to appear in EUROGRAPHICS 2001 Proceedings
A. Chalmers and T.-M. Rhyne (Guest Editors)

COMPUTER GRAPHICS Forum, Volume 20 (2001), Number 3

Ray Tracing Triangular Bézier Patches

S. H. Martin Roth, Patrick Diezi†, Markus H. Gross

Computer Science Department
Swiss Federal Institute of Technology (ETH)
Zurich, Switzerland

email: {roth, grossm}@inf.ethz.ch
http://graphics.ethz.ch/

Abstract

We present a new approach to finding ray–patch intersections with triangular Bernstein–Bézier patches of arbitrary degree. This paper extends and complements on the short presentation [17]. Unlike a previous approach which was based on a combination of hierarchical subdivision and a Newton–like iteration scheme [21], this work adapts the concept of Bézier clipping to the triangular domain.

The problem of reporting wrong intersections, inherent to the original Bézier clipping algorithm [14], is investigated and opposed to the triangular case. It turns out that reporting wrong hits is very improbable, even close to impossible, in the triangular set–up. A combination of Bézier clipping and a simple hierarchy of nested bounding volumes offers a reliable and accurate solution to the problem of ray tracing triangular Bézier patches.

Keywords: Computer graphics, parametric surfaces, piecewise polynomials, ray tracing, Bézier clipping, Bernstein–Bézier patches, Chebyshev boxing

1. Introduction

Piecewise polynomial surfaces, or parametric free–form surfaces, have proven useful for representing objects in computer aided design and computer graphics (for an introduction, see [8]). In general, we have to distinguish two types of free–form surfaces: rectangular and triangular surfaces. Most modeling systems strictly rely on rectangular surface formulations since their definition as a tensor product extension of the univariate case is conceptually simpler. However, triangular formulations offer considerable advantages both topologically and analytically. Further, from a mathematical point of view, triangular Bézier patches in terms of barycentric coordinates are a more natural generalization of Bézier curves than tensor product patches. More and more, triangular surfaces are used for modeling complex geometries or to represent deformable models [7, 18]. Further, they lend themselves well to scattered data interpolation [1]. All those disciplines have a demand for an accurate visualization of the resulting surfaces.

Conceptually, there are two alternatives to render free-form surfaces: firstly, the conversion to a polygonal model and subsequent rendering of the resulting polygonal primitives or, secondly, direct ray tracing of the parametric surface description. The disadvantages of polygonalization are obvious. On the one hand, shading artifacts occur if the polygonalization is too coarse. On the other hand, visual effects like reflection and refraction as well as correct lighting is difficult or impossible to achieve. In contrast thereof, ray tracing, although being computationally more expensive, offers a means to accomplish all fore–mentioned effects. Further, ray tracing as a high quality rendering technique is well known and has been investigated thoroughly over the past decades (see e.g. [10]).

The basic requirement for ray tracing of objects, it be geometric primitives or a parametric surface, is the computation of intersections between a ray and the surface description. This task is referred to as the *ray–patch intersection problem.* This paper describes a new approach to this problem for triangular Bézier patches of arbitrary degree [6] based on the concept of Bézier clipping [14].

† Currently at ViewTec, Zurich, Switzerland
email: diezi@viewtec.ch

The paper is organized as follows: after an overview of previous work in Section 2, the extension of Bézier clipping to the triangular domain will be presented in Section 3. Section 4 will give some details about the problem of reporting wrong intersections inherent to the original clipping algorithm for tensor product formulations and investigate differences and advantages of triangular Bézier clipping. Section 5 will present results and comparisons to a simple approach based on nested bounding volumes.

2. Related Work

In general, the ray–patch intersection problem with free–form surfaces of arbitrary degree cannot be solved directly. As a consequence, one has to resort to an iterative computation of intersection points. Fournier and Buchanan [9] distinguish between two principle approaches: *geometric* and *parametric* intersection. Geometric intersection aims at finding the world coordinates of intersection points whereas parametric intersection determines their parametric coordinates. Most often, it is not only the intersection point itself one is interested in but also the local surface normal as well as shading and texture information. As far as free–form surfaces are concerned, all these requirements make it inevitable to know the world as well as the parametric location of an intersection. Further, as a matter of fact, knowing the parametric intersection coordinates automatically implies the corresponding point in 3D space whereas the contrary in general does not hold. As a consequence, with exception of early works [13, 19] emphasis in previous research as well as in this work has been put on parametric intersection methods.

Again, parametric intersection methods divide into two categories: *nested bounding volumes* in general followed by a root finding scheme such as Newton iteration and *parameter interval iteration*. These classes of algorithms will be revisited in the next two sections.

2.1. Nested bounding volumes

Approaches using nested bounding volumes basically compute a hierarchy of bounding volumes for every patch. In a preprocessing step each patch is hierarchically subdivided until the resulting sub–patches meet a certain stopping criterion, most often based on the flatness of the sub-patch. The computation of a ray–patch intersection now consists of a traversal of such a hierarchy guided by intersection tests between the ray and the bounding volumes on the respective hierarchy level. As soon as a leaf of the hierarchy is reached, the intersection between the ray and the leaf geometry is calculated. This is accomplished either by using Newton iteration, known to converge quickly on nearly planar surfaces, or by direct intersection of the ray with an approximating primitive.

One has to choose a suited bounding primitive providing for an optimal trade–off between tight enclosure and efficient intersection testing. Bounding spheres offering a very efficient intersection test [10] have been proposed as well as axes-aligned bounding boxes [22], oriented slabs [25], and parallelepipeds [2]. Especially the use of parallelipipeds is very suitable for Bézier patches as it takes advantage of their convex hull property. Another conceptually elegant and very efficient method, *Chebyshev boxing*, was introduced by Fournier and Buchanan in [9]. In order to find a hierarchy of enclosing bounding volumes they use the coefficients of the Chebyshev representation of bilinear patches approximating the sub–patches in the hierarchy. The actual point of intersection is finally found by intersecting interpolating bilinear patches in the leaves of the hierarchy with the ray. Obviously, such a procedure, as does any approach using an approximation of the surface in the leaves, requires solving the cracking problem between adjacent patches. Further, Chebyshev boxing can only deal with integral patches [5]. Therefore, Campagna et al. in [5] proposed an approach very similar to Chebyshev boxing which can handle rational patches, the *bounding volume hierarchy* (BVH). It computes nested bounding volumes using the Bézier representation. The actual intersection points are found either using Bézier clipping [14] (see section Section 2.2) on the sub–patches in the hierarchy leaves or by intersecting an interpolating bilinear sub–patch. In contrast to Chebyshev boxing, there is no need for calculating these sub–patches since the four corner control points of the corresponding Bézier sub–patch already form an interpolating bilinear patch.

2.2. Parameter interval iteration

In contrast to the fore–mentioned methods, parameter interval methods directly operate on the parametric domain by narrowing the candidate interval for an intersection. First approaches using multivariate Newton iteration employed interval arithmetic to overcome the inherent problem of finding a good starting point for the iteration [23]. More recently, Nishita et al. introduced the technique of Bézier clipping [14] which makes extensive use of the convex hull property of Bézier curves. Although being less computationally efficient than e.g. Chebyshev boxing, this approach is applicable to rational patches and, with slight improvements [4, 5], yields a robust and stable algorithm. In contrast to many nested bounding volume algorithms, Bézier clipping is guaranteed to find all intersections. Further, the memory usage of Bézier clipping is very low compared to nested bounding volume approaches, since only the control points of the patches have to be stored instead of hierarchical data structures and thousands of bilinear patches and bounding volumes.

In summary, in accordance with [5], we prefer Bézier clipping as a general choice for producing high–quality pictures

without artifacts in reasonable time and with few memory requirements. Chebyshev Boxing and BVH are still very useful for animation set–ups where the same scene may have to be rendered several times. A combination of a nested bounding volume approach together with Bézier Clipping seems very promising. On the one hand, it eliminates the first clipping iterations which in general are less efficient due to little clipping of curved patches. On the other hand, it avoids potential approximation artifacts.

2.3. Ray tracing triangular patches

Except for the early work on triangular Steiner patches in [19], to the author's knowledge only Stürzlinger addressed the problem of ray tracing triangular free–form surface patches [21]. Stürzlinger's approach must be attributed to the class of nested bounding volume algorithms using *tripipeds* (skewed triangular prisms) as bounding primitives and Newton iteration similar to [2]. Chebyshev boxing unfortunately does not directly generalize to the triangular domain. Although having been considered by Stürzlinger to be «not straightforwardly applicable to triangular surfaces», the next section presents an extension of Bézier clipping to the triangular domain.

3. Triangular Bézier Clipping

After a short introduction to triangular Bernstein–Bézier patches we in this section will present the Bézier clipping algorithm for triangular patches. We will point out to differences and analogies to the tensor product situation.

3.1. Triangular Bernstein–Bézier patches

A Bézier curve of degree n in the local coordinate u, $0 \le u \le 1$ is represented as

$$\mathbf{b}^n(u) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(u)$$

with control points $\mathbf{b}_i$ and the univariate Bernstein polynomials $B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$. For triangular surfaces and using barycentric coordinates as local coordinate system, the Bernstein polynomials generalize very naturally to

$$B_{ijk}^n(r,s,t) = B_{\mathbf{i}}^n(r,s,t) = \binom{n}{\mathbf{i}} r^i s^j t^k = \frac{n!}{i!\,j!\,k!} r^i s^j t^k$$

Correspondingly, a triangular Bézier surface of degree n is given by

$$\mathbf{b}^n(r,s,t) = \sum_{|\mathbf{i}|=n} \mathbf{b}_{\mathbf{i}} B_{\mathbf{i}}^n(r,s,t)$$

where $|\mathbf{i}| = n$ stands for $i+j+k = n$ on the assumption that $i, j, k \ge 0$. Figure 1 illustrates the situation on the example of a quadratic Bézier patch.

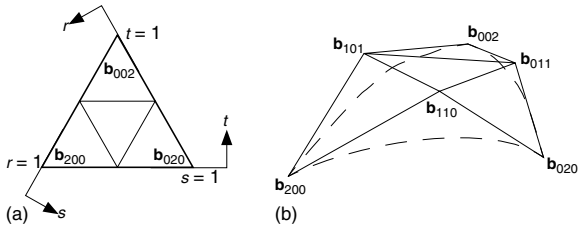

Figure 1: *A triangular quadratic Bézier patch:*
(a) barycentric parametric domain
(b) corresponding patch and control points

Please notice that although there are three barycentric coordinates we are dealing with the bivariate case due to the linear dependency of the third barycentric coordinate $t = 1 - r - s$. As a consequence, in the remainder of the paper we will adhere to a notation using only r and s and omitting t. A triangular patch therefore is of the form:

$$\mathbf{b}^n(r,s) = \sum_{j=0}^{n} \sum_{i=0}^{n-j} \mathbf{b}_{ij} B_{ij}^n(r,s) \qquad (1)$$

3.2. Ray–patch intersection problem

The ray–patch intersection problem refers to the task of finding the intersections of a ray

$$\mathbf{r}(u) = \mathbf{r}_0 + u \cdot \mathbf{r}_d, \qquad u \in \mathrm{R}^+, \qquad \|\mathbf{r}_d\| = 1$$

with a Bézier patch according to (1). As do [13, 14] we represent the ray as the intersection of two planes given by their normalized implicit equations

$$a_k x + b_k y + c_k z + e_k = 0, \qquad k \in \{0, 1\} \qquad (2)$$

with $a_k^2 + b_k^2 + c_k^2 = 1$ (see Figure 2).

In practice, the two planes are orthogonal. To this aim, we define the normals $\mathbf{n}_k = (a_k, b_k, c_k)^T$ and the distances to the origin e_k as

$$\mathbf{n}_0 = \frac{\mathbf{r}_0 \times \mathbf{r}_d}{\|\mathbf{r}_0 \times \mathbf{r}_d\|}, \qquad e_0 = -(\mathbf{n}_0 \cdot \mathbf{r}_0)$$

$$\mathbf{n}_1 = \frac{\mathbf{n}_0 \times \mathbf{r}_d}{\|\mathbf{n}_0 \times \mathbf{r}_d\|}, \qquad e_1 = -(\mathbf{n}_1 \cdot \mathbf{r}_0)$$

If $\mathbf{r}_0 \parallel \mathbf{r}_d$ we instead of $\mathbf{r}_0$ use a vector given by permuting the two biggest values in $\mathbf{r}_d$ in order to define $\mathbf{n}_0$.

3.3. Reduction to two dimensions

In complete analogy to [5, 14] the problem of finding an intersection

$$\mathbf{r}(u) = \mathbf{b}^n(r,s)$$

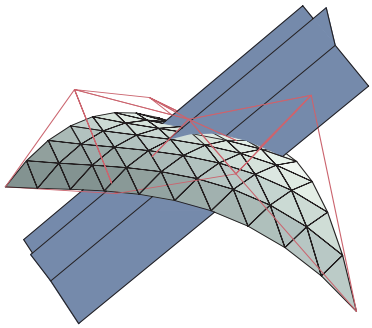

Figure 2: *Representation of a ray by two orthogonal planes (control net of patch indicated in red)*

can be reduced from three to two dimensions even if the triangular patch $\mathbf{b}^n(r,s)$ was rational. This is accomplished by substituting (1) into (2) which yields

$$\mathbf{d}^n(r,s) = \sum_{j=0}^{n}\sum_{i=0}^{n-j}\mathbf{d}_{ij}B_{ij}^n(r,s) \stackrel{!}{=} \begin{pmatrix}0\\0\end{pmatrix} \tag{3}$$

with $$\mathbf{d}_{ij} = \begin{pmatrix}a_0\\a_1\end{pmatrix}x_{ij} + \begin{pmatrix}b_0\\b_1\end{pmatrix}y_{ij} + \begin{pmatrix}c_0\\c_1\end{pmatrix}z_{ij} + \begin{pmatrix}e_0\\e_1\end{pmatrix} \tag{4}$$

and the coordinates x_{ij}, y_{ij}, and z_{ij} of the control points of the patch. The components 0 and 1 (in the remainder referred to as x and y) of $\mathbf{d}_{ij}$ geometrically represent the distance of the point to plane 0 and 1, respectively (see Figure 3). For rational patches, these distances are scaled by the weights. The problem now reduces to finding the roots of (3).

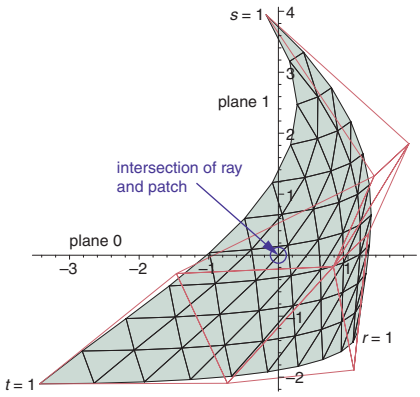

Figure 3: *Reduction of the ray–patch intersection problem to two dimensions (control net indicated in red)*

3.4. Finding intersections

Bézier clipping basically clips away regions in the parametric domain which are known not to intersect the patch. For

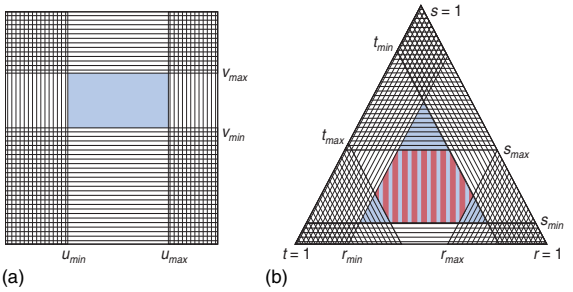

Figure 4: *Bézier Clipping in the parametric domain (candidate region highlighted in blue):*
(a) Rectangular (tensor product) domain
(b) Triangular (barycentric) domain

tensor product surfaces, Nishita et al. in [14] determine both (u_{min}, u_{max}) and (v_{min}, v_{max}) of the parametric candidate region for an intersection with the ray (see Figure 4a). Using these bounds, they subdivide the patch and iterate the procedure until the patch is small enough to satisfy a tolerance condition which assures sub–pixel accuracy.

A similar approach on the triangular domain of the barycentric coordinates r, s and t in general yields a complex non–triangular candidate region (see Figure 4b, red striped region). A triangular upper bound of this region can be found using only r_{min}, s_{min} and t_{min} (see Figure 4b, blue region). Subdividing with respect to this triangular domain and iterating the procedure similar to the tensor product case determines the parametric intersection. In order to find potential multiple intersections, a patch will be subdivided into four sub-patches if in one clipping iteration in r, s and t too little of a patch was to be clipped away (see Figure 13a). In our implementation, the following subdivision criterion proved to be a good choice

$$\text{subdivide if } r_{min} + s_{min} + t_{min} < 0.5$$

In the following, the procedure of finding r_{min} on the example of a cubic patch will be illustrated. The steps for s_{min} and t_{min} follow from symmetry.

Firstly, we determine a line L_r parallel to the vector from $\mathbf{d}_{00}$ to $\mathbf{d}_{0n}$ through the origin. This line can be seen as a linear approximation of the curve of constant r through the origin. Expressing this line in its implicit form

$$ax + by + c = 0, \qquad a^2 + b^2 = 1$$

yields the distances dr_{ij} of the control points $\mathbf{d}_{ij} = (x_{ij}, y_{ij})$ to the line L_r as

$$dr_{ij} = ax_{ij} + by_{ij} + c, \qquad 0 \le i + j \le n$$

with $c = 0$ since the line passes through the origin. Figure 5 clarifies the situation.

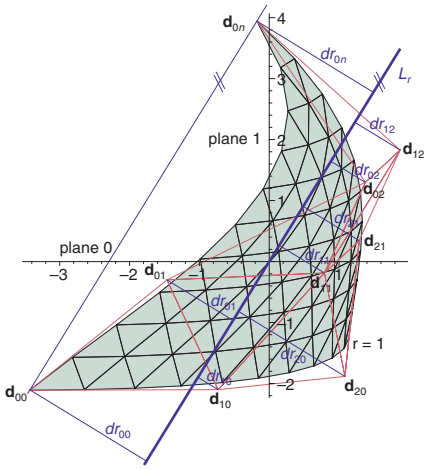

Figure 5: *Line L_r and corresponding distances dr_{ij} of control points of a cubic patch (control point d_{n0} is hidden)*

The distance $dr^n(r,s)$ of an arbitrary point to L_r consequently becomes

$$dr^n(r,s) = \sum_{j=0}^{n} \sum_{i=0}^{n-j} dr_{ij} B_{ij}^n(r,s)$$

The distance function $dr^n(r,s)$ can be regarded as a functional surface over the triangular domain as

$$\mathbf{dr}^n(r,s) = \sum_{j=0}^{n} \sum_{i=0}^{n-j} \mathbf{dr}_{ij} B_{ij}^n(r,s) = (r,s,dr^n(r,s)) \quad (5)$$

with $\mathbf{dr}_{ij} = (r_i, s_j, dr_{ij})$ and equidistant $r_i = i/n$ and $s_j = j/n$ for $0 \le i+j \le n$. Figure 6 illustrates the functional distance patch and the corresponding distances.

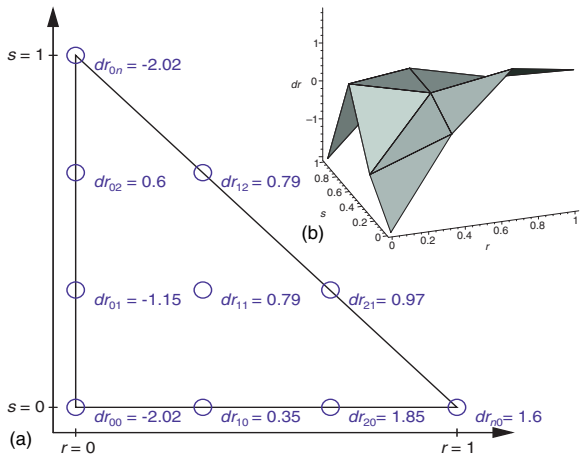

Figure 6: *Functional distance patch: (a) top view, (b) 3D view*

In a next step, the functional surface is projected along the L_r direction which corresponds to the $r = 0$ parametric line or s direction in Figure 6. Figure 7 shows the projected points and their convex hull.

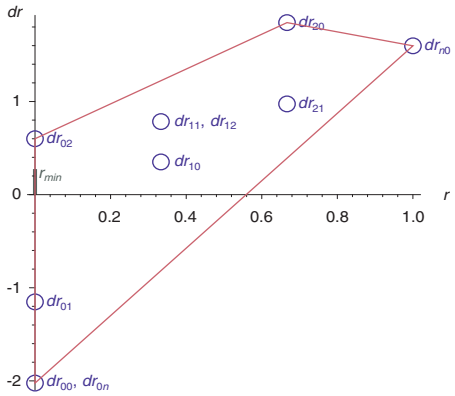

Figure 7: *Projection of functional L_r distance patch along the s direction, its convex hull and the resulting r_{min}*

The clipping value r_{min} can now be found by intersecting the convex hull of the projected distances with the r coordinate axis (see Figure 7). In this example r_{min} evaluates to zero. In the very same manner, a clipping value s_{min} can be found. Figure 8 shows the projected L_s distances and their convex hull as well as the resulting s_{min} value. For the determination of t_{min}, the convex hull projection direction is given by the line $t = 0$, which corresponds to the diagonal line in Figure 6.

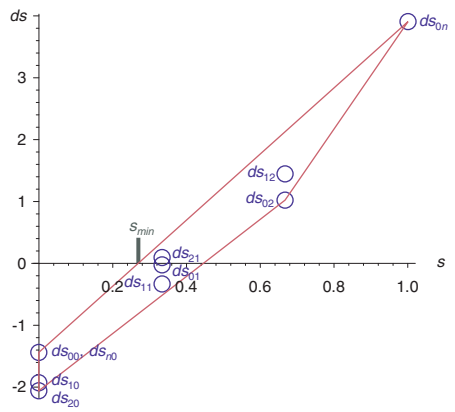

Figure 8: *Projection of functional L_s distance patch along the r direction, its convex hull and the resulting s_{min}*

For parametric values below those minima there cannot be an intersection due to the convex hull property of Bézier patches. Further, the ray does not intersect the patch if

$$r_{min} + s_{min} + t_{min} > 1$$

If the parametric candidate domain is small enough to meet the stopping criterion, e.g. its projected size in screen space drops below one pixel, the centroid of r_{min}, s_{min} and t_{min} is taken as the parametric point of intersection.

Otherwise, the patch is subdivided according to the minima found (see Figure 9). Subdivision of triangular Bézier patches with respect to three arbitrary internal points **r**, **s** and **t** was first introduced by Goldman [11] and clarified by Boehm and Farin [3]. We follow the more general equivalent notation of Seidel [20] and find the sub–patch control points $\mathbf{c}_{ij}$ with respect to the triangular sub–domain $(\mathbf{r}, \mathbf{s}, \mathbf{t})$ as

$$\mathbf{c}_{ij} = \mathbf{d}[\mathbf{r}^{\langle i \rangle}, \mathbf{s}^{\langle j \rangle}, \mathbf{t}^{\langle n-i-j \rangle}] \qquad (6)$$

with

$$\mathbf{r} = (1 - s_{min} - t_{min}, s_{min}, t_{min})$$
$$\mathbf{s} = (r_{min}, 1 - r_{min} - t_{min}, t_{min})$$
$$\mathbf{t} = (r_{min}, s_{min}, 1 - r_{min} - s_{min})$$

and **d** referring to the control points of the projected patch according to (4). The notation in square brackets in (6) is the so–called *blossoming principle* [8, 15, 16]. It can be interpreted as taking i de Casteljau steps with respect to **r**, followed by j steps with respect to **s** and $n-i-j$ steps with respect to **t**. The clipping procedure then continues on the resulting sub–patch as shown in Figure 9.

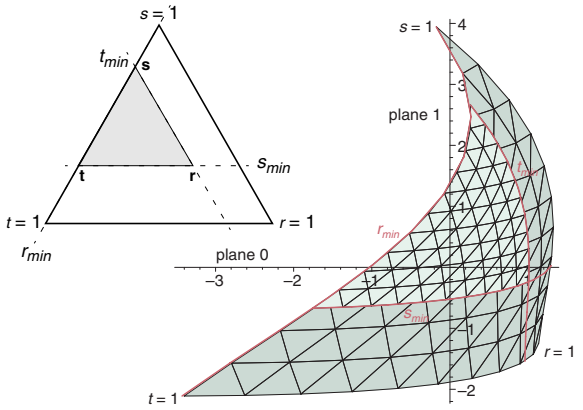

Figure 9: *Subdivision of patch according to clipping minima r_{min}, s_{min} and t_{min}*

3.5. Convex hull determination

As we have seen in the previous section, each clipping iteration requires finding the convex hull of three side views of different distance patches. Using Figure 8 as an example, we will investigate the procedure of efficient determination of such a convex hull and the corresponding value s_{min}.

In a first step, the maxima h_i and minima l_i for projected points of equal s parameter values are determined. The h_i and l_i define two polylines, P_h and P_l respectively (see Figure 10). Finding the convex hull of all points can now be divided into finding the *upper convex hull* from P_h and the *lower convex hull* from P_l. This can be accomplished using an iterative approach. It turns out, however, that a complete determination in general is not required [5].

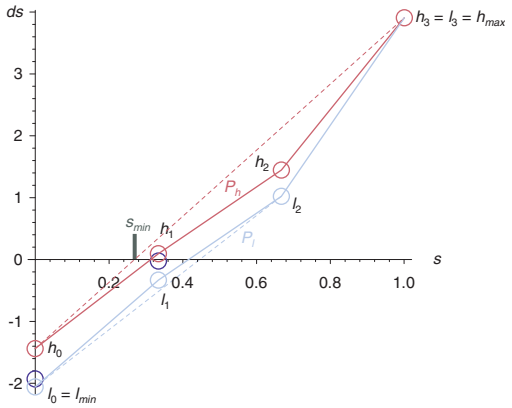

Figure 10: *Computation of upper and lower convex hull*

Before proceeding to further computations one should decide whether the s axis intersects the convex hull at all. To this aim, we compute the maximum h_{max} of all h_i and the respective minimum l_{min}. If $h_{max} < 0$ or $l_{min} > 0$ there is no intersection of the convex hull and the ray will not intersect the patch.

For all other cases, three situations have to be dealt with:

$h_0 < 0$: Only the upper convex hull of the points h_0 to h_{max} has to be determined.

$l_0 > 0$: Only the lower convex hull of the points l_0 to l_{min} has to be computed.

$h_0 \geq 0$: $s_{min} = 0$ (as is the case for r_{min} in Figure 7)

4. Reporting of Wrong Hits

The original Bézier clipping algorithm can report wrong intersections (see e.g. [4]). Nishita et al. in [14] proposed the following enlargement of the parametric candidate region in order to cope with what they considered numerical problems:

$$s_{min} = 0.99 \cdot s_{min}, \qquad s_{max} = 0.99 \cdot s_{max} + 0.01$$

Campagna and Slusallek in [4] showed that the problem is inherent to the algorithm and not due to numerical round–

off. They proposed both a more subtle enlargement of the candidate region and an extension of the algorithm which ensures correct results, unfortunately at the cost of additional computations. After a short description of the problem in the tensor product setting, we will show that a comparable situation in the set–up of triangular patches is very improbable, nay close to impossible.

In short terms, the error can occur whenever the convex hull of projected distances intersects the corresponding parameter axis even if the patch actually does not. If the candidate region computed due to this intersection happens to be very small, the iteration may terminate and report a wrong intersection if, by coincidence, the second parametric candidate region drops below the threshold, too.

Figure 11 illustrates the situation on the example of projected L_u distances. The slight intersection of the convex hull yields a small candidate region between u_{min} and u_{max}. At the same time, the convex hull of projected L_v distances converts to a line which results in a collapsing candidate domain in v. Thus, a potentially wrong intersection is reported. As we can see, the coincidence mentioned above is not very improbable as a collapsing domain results whenever the projection of the patch in one dimension is undistorted as in Figure 11 and the convex hull of projected distances consequently converts to a line. Obviously, the error can only occur for non–interpolating control points.

There are several reasons why a similar situation for triangular patches is difficult to construct. First and possibly most important, the algorithm only makes use of r_{min}, s_{min} and t_{min} but does not take into account the respective maxima. Thus, if the three minima in one iteration do not sum up to approximately 1 and therefore cause the iteration to stop, in the following clipping iteration steps the error is likely to disappear. This is due to the fact, that the control nets of subsequent subdivisions approximate the patch better and better. Second, distances are computed with respect to three coordinate directions, which due to the barycentric setting are linearly dependent and thus to some respect redundant. As a consequence of these two facts, wrong intersections can only occur if both of the following conditions are met:

- In at least one projective view of distances, one non interpolating control point lies above or below the respective axis and the others do not.
- The minima r_{min}, s_{min} and t_{min} accidentally sum up to approximately but not more than 1.

It is the second condition that makes triangular Bézier clipping far less error–prone. Figure 12 clarifies these conditions: The situation is very similar to the tensor product example in Figure 11. In the projective view of L_r distances one control points lies slightly above the r axis. Further, the convex hull of projected L_s distances converts to a line. The tensor product error conditions are hereby met. In the triangular case, due to the second of the above conditions, an erroneous intersection is reported only if in the third projective view t_{min} happens to evaluate accidentally to approximately $1 - r_{min} - s_{min}$.

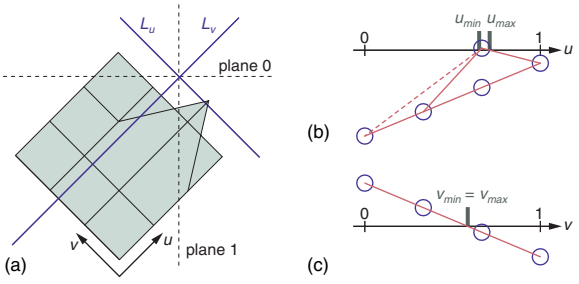

Figure 11: *Potential reporting of wrong intersections for Bézier clipping of a cubic tensor product patch:*
(a) reduced problem and lines L_u, L_v perpendicular to u,v parameter directions
(b) projection of L_u distances along v and corresponding small candidate region between u_{min} and u_{max}
(c) projection of L_v distances along u and corresponding collapsing candidate region $v_{min} = v_{max}$

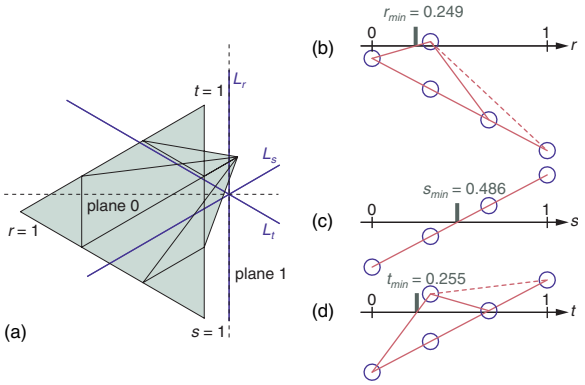

Figure 12: *Potential reporting of wrong intersections for Bézier clipping of a cubic triangular patch:*
(a) reduced problem and lines L_r, L_s and L_t
(b) projection of L_r distances and corresponding r_{min}
(c) projection of L_s distances and corresponding s_{min}
(d) projection of L_t distances and corresponding t_{min}

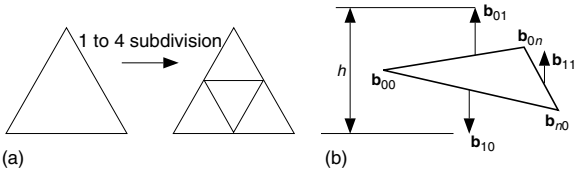

Figure 13: *(a) Subdivision of a patch for hierarchy build–up*
(b) Computing the height of a quadratic patch

5. Results

As a proof of concept we extended the object oriented ray tracer (OORT) of Wilt [24] to handle triangular Bézier patches. The object oriented design of this ray tracer makes it easy to integrate new types of objects. Unfortunately, its shading capabilities are limited to some extent.

To speed up the Bézier clipping algorithm we additionally implemented a *bounding sphere hierarchy* (BSH) which eliminates the first clipping iterations. Although bounding spheres are far from optimal with respect to tight enclosure of the patch, we have chosen spheres for their simplicity and fast intersection testing. The BSH is built by subdividing the triangular Bézier patch into four sub–patches until a flatness criterion is met (see Figure 13). We define the flatness of a patch as its height h. The height is computed as the extent of the control points along the normal of the plane spanned by the patch's corner nodes

$$h = \max\left\{0, \max_{0\le i+j\le n}(\mathbf{n}\cdot\mathbf{b}_{ij})\right\} - \min\left\{0, \min_{0\le i+j\le n}(\mathbf{n}\cdot\mathbf{b}_{ij})\right\}$$

with the normalized plane normal $\mathbf{n} = \bar{\mathbf{n}}/\|\bar{\mathbf{n}}\|$ and $\bar{\mathbf{n}}$

$$\bar{\mathbf{n}} = (\mathbf{b}_{0n} - \mathbf{b}_{00}) \times (\mathbf{b}_{n0} - \mathbf{b}_{00})$$

Finding intersections now consists of a BSH traversal and subsequent Bézier clipping on the leaf sub–patch.

In order to test the implementation, we converted the Utah teapot from 32 bicubic tensor product patches to 64 sextic triangular Bézier patches using the approach of Goldman and Filip [12]. On the one hand, patches of degree six are a demanding task for a ray tracer. On the other hand, the teapot geometry features patches of different curvature. Figure 14 shows the sextic control net and the corresponding ray traced image. The pairs of red and blue triangular patches represent the original bicubic rectangular patches.

Figure 15 shows a close–up of the knob. As a consequence of degenerated patches in the original model, the four red patches at the knob suffer from a collapsing triangle edge (seven control points coincide). In general, unlike rectangular models, a model made of triangular patches would not require degenerated edges. Such degeneracies need special treatment in the Bézier clipping algorithm since the line L is not defined for collapsing edges. In fact, we in such a situation determine L using the two adjacent control points on the non–degenerated edges.

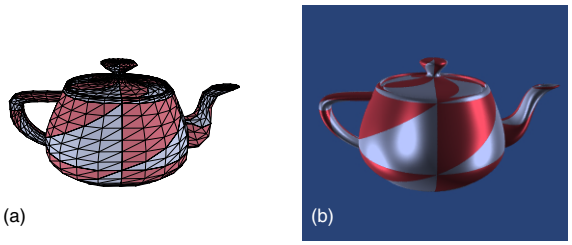

Figure 14: *(a) Control net of Utah teapot made of 64 triangular sextic Bézier patches*
(b) Corresponding ray traced image

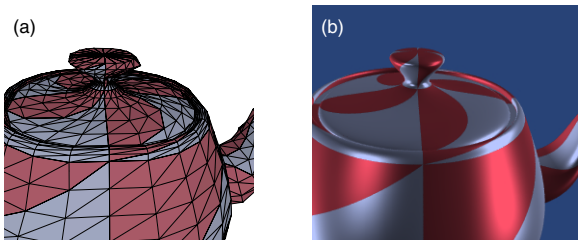

Figure 15: *(a) Close-up of control net at knob*
(b) Corresponding ray traced image

Due to the very straightforward implementation, it is difficult to make a quantitative performance analysis. What can be stated qualitatively is that Bézier clipping is clearly slower than a pure BSH based ray–patch intersection. Since the cost of the clipping operation grows with the cube of the patch's degree, ray tracing of sextic patches using Bézier clipping is roughly ten times slower than with pure BSH. Combining BSH and Bézier clipping, this ratio drops to about five. For patches of lower degree, the efficiency of Bézier clipping improves.

Figure 16: *Textured teapot*

6. Conclusions

We presented a new approach to ray tracing triangular Bézier patches using an extension of Bézier clipping. We have shown that the accuracy of triangular Bézier Clipping equals tensor product Bézier clipping while its reliability even excels the original algorithm. In combination with a hierarchy of nested bounding volumes, triangular Bézier clipping yields results of highest quality in reasonable time.

Future work includes the incorporation of trimming which can be done similar to the original work on Bézier clipping in [14]. Further, a more efficient implementation of the algorithm (e.g. as a POV–ray extension, http://www.povray.org/) will yield quantitative information about the performance of triangular Bézier clipping.

References

1. C. L. Bajaj, F. Bernardini, and G. Xu. "Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans." In *SIGGRAPH'95 Conference Proceedings*, Annual Conference Series, pages 109–118. ACM SIGGRAPH, Addison Wesley, 1995.
2. W. Barth and W. Stürzlinger. "Efficient Ray Tracing for Bézier and B-Spline Surfaces." *Computers & Graphics*, 17(4):423–430, 1993.
3. W. Boehm and G. Farin. "Letter to the Editor." *Computer-Aided Design*, 15(5):260–261, 1983.
4. S. Campagna and P. Slusallek. "Improving Bézier Clipping and Chebyshev Boxing for Ray Tracing Parametric Surfaces." In B. Girod, H. Niemann, and H.-P. Seidel, editors, *Proceedings of 3D Image Analysis and Synthesis '96*, pages 95–102, 1996.
5. S. Campagna, P. Slusallek, and H.-P. Seidel. "Ray Tracing of Spline Surfaces: Bézier Clipping, Chebyshev Boxing, and Bounding Volume Hierarchy – A Critical Comparison with New Results." *The Visual Computer*, 13(6):265–282, 1997.
6. G. Farin. "Triangular Bernstein-Bézier Patches." *Computer Aided Geometric Design*, 3(2):83–127, 1986.
7. G. Farin. "The Use of Triangular Patches in CAD." In M. Wozny, H. McLaughlin, and J. Encarnação, editors, *Geometric Modeling for CAD Applications*, pages 191–194. North-Holland, Amsterdam, 1988.
8. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1990.
9. A. Fournier and J. Buchanan. "Chebyshev polynomials for boxing and intersections of parametric curves and surfaces." In *COMPUTER GRAPHICS Forum*, Vol. 13, No. 3, pages 127–142. Eurographics, Basil Blackwell Ltd, 1994. EUROGRAPHICS'94 Conference.
10. A. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, 1989.
11. R. N. Goldman. "Subdivision Algorithms for Bézier Triangles." *Computer-Aided Design*, 15(3):159–166, 1983.
12. R. N. Goldman and D. J. Filip. "Conversion from Bézier Rectangles to Bézier Triangles." *Computer-Aided Design*, 19(1):25–27, 1987.
13. J. T. Kajiya. "Ray Tracing Parametric Patches." In *SIGGRAPH'82 Conference Proceedings*, Annual Conference Series, pages 245–254. ACM SIGGRAPH, Addison Wesley, 1982.
14. T. Nishita, T. W. Sederberg, and M. Kakimoto. "Ray Tracing Trimmed Rational Surface Patches." In *SIGGRAPH'90 Conference Proceedings*, Annual Conference Series, pages 337–345. ACM SIGGRAPH, Addison Wesley, 1990.
15. L. Ramshaw. "Blossoming: a Connect–the–Dots Approach to Splines." Technical report, Digital Systems Research Center, Palo Alto, CA, 1987.
16. L. Ramshaw. "Blossoms Are Polar Forms." *Computer Aided Geometric Design*, 6(4):323–359, 1989.
17. S. H. M. Roth, P. Diezi, and M. H. Gross. "Triangular Bézier Clipping." In *PACIFIC GRAPHICS 2000 Proceedings*, pages 413–414. IEEE, IEEE Computer Society, 2000.
18. S. H. M. Roth, M. H. Gross, S. Turello, and F. R. Carls. "A Bernstein-Bézier Based Approach to Soft Tissue Simulation." In *COMPUTER GRAPHICS Forum*, Vol. 17, No. 3, pages C285–C294. Eurographics, Blackwell Publishers Ltd, 1998. EUROGRAPHICS'98 Conference.
19. T. W. Sederberg and D. C. Anderson. "Ray Tracing of Steiner Patches." In *SIGGRAPH'84 Conference Proceedings*, Annual Conference Series, pages 159–164. ACM SIGGRAPH, Addison Wesley, 1984.
20. H. Seidel. "A General Subdivision Theorem for Bézier Triangles." In T. Lyche and L. L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 573–581. Academic Press, Inc., 1989.
21. W. Stürzlinger. "Ray Tracing Triangular Trimmed Free Form Surfaces." *IEEE Transactions on Visualization and Computer Graphics*, 4(3):202–214, 1998.
22. M. Sweeney and R. Bartels. "Ray-Tracing Free-Form B-Spline Surfaces." *IEEE Computer Graphics and Applications*, 6(2):41–49, 1986.
23. D. L. Toth. "On Ray Tracing Parametric Surfaces." In *SIGGRAPH'85 Conference Proceedings*, Annual Conference Series, pages 171–179. ACM SIGGRAPH, Addison Wesley, 1985.
24. N. Wilt. *Object-Oriented Ray Tracing*. Wiley, 1st edition, 1994.
25. J. Yen, S. Spach, M. Smith, and R. Pulleyblank. "Parallel Boxing in B-Spline Intersection." *IEEE Computer Graphics and Applications*, 11(1):72–79, 1991.