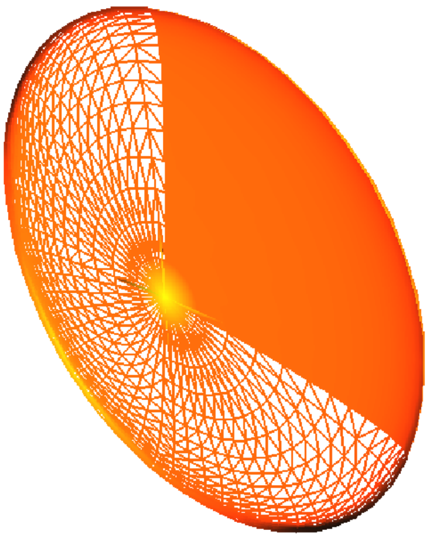


Figure 1: Teapot base: a triangle-shaped patch (shaded)

Abstract

We present an efficient algorithm for rendering surface models comprising triangular Bézier surfaces. Each surface patch is dynamically triangulated using uniform domain tessellation. The density of tessellation depends on the viewing position. The overall algorithm includes efficient visibility computation and use of coherence to perform incremental tessellation. The algorithm allows models to consist of both triangular and tensor-product surfaces and generates triangulations without cracks at surface boundaries. In addition, techniques to convert degenerate tensor-product surfaces to triangular form are presented.

1 Introduction

Non-Uniform Rational B-Spline surfaces are routinely used to design surfaces of geometric models like automobiles, submarines, airplanes, etc. This class of representation includes tensor-product and triangular Bézier surfaces. Often these surfaces are approximated with triangles for efficient display on current graphics systems. A one-time *static* tessellation typically requires the generation of a large number of triangles, which can lead to a slow rendering speed. However, recent advances in *dynamic* tessellation algorithms [RHD89, AES91, KML96] have resulted in significant speedup in rendering curved surfaces. Unfortunately, all of these algorithms assume the tensor-product representation. Due to the lack of support for displaying triangular Bézier surfaces, designers are restricted to using degenerate, tensor-product parametrization for triangular surfaces. For example, the base of the Utah teapot consists of four tensor-product Bézier patches (one of them shaded in Fig. 1). However, each of these patches is three sided. A four sided parametrization is obtained by shrinking one of the four boundary curves, say $v = 1$, to a point. The partial derivative, $\mathbf{F}_v$, of the surface vanishes at that point making the normal degenerate which in turn leads to shading anomaly (Fig. 3). Further, uniform triangulation in the vicinity of such points tends to result in long and skinny triangles. Such triangles not only produce rendering artifacts but also degrade the performance of algorithms like finite element analysis. A triangular domain provides a better behaved parametrization for such surfaces. If such a triangular patch is ill-parametrized as a tensor product, we convert it to the more appropriate triangular form.

This paper extends the results of Kumar et al. [KML96, Kum96] to triangular domains. In particular, it presents algorithms for view-point dependent uniform sampling and tessellation of triangular patches. While it is possible to reparametrize a triangular domain onto a rectangular domain [IW92] and then apply the algorithms designed for tensor-product surfaces, that defeats the purpose of using triangular Bézier surfaces (see section 3). We present an algorithm to directly generate (planar) triangle strips for triangular

Bézier patches. This algorithm is similar in approach to that for display of tensor-product surfaces [Kum96]. This similarity allows us to mix triangular and tensor-product surfaces in the same model and generate tessellations free of cracks between adjacent surfaces even if they are parametrized on different domains.

Curved surface rendering has been an active area of research for more than two decades. The techniques based on pixel-level surface subdivision, ray tracing and scan-line display [Cat74, LCWB80, Kaj82, NSK90] (See [Kum96] for a treatise.) typically do not take advantage of hardware rendering acceleration available on current graphics systems. As a result they are not as efficient as the techniques based on surface polygonization. A number of different methods, based on adaptive or uniform subdivision, have been proposed for polygonization of tensor-product surfaces [RHD89, FK90, AES91, LC93, PR95, KML96]. Adaptive tessellation of triangular surfaces [Pet84, Fil86] have also been used. Surprisingly, however, literature seems silent on algorithms for uniform tessellation for triangular parametric surfaces. As shown by a study in [Kum96], uniform tessellation algorithms often outperform adaptive algorithms as they do not require multiple termination tests. Our desire to interactively render large models made of triangular Bézier surfaces inspired us to investigate uniform tessellation for triangular patches. We present such an algorithm which has the performance characteristics similar to the one for tensor-product surfaces [KML96] and can render more than ten thousand Gouraud shaded triangular patches per second on an SGI Indigo 2 with Maximum Impact graphics. The main components of our algorithm are:

1. Re-parametrize degenerate tensor-product patches on a triangular domain
2. Pre-compute bounds on magnitudes of derivatives for each surface patch
3. Compute patch visibility; discard a patch that is completely hidden
4. Perform view-dependent sampling of each patch
5. Perform crack-free triangulation of each patch

In the rest of this paper, a basic familiarity with NURBS and Bézier surfaces is assumed. In section 2 we present the mathematical basics to introduce our notation. Section 3 presents the reparametrization technique we use to rid our models of degenerate parametrization. The recently discovered back-patch visibility computation algorithm [KM96] extends rather trivially to triangular surfaces and is briefly described in section 4. The sampling theory for tensor-product patches extends quite gracefully to triangular patches as described in section 5. We present the tessellation algorithm for triangular surfaces in section 6. Section 7 discusses some results of our prototype implementation and presents our conclusions.

2 Mathematical Background

Bézier surfaces defined over a triangular domain are often called Bézier triangles. Sabin [Sab77] first generalized Bézier to triangular B-splines. The theory of triangular NURBS has been further developed recently by Dahmen et al. [DMS92].

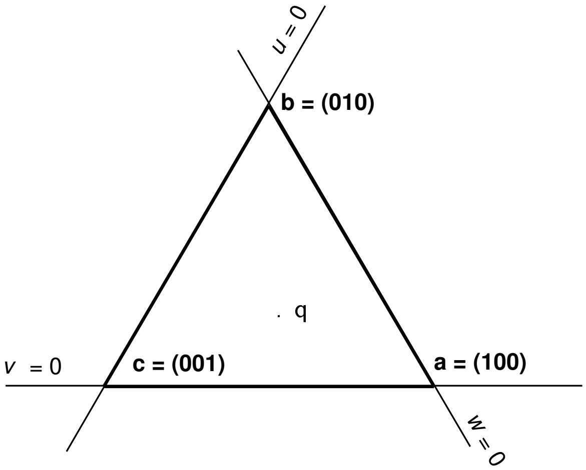


Figure 2: Triangular domain and barycentric coordinates

We use the barycentric coordinates of the domain triangle. In Fig. 2, the triangle **abc** represents the domain triangle. Each point on the domain is represented by three coordinates, (u, v, w). The coordinates of a point **q** are:

$$u = \frac{\mathcal{A}(\mathbf{q},\mathbf{b},\mathbf{c})}{\mathcal{A}(\mathbf{a},\mathbf{b},\mathbf{c})},\ v = \frac{\mathcal{A}(\mathbf{a},\mathbf{q},\mathbf{c})}{\mathcal{A}(\mathbf{a},\mathbf{b},\mathbf{c})},\ w = \frac{\mathcal{A}(\mathbf{a},\mathbf{b},\mathbf{q})}{\mathcal{A}(\mathbf{a},\mathbf{b},\mathbf{c})},$$

where $\mathcal{A}$ is the area function. Notice that $u+v+w = 1$, thus there are only two independent variables. A triangular Bézier patch of degree n is given by:

$$\mathbf{F}(\mathbf{u}) = \frac{\displaystyle\sum_{|\mathbf{i}|=n} w_{\mathbf{i}}\mathbf{p}_{\mathbf{i}}\mathcal{B}^n_{\mathbf{i}}(\mathbf{u})}{\displaystyle\sum_{|\mathbf{i}|=n} w_{\mathbf{i}}\mathcal{B}^n_{\mathbf{i}}(\mathbf{u})} = \frac{\mathbf{P}(\mathbf{u})}{W(\mathbf{u})}, \tag{1}$$

where **u** is a point on the domain given by some (u, v, w), **i** is a vector of three numbers (i, j, k), $|\mathbf{i}| = i + j + k$, and

$$\mathcal{B}^n_{\mathbf{i}}(\mathbf{u}) = \frac{n!}{i!j!k!}u^i v^j w^k;\ i+j+k = n.$$

The $\mathbf{p_i}$ form a triangular mesh of control points. The three boundary curves of a triangular Bézier patch, $u = 0, v = 0$, and $w = 0$, are all Bézier curves. Instead of partial derivatives with respect to the two parameters of the rectangular domain, we now compute directional derivatives. The cross product of two (non-parallel) directional derivatives produces the normal at a point on the surface. The directional derivatives of a Bézier surface are again triangular Bézier surfaces themselves. In general the polynomial derivative in the direction $\mathbf{d} = (d, e, f)$ is

$$D_{\mathbf{d}}(\mathbf{P}(\mathbf{u})) = n \sum_{|\mathbf{i}|=n-1} \mathbf{q}_{\mathbf{i}} \mathcal{B}_{\mathbf{i}}^{n-1}(\mathbf{u}),$$

$$\mathbf{q}_{\mathbf{i}} = \left[d w_{\mathbf{i+e1}} \mathbf{p}_{\mathbf{i+e1}} + e w_{\mathbf{i+e2}} \mathbf{p}_{\mathbf{i+e2}} + f w_{\mathbf{i+e3}} \mathbf{p}_{\mathbf{i+e3}} \right],$$

where, $\mathbf{e1} = (1, 0, 0)$, $\mathbf{e2} = (0, 1, 0)$ and $\mathbf{e3} = (0, 0, 1)$. In particular, the derivative along the u, v and w axes, respectively are:

$$D_u(\mathbf{P}(\mathbf{u})) = n \sum_{|\mathbf{i}|=n-1} (w_{\mathbf{i+e1}} \mathbf{p}_{\mathbf{i+e1}} - w_{\mathbf{i+e3}} \mathbf{p}_{\mathbf{i+e3}}) \mathcal{B}_{\mathbf{i}}^{n-1}(\mathbf{u})$$

$$D_v(\mathbf{P}(\mathbf{u})) = n \sum_{|\mathbf{i}|=n-1} (w_{\mathbf{i+e2}} \mathbf{p}_{\mathbf{i+e2}} - w_{\mathbf{i+e3}} \mathbf{p}_{\mathbf{i+e3}}) \mathcal{B}_{\mathbf{i}}^{n-1}(\mathbf{u})$$

$$D_w(\mathbf{P}(\mathbf{u})) = n \sum_{|\mathbf{i}|=n-1} (w_{\mathbf{i+e2}} \mathbf{p}_{\mathbf{i+e2}} - w_{\mathbf{i+e1}} \mathbf{p}_{\mathbf{i+e1}}) \mathcal{B}_{\mathbf{i}}^{n-1}(\mathbf{u})$$

Sometimes we abbreviate $D_u(\mathbf{P}(\mathbf{u}))$ and $D_v(\mathbf{P}(\mathbf{u}))$ as $\mathbf{P}_u$ and $\mathbf{P}_v$, respectively. To compute the derivatives of rational patch $D_u\left(\frac{\mathbf{P}}{W}\right)$, we use the quotient rule: $\frac{W D_u(\mathbf{P}) - D_u(W)\mathbf{P}}{W^2}$.

3 Reparametrization

Triangular surface patches are often parametrized as tensor-product patches resulting in degeneracies. It is often possible to reparametrize those as Bézier triangles and eliminate the degeneracies. For example Fig. 3(a) shows a tensor-product sphere — notice the erroneous bright spot at the pole. In contrast, when converted to a set of Bézier triangles (Fig. 3(b)), the final image has no such artifact. A tensor-product domain $\underline{u} \times \underline{v}$ can be reparametrized on a triangular domain $\mathbf{u} = (u, v, w)$ as follows: $u = \underline{u}(1 - \underline{v})$; $v = \underline{v}; w = 1 - u - v$.

Since $\mathtt{det}[\mathbf{u}_{\underline{u}} \mathbf{u}_{\underline{v}}] \neq 0$, we obtain the exact Bézier triangle. To obtain the control points of the corresponding triangular patch, we can employ functional composition [DGHM93]. However, a degree $m \times n$ tensor-product patch can result in degree $(m+n)$ triangular surface. This increase in degree not only slows down the surface evaluation (which is quadratic in degree), but also induces stability problems in computation of bounds on surface derivatives. We have found that an approximate parametrization for display purposes is sufficient and speeds up surface display. In this approximation we ensure that the tangent planes at the surface boundaries do not change, thus ensuring that no shading discontinuities are introduced. In our application we are primarily interested in reparametrizing tensor-product patches that have one of their boundary curves identified to a point.

Without loss of generality, assume that the curve $\underline{v} = 1$ on the degree $n \times n$ tensor-product patch $\underline{\mathbf{F}}$ is identified to a point and that the control points $\underline{\mathbf{b}}_{in}$ are all equal (see Fig. 4). For simplicity of discussion, further assume that $\underline{\mathbf{F}}$ is non-rational. The results hold for rational surfaces as well. The boundary

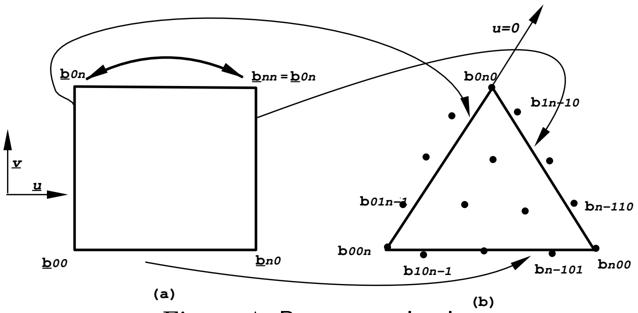
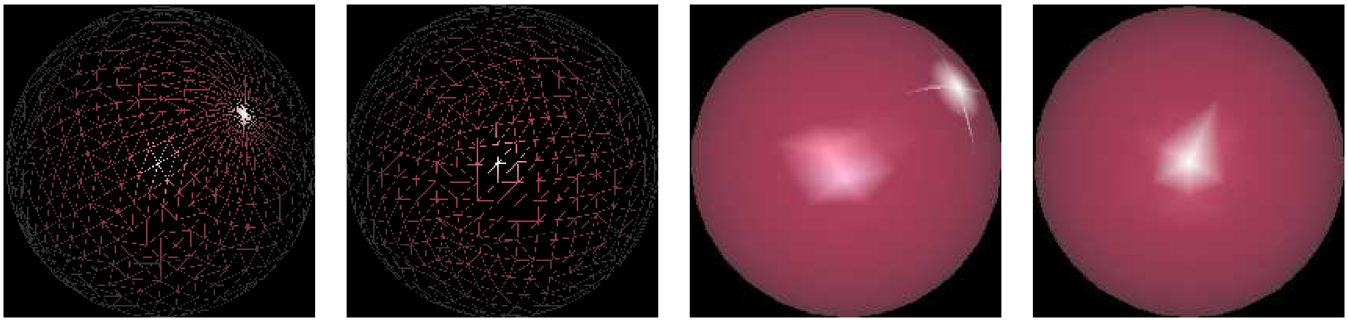


Figure 4: Reparametrization

control points of the equivalent triangular patch $\mathbf{F}$ are the same as those of $\underline{\mathbf{F}}$. In particular, $\mathbf{b}_{i0(n-i)} = \underline{\mathbf{b}}_{i0}$, $\mathbf{b}_{0i(n-i)} = \underline{\mathbf{b}}_{0i}$ and $\mathbf{b}_{i(n-i)0} = \underline{\mathbf{b}}_{n0}$. We cannot, however, similarly equate the control points of the derivative curves at the boundary. To see why, consider the cross boundary derivatives along the curve $u = 0$ on $\mathbf{F}$. To equate these derivatives with the corresponding cross-derivatives $\underline{\mathbf{F}}_u(0, \underline{v})$, we need to set $\mathbf{b}_{1i(n-1-i)}$ such that $\mathbf{b}_{1i(n-1-i)} - \mathbf{b}_{0i(n-i)} = \underline{\mathbf{b}}_{1i} - \underline{\mathbf{b}}0i$. Recall, however, that $\mathbf{b}_{1(n-1)0}$ already was set to $\underline{\mathbf{b}}_{n(n-1)}$ to equate the boundary curves. Simply raising the degree of the boundary curves does not solve this problem. Rather, we need to relax the condition that the cross-boundary derivates be the same. We only require that the tangent plane at the boundary curve remains the same after reparametrization, i.e. $\mathbf{F}_h \times \mathbf{F}_v$ is parallel to $\underline{\mathbf{F}}_{\underline{u}} \times \underline{\mathbf{F}}_{\underline{v}}$, where h is an arbitrary vector on the domain of $\mathbf{F}$. In other words, $\mathbf{F}_h \cdot \underline{\mathbf{F}}_{\underline{u}} \times \mathbf{F}_v = 0$ as $\mathbf{F}_v = \underline{\mathbf{F}}_{\underline{v}}$ by construction. If we allow $h = \alpha u + \beta v$ for arbitrary constants α and β, we obtain enough degrees of freedom to set up n simultaneous equations to compute the control points $\mathbf{b}_{1ij}$: the row adjacent to the boundary points $\mathbf{b}_{0ij}$. We equate the control points $\underline{\mathbf{F}}_{\underline{u}}$ along $\underline{u} = 0$ and those of $\mathbf{F}_h$ along $u = 0$. First, we must elevate the degree [Far93] of $\mathbf{F}$ to $n+1$ in order to ensure that we have the same number of control points on both sides. Overloading notation, we denote the control points of $\mathbf{F}$ after degree-elevation also by $\mathbf{b}_{\mathbf{u}}$ in the following equations:

$$\kappa_1(n+1)[\beta \mathbf{b}_{\mathbf{i+e1}} + (\alpha - \beta)\mathbf{b}_{\mathbf{i+e2}} - \alpha \mathbf{b}_{\mathbf{i+e3}}] = n[\underline{\mathbf{b}}_{1i} - \underline{\mathbf{b}}0i], \ |\mathbf{i}| = n. \tag{2}$$

It is worth pointing out that these equations are effectively linear even though α and β appear as products of unknown $\mathbf{b}_{\mathbf{i}}$'s on LHS. This is due to the fact that the first and the last equations contain non-zero coefficients only for α and β. The unknown control points are obtained up to a scalar factor κ_1 after solving (2). We similarly compute the other two rows of control points, up to scalar factors κ_2 and κ_3, respectively. Note that independent scalar multiples of points $\mathbf{b}_{11(n-2)}, \mathbf{b}_{1(n-2)1}$ and $\mathbf{b}_{(n-2)11}$ are computed twice. By equating these, we can solve for



(a) **Tensor-product** (b) **Triangular** (c) **Tensor-product** (d) **Triangular**

Figure 3: Reparametrization of degenerate tensor-product surfaces. Notice the absence of skinny triangles for the triangular patches. Also, notice the absence of the erroneous bright spot at a pole, which occurs in the tensor-product case due to degenerate normals.

κ_1, κ_2 and κ_3 and thence compute the control points exactly.

If the degree of $\mathbf{F}$ is higher than 4, there are $n_F = \frac{(n+2)(n+1)}{2} - (6n-9)$ extra internal control points to be found. We compute these by choosing n_F arbitrary points $(\underline{u}_j, \underline{v}_j)$ and setting up a linear system of n_F (vector) equations in n_F unknowns:

$$\mathbf{F}(u_j, v_j, w_j) = \underline{\mathbf{F}}((\underline{u}_j, \underline{v}_j), \text{ where } u_j = \underline{u}_j(1-\underline{v}_j) \text{ and } v_j = \underline{v}_j.$$

The problem with this method is that the points must be picked such that the resulting set of linear equations has full rank. However, the probability of selecting dependent equations is so small that a random choice works well in practice — if a selection is found to generate a non-independent equation, we just choose a different random point.

If $\mathbf{F}$ is trimmed, we retain the same trimming curve equations. These equations evaluate to point $(\underline{u}, \underline{v})$ on the domain of $\underline{\mathbf{F}}$. We simply transform the result to (u, v, w) on the domain of $\mathbf{F}$ before performing the triangulation.

4 Visibility

Recently [KML95] introduced the concept of back-patch culling to discard Bézier patches facing away from the viewpoint in addition to those outside the viewing frustum. A patch is marked as outside the viewing frustum if all its control points lie outside the frustum. Similarly a patch is marked as back-facing if all its normals face away from the view point. This condition can be verified by checking if a bounding volume of the pseudo-normal [KML96] surface of the patch lies in the back-space (shaded region in Fig. 5). The pseudo-normal surface for a degree $m \times n$ tensor-product patch, $\mathbf{F}$, is $\mathbf{N} = \mathbf{F}_u \times \mathbf{F}_v$, which was shown to be a Bézier patch of degree $3m \times 3n$ by [KML95]. The pseudo-normal patch for a triangular Bézier patch is computed similarly; only $\mathbf{F}_u$ and $\mathbf{F}_v$ are directional derivatives along the primary axis, namely, $D_u(\mathbf{F})$ and $D_v(\mathbf{F})$, each of which is a degree $2n$ triangular Bézier patch. The cross product

$$\mathbf{N} = \frac{(WD_u(\mathbf{F}) - \mathbf{F}D_u(W)) \times (WD_v(\mathbf{F}) - \mathbf{F}D_v(W))}{W^4} =$$
$$\frac{WD_u(\mathbf{F}) \times D_v(\mathbf{F}) - D_v(W)D_u(\mathbf{F}) \times \mathbf{F} - D_u(W)\mathbf{F} \times D_v(\mathbf{F})}{W^3}$$

is a degree $3n$ triangular Bézier patch. Once we compute the bounding volumes of the control points of this patch, we store the volumes in a ONE-tree [KM96]. The tree traversal algorithm of [KM96] then culls the patch if the volume lies in back-space.

5 Sampling

In order to generate sufficiently dense (with respect to the current view-point) tessellation for a surface patch, we impose restrictions on the property of triangles generated. These properties include on-screen size of the triangles, their deviation from the surface, etc. [Kum96] discusses several other tessellation criteria. These criteria require the evaluation of bounds on the maximum magnitude of different surface derivatives. One commonly employed assurance is a user-specified bound Δ on the screen size of the triangles generated: To assure this bound we need to compute the maximum magnitudes of the first derivatives of a given surface patch in screen coordinates.

Given Δ, [Kum96] shows how the computation of the sampling size (assuming uniform sampling) may be decomposed into an expensive object space computation and a cheap screen space transformation:

1. Along each dimension, determine δ_o, the number of domain steps needed to ensure that each resulting triangle has no edge larger than Δ_o in object space. This is an off-line process.
2. At run time, find $\mathcal{S}$, the minimum scaling factor [Kum96] of vectors due to viewing transformation. A step size of $\delta = \delta_o \mathcal{S}$ guarantees that the screen size of each edge of the resulting triangle is less than Δ.

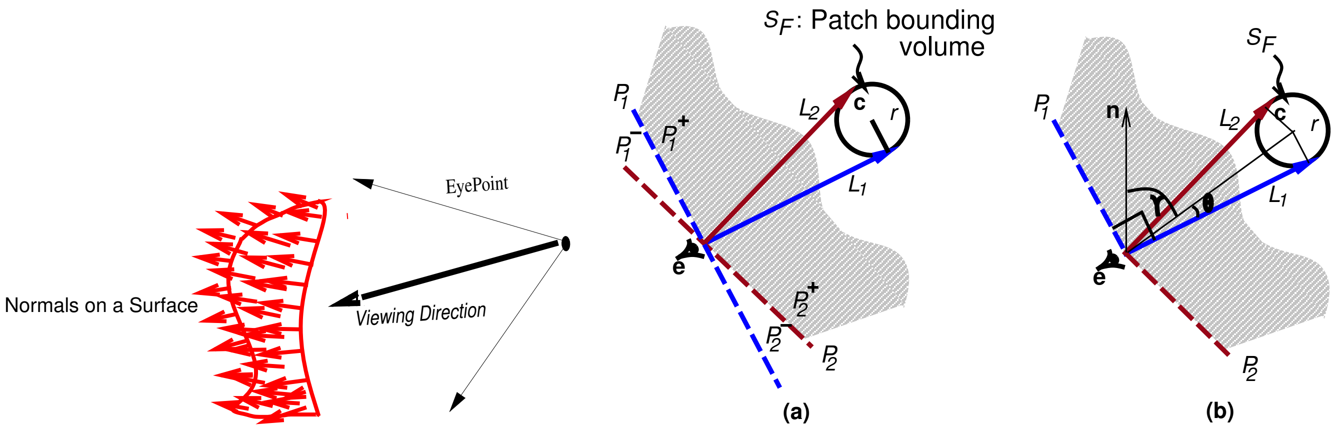


Figure 5: Back-patch visibility computation

To perform stage 1, we need to find a bound on the maximum magnitude of directional derivatives of a Bézier triangle. We find the maxima in all three directions: $u = 0$, $v = 0$ and $w = 0$. This is equivalent to finding the maxima of differences of partial derivatives $M_w = |F_u - F_v|$, $M_v = |F_u - F_w|$ and $M_u = |F_v - F_w|$. These maxima are computed using the method of resultants [MD94, Kum96]. A domain step-size of δ_o in the direction parallel to $u = 0$ results in an object space step-size of no more than $M_w \delta_o$ (by mean value theorem). In order to perform the uniform tessellation we discard the minimum of the three maxima, say M_w, and pick iso-parametric lines parallel to the axes corresponding to the other two maxima, at intervals δ_u and δ_v, respectively (see Fig. 6(c)).

Interestingly, if we generalize this technique to tensor-product surfaces and compute $F_d = \min(|F_{u+v}|_{\max}, |F_{u-v}|_{\max})$[1] in addition to $|F_u|_{\max}$ and $|F_v|_{\max}$, we can reduce the total number of triangles generated by [KML96]: Instead of using $\frac{M_u}{\sqrt{2}}$ and $\frac{M_v}{\sqrt{2}}$, respectively, for the step-sizes along the u and v axes we can choose $\max(\frac{M_u}{\sqrt{2}}, F_d)$ and $\max(\frac{M_v}{\sqrt{2}}, F_d)$, respectively.

Stage 2, the on-line part, is exactly the same as that for tensor-product patches. For each patch, we compute its maximum scaling factor [Kum96] for the current frame and multiply the corresponding δ_o by the factor.

This sampling algorithm computes δ_u, δ_v, and δ_w, the step sizes needed in each of the primary directions. However, if adjacent patches generate different numbers of samples on their common boundary, the resulting triangulation would exhibit cracks. In order to avoid these cracks, while still allowing the two patches to be tessellated independently, we sample the interior separately from the boundary curves. Thus the sampling algorithm also computes δ_1, δ_2 and δ_3, the number of samples needed to satisfactorily tessellate each boundary curve. If the boundary curve is represented in the same way on the two patches adjacent to it, the triangulations of the two patches automatically match up. Recall that reparametrization requires degree elevation. If only one of the two adjacent patches is reparametrized, we use the degree-elevated version of the curve for computing corresponding δs for both the patches.

[1] The maxima are computed coordinate-wise. See [KML96] for details.

6 Triangulation

While uniform tessellation of tensor-product patches is well understood, it is not clear what uniform tessellation of triangular patches means. Fig. 6 shows three possibilities, each of which generalizes in some sense to the concept of uniform sampling in the tensor-product case. Option (a) generates recursively nested triangles and does not require special interior processing for crack prevention. However, differing number of samples on the three boundary curves induce several special terminating conditions for the recursion. In addition, for incremental tessellation three rows of samples must be added for any update thus resulting in over-tessellation at the cost of rendering efficiency. Option (b) chooses the minimum of $\min\{\delta_u, \delta_v, \delta_w\}$ and generates several iso-parametric lines parallel to the corresponding axis, say u. The number of samples picked on each line goes down as v goes up: the step size is chosen to be smaller than $\delta_u(1-v)$. While the triangulation still is proportional to the number of samples for option (b), the constant grows large: There are multiple tests requires at each step. Option (c) is the most appropriate and the most efficient, even if it requires special handling near the line $w = 0$ to account for the fact that unlike the tensor-product case the grid does not necessarily align with $w = 0$. We use option (c) and describe it next.

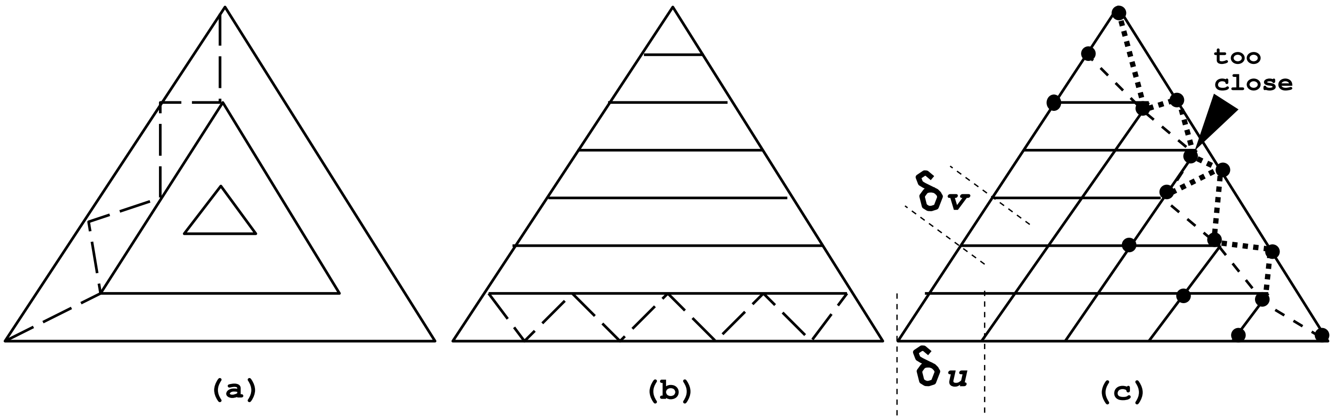


Figure 6: Uniform tessellation options

6.1 Uniform Tessellation

Recall that the step-size needed for the three boundaries are δ_1, δ_2 and δ_3, respectively and the interior step-sizes along the three directions are δ_u, δ_v and δ_w. For uniform tessellation, we pick the minimum two of these. Without loss of generality, assume $\delta_u \leq \delta_w$ and $\delta_v \leq \delta_w$. Also denote $\lceil \frac{1}{\delta_i} \rceil$ by n_i. The main steps of the algorithm are:

1. Pick n_i equi-spaced samples on the i^{th} boundary (see Fig. 7(a)).
2. Generate Offset curves $u = \frac{\delta_v}{2}$ and $v = \frac{\delta_u}{2}$ as shown in Fig. 7(a). Triangulate the strips between the two curves and the corresponding boundary curves: $u = 0$ and $v = 0$. The factor of $\frac{1}{2}$ is needed to guarantee bounds on edge lengths at the boundary.
3. Generate iso-parametric lines $u = (i + \frac{1}{2})\delta_v$ and $v = (j + \frac{1}{2})\delta_u$, $i > 1, j > 1$.
4. Iso-parametric lines are delimited by the offset curve and the boundary $w = 0$ (Fig. 7(b)).
5. Diagonalize each parallelogram generated in step 4 including the ones intersected by $w = 0$ (shown in darker dashes in Fig. 2(c)). The remaining polygon consists of a chain of points along a straight line and another chain monotonic with respect to that line. This polygon may be triangulated in linear time by sweeping along the line $w = 0$.

Note that it is possible for an interior sample to get arbitrarily close to axis $w = 0$ (see Fig. 6). Such cases can force long and thin triangles which are undesirable for smooth shading. We avoid such cases by discarding points that are closer than $\frac{\max(\delta_u, \delta_v)}{2}$ to the axis $w = 0$. (The triangle size bounds are still maintained.)

6.2 Incremental Triangulation

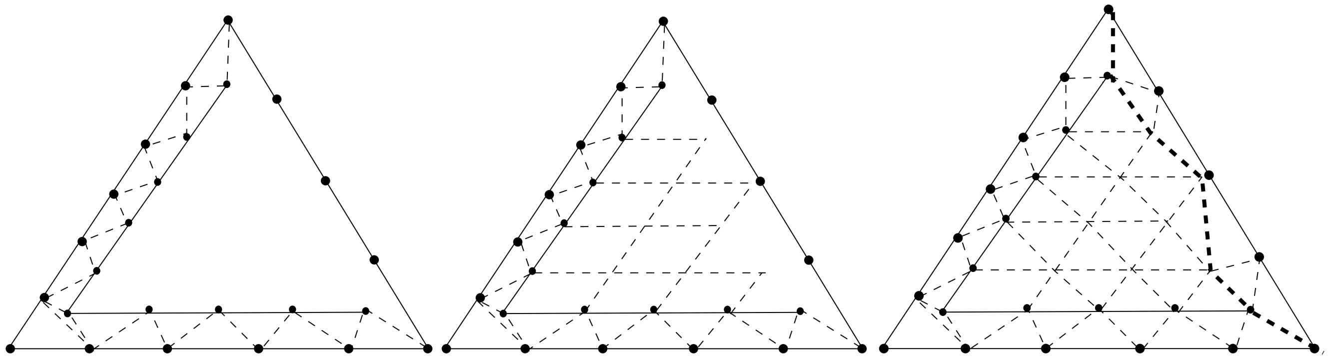
In most graphics simulations the view-point moves in a coherent manner. This implies that the sampling required for a guaranteed triangulation quality changes slowly. Indeed, more than 80% of the patches [Kum96] do not need to be re-triangulated in an average frame of a typical animation sequence. We take advantage of this property by adding or deleting only a row or a column of samples at a time. The triangles intersection with the event row or the event column are deleted and new triangles generated. Thus only a small update of triangulation is needed each frame. In fact, the incremental tessellation can be performed in O(# samples added or deleted). If trimming curves are present, this is the expected complexity [Kum96].

Like [KML96], our uniform tessellation algorithm has the property that a whole row or column of samples is updated at a time. Thus the cache management algorithm of [KML96] for incremental placement of event rows or columns for tensor-product surfaces extends directly to the triangular case. We add a row between two adjacent rows that have the maximum difference in the magnitude of their derivatives in the corresponding direction. Similarly, a row is deleted if the derivative difference is the smallest. Columns updates are analogous.

7 Implementation and Performance

We have presented an efficient algorithm for displaying Bézier triangles. The complexity of this algorithm is linear in the number of updates needed, which matches that of the algorithm to tessellate tensor-product patches [KML96]. The general pipeline of our algorithm is also similar to that of [KML96]. In addition, the visibility and coherence management schemes of [KML96] extend trivially to the triangular case due to the choice of our tessellation strategy.

We have implemented and tested our algorithm on a number of models. On an Indigo2 Maximum Impact with 195 MHz R10000 processor, we are able to render ten thousand Bézier triangles per second on average us-



(a) (b) (c)

Figure 7: Tessellation of the triangular domain

ing the full resolution 1024x1280 screen and a triangle-size bound of 50 pixels. This matches the rendering rates we achieved for the tensor-product patches. Dynamic tessellation achieves better frame rates for highly varying viewing transformation. On the other hand, employing static levels of detail can produce LODs with much fewer triangles for far-away models. Combining static and dynamic tessellations [KMZH97] can yield significant model size reduction and rendering speed-up over a large range of view-points. For further analysis of dynamic tessellation we refer the reader to [Kum96].

References

[AES91] S.S. Abi-Ezzi and L.A. Shirman. Tessellation of curved surfaces under highly varying transformations. *Proceedings of Eurographics*, pages 385–397, 1991.

[Cat74] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, University of Utah, 1974.

[DGHM93] T. DeRose, R. Goldman, H. Hagen, and S. Mann. Functional composition algorithm via blossoming. *ACM Transactions on Graphics*, 12(2):113–135, 1993.

[DMS92] W. Dahmen, C. Michelli, and H.P. Seidel. Blossoming begets B-spline bases built better by B-patches. *Mathematics of Computation*, 59(199):97–115, 1992.

[Far93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., 1993.

[Fil86] D. Filip. Adaptive subdivision algorithms for a set of Bézier triangles. *Computer-Aided Design*, 18(2):74–78, 1986.

[FK90] D.R. Forsey and V. Klassen. An adaptive subdivision algorithm for crack prevention in the display of parametric surfaces. In *Proc. Graphics Interface*, pages 1–8, 1990.

[IW92] K. Iino and D. Wilde. Subdivision of triangular Bézier patches into rectangular Bézier patches. *Transactions of the ASME*, 1992.

[Kaj82] J. Kajiya. Ray tracing parametric patches. *ACM Computer Graphics*, 6(3):245–254, 1982. (SIGGRAPH Proceedings).

[KM96] S. Kumar and D. Manocha. Hierarchical visibility culling for spline models. In *Proc. Graphics Interface*, pages 142–150, Toronto, Canada, 1996.

[KML95] S. Kumar, D. Manocha, and A. Lastra. Interactive display of large scale NURBS models. In *Proc. Symposium on Interactive 3D Graphics*, pages 51–58, Monterey, CA, 1995.

[KML96] S. Kumar, D. Manocha, and A. Lastra. Interactive display of large NURBS models. *IEEE Transactions on Visualization and Computer Graphics*, 2(4):323–336, Dec 1996.

[KMZH97] S. Kumar, D. Manocha, H. Zhang, and K. Hoff. Accelerated walkthrough of large spline models. In *Proc. Symposium on Interactive 3D Graphics*, pages 91–101, Providence, RI, 1997.

[Kum96] S. Kumar. *Interactive Rendering of Parametric Spline Surfaces.* PhD thesis, University of North Carolina, Department of Computer Science, 1996. Also available as UNC Technical Report TR96-039.

[LC93] W.L. Luken and Fuhua Cheng. Rendering trimmed NURB surfaces. Computer science research report 18669(81711), IBM Research Division, 1993.

[LCWB80] J.M. Lane, L.C. Carpenter, J. T. Whitted, and J.F. Blinn. Scan line methods for displaying parametrically defined surfaces. *Communications of ACM*, 23(1):23–34, 1980.

[MD94] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves i: simple intersections. *ACM Transactions on Graphics*, 13(1):73–100, 1994.

[NSK90] T. Nishita, T.W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *ACM Computer Graphics*, 24(4):337–345, 1990. (SIGGRAPH Proceedings).

[Pet84] C. Peterson. Adaptive contouring of three-dimensional surfaces. *Computer Aided Geometric Design*, 1:61–74, 1984.

[PR95] L. Piegl and A. Richard. Tessellating trimmed NURBS surfaces. *Computer Aided Geometric Design*, 27(1):16–26, 1995.

[RHD89] A. Rockwood, K. Heaton, and T. Davis. Real-time rendering of trimmed surfaces. *ACM Computer Graphics*, 23(3):107–117, 1989. (SIGGRAPH Proceedings).

[Sab77] M. Sabin. *The Use of Piecewise Forms for Numerical Representation of Shape.* PhD thesis, Hungarian Academy of Sciences, Budapest, 1977.