Robust and Numerically Stable Bézier Clipping Method for Ray Tracing NURBS Surfaces

Alexander Efremov* Vlastimil Havran† Hans-Peter Seidel‡
MPI Informatik, Saarbrücken, Germany

Abstract

Raytracing has become a popular method for generating realistic images and movies. The progress in hardware development shows that the real time raytracing on a single PC might be possible in the ongoing future. Obviously, that new generation of raytracing based applications will require more visualization precision and flexibility. Most of the modern raytracing based applications only deal with triangles as basic primitives, which brings limitations to an application and may cause visual artifacts to appear. NURBS surface representation is common for most of 3*D* modeling tools because of its compactness and useful geometric properties of NURBS surfaces. Using the direct raytracing NURBS surfaces, one can achieve better quality of rendered images. Although, many such approaches have already been presented, almost all of them suffer from numerical problems or do not work in some special cases. This paper presents a modified Bézier clipping method for finding ray - NURBS surface intersection points, which is fast, robust, and numerically stable.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing.

Keywords: ray tracing, NURBS surface, rational Bézier surface, Bézier clipping method.

1 Introduction

Raytracing has become a popular method for generating realistic images and movies. It allows to simulate a great variety of amazing light effects, including caustics, diffuse reflections, and soft shadows from environment maps.

Most of the modern raytracing based applications only deal with triangles as basic primitives because of their fast intersection test with a ray. But such restriction brings artifacts to the appearance of models. Even with the normal interpolation technique, visual artifacts along silhouette edges may still occur. This negative effect becomes more troublesome in the case of a close-up view on the surface silhouette.

Models for raytracing applications are usually supplied by 3*D* modeling tools, which predominantly work with NURBS surfaces. Therefore, a model has to be tessellated into triangles before being raytraced. During the tessellation process some shape information is lost. If the initial model is represented by trimmed NURBS surfaces, the tessellation process becomes more complicated and often results in artifacts along trimming contours [Guthe et al. 2002]. Moreover, the tessellation of complex shape models results in a large number of triangles, which require large amount of memory and preprocessing time for building acceleration data structures.

All these problems can be avoided if a method of direct raytracing NURBS surfaces is used. The problem, which arises here, is complexity of deterministic solutions for finding the nearest intersection point between a ray and a NURBS surface. Although the exact intersection is difficult or even impossible to obtain, especially for higher order surfaces, an approximate solution can be obtained using numerical procedures. Many different approaches for finding an approximate solution have been proposed [Lischinski and Gonczarowski 1990; Wang et al. 2000; Martin et al. 2000], which are mostly based on two methods: Newton's iteration method [Toth 1985] and Bézier clipping method [Nishita et al. 1990].

Newton's iteration method has found its application in many research areas for solving a great variety of problems. However, it has one significant drawback - its convergence to the root depends on the initial guess, which must lie in the vicinity of the root. If the initial guess is not chosen properly, Newton's iteration method may converge to a wrong intersection or not converge at all. An approach for identifying regions of parameter space of a surface in which Newton's iteration method is guaranteed to converge to a unique solution has been proposed [Toth 1985], but it is very time consuming, and therefore not practical. Some other improvements have also been applied [Barth and Stürzlinger 1993; Qin et al. 1997] to reduce the probability of reporting wrong intersections, but neither of them gives 100% guarantee of the right convergence. Therefore, visual artifacts can still be observed, especially along the surface silhouettes.

Bézier clipping method does not suffer from the initial guess problem. Moreover, it can compute all intersections of a ray with a NURBS surface if multiple intersections exist. However, it has some bottlenecks, which are highlighted in this paper. This paper presents a modified Bézier clipping method for finding ray - NURBS surface intersection points, which is fast, robust, and numerically stable.

*e-mail: aefremov@mpi-sb.mpg.de
†e-mail: havran@mpi-sb.mpg.de
‡e-mail: hpseidel@mpi-sb.mpg.de

1.1 Outline

The content of this paper is organized as follows. Section 2 recalls basic ideas behind the NURBS to rational Bézier conversion. Section 3 describes Bézier clipping method originally proposed by Nishita et al. [1990]. Problems of the original Bézier clipping method, which have been detected by Campanga et al. [1997], are described in Section 4. An improved Bézier clipping method, which is fast, robust, and numerically stable, is described in Section 5. Section 6 presents the achieved results, and future work is discussed in Section 7.

2 From NURBS to Rational Bézier Representation

When dealing with raytracing nonstandard primitives, like parametric surfaces, we are mostly interested in such procedures as surface subdivision, point evaluation, partial derivatives, and normal evaluations. As NURBS surfaces do not have fast evaluation procedures, they are not suitable for the direct raytracing. Fortunately, any NURBS surface can be represented by the number of rational Bézier patches without losing accuracy. Evaluation procedures of rational Bézier patches are mostly based on fast *de Casteljau algorithm* for surface subdivision. Therefore, we convert NURBS surfaces into rational Bézier patches during the preprocessing stage of raytracing.

A NURBS surface $S(u,v)$ of degree p in u parameter direction and degree q in v parameter direction defined of the parameter domain $[a,b]\times[c,d]$ is represented in a $3D$ (x,y,z) coordinate space by equation

$$S(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}N_i^p(u)N_j^q(v)w_{i,j}^e\bar{e}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}N_i^p(u)N_j^q(v)w_{i,j}^e} \tag{1}$$

where $a\le u\le b,\ c\le v\le d,\ \bar{e}_{i,j}\in R^3$ are *control points*, and $w_{i,j}^e\in R$ are *weights* assigned to the corresponding control points. The control points form a *control mesh* of the surface. The most important property of NURBS surfaces in the context of the presented paper is the *convex hull property* – the whole NURBS surface is completely enclosed in the convex hull of its control mesh.

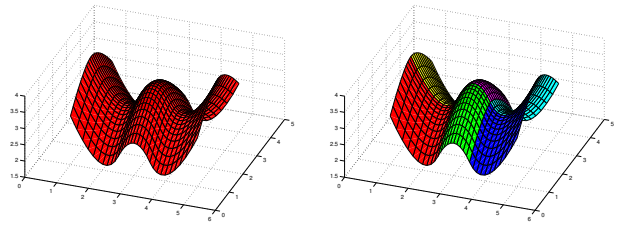
NURBS surfaces are defined on two *knot vectors*, which correspond to u and v parameter directions:

$$\begin{aligned}U&=\{\underbrace{a,\ldots,a}_{p+1},u_{p+1},\ldots,u_{r-p-1},\underbrace{b,\ldots,b}_{p+1}\}\\V&=\{\underbrace{c,\ldots,c}_{q+1},v_{q+1},\ldots,v_{s-q-1},\underbrace{d,\ldots,d}_{q+1}\}\end{aligned} \tag{2}$$

where $r=n+p+1$ and $s=m+q+1$. Notice that the control mesh is formed by $(n+1)\times(m+1)$ control points.

$N_i^p(u)$ and $N_j^q(v)$ are *B-spline basis functions* of degree p and q respectively. Definition and properties of B-spline basis functions as well as other useful information about NURBS surfaces can be found in [Piegl and Tiller 1995].

In order to represent a NURBS surface by the number of rational Bézier patches we have to modify the knot vectors in such a way that each knot has multiplicity $p+1$ in U knot vector and multiplicity $q+1$ in V knot vector. This can be done using the *knot insertion* or the *knot refinement* algorithms, whose description and efficient implementation framework can be found in [Piegl and Tiller 1995].



(a) (b)

Figure 1: NURBS to rational Bézier conversion. (a) Initial NURBS surface. (b) Rational Bézier patches obtained after changing the representation.

After the modification of the knot vectors, the number of control points in u and v parameter directions increases proportionally to the number of inserted knots, and each parameter subdomain $[u_k,u_{k+1}]\times[v_t,v_{t+1}]$ of not coinciding subsequent knots u_k and u_{k+1} in u direction and v_t and v_{t+1} in v direction represents a rational Bézier patch of the form

$$S(u,v)=\frac{\sum_{i=0}^{p}\sum_{j=0}^{q}B_i^p(u)B_j^q(v)w_{i,j}^p\bar{p}_{i,j}}{\sum_{i=0}^{p}\sum_{j=0}^{q}B_i^p(u)B_j^q(v)w_{i,j}^p} \tag{3}$$

where $\bar{p}_{i,j}=\bar{e}_{k-(p-i),t-(q-j)}$ are the control points and $w_{i,j}^p=w_{k-(p-i),t-(q-j)}^e$ are the weights.

Notice that each rational Bézier patch is defined on its own parameter domain determined by $u\in[u_k,u_{k+1}]$ and $v\in[v_t,v_{t+1}]$. $B_i^p(u)$ and $B_j^q(v)$ are *Bernstein basis functions* of degree p and q respectively, whose definition and properties can be found in [Piegl and Tiller 1995]. Figure 1 shows an example of a NURBS surface and the corresponding rational Bézier patches obtained after changing the representation. Notice that the convex hull property also holds for rational Bézier patches, i.e., each rational Bézier patch is completely enclosed in the convex hull of its control mesh.

3 Original Bézier Clipping Method

Bézier clipping method was originally introduced by Nishita et al. [1990]. This section explains in detail the ideas behind the method. The detailed explanation is necessary to better understand improvements of the method, which are described in Section 5. An interested reader is advised to access the original paper [Nishita et al. 1990].

Suppose we have a ray defined by its parametric equation

$$R(t)=\bar{o}+t\bar{d} \tag{4}$$

where $\bar{o}\in R^3$ is the ray origin and $\bar{d}\in R^3$ is the ray direction. Suppose we are looking for intersection points between the ray and a rational Bézier patch in a $3D$ (x,y,z) coordinate space given by equation

$$S(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(u)B_j^m(v)w_{i,j}^p\bar{p}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(u)B_j^m(v)w_{i,j}^p} \tag{5}$$

where $\bar{p}_{i,j}\in R^3$ are the control points, $w_{i,j}^p\in R$ are the weights, $a\le u\le b,\ c\le v\le d$. Bézier clipping method consists of several steps, which are explained below.

Step 1. We want to represent the ray as an intersection of two orthogonal planes further denoted by P_1 and P_2 as shown in Figure 2 (a). In order to define these planes we have to define their normal vectors. Suppose, $\bar{n}_1 = (a_1, b_1, c_1)$ and $\bar{n}_2 = (a_2, b_2, c_2)$ are normal vectors of the first and the second plane respectively. Then we can represent the ray as an intersection of two planes given by their implicit equations

$$\begin{aligned} P_1 &: \; a_1x + b_1y + c_1z + \underbrace{(-a_1o_x - b_1o_y - c_1o_z)}_{d_1} = 0 \\ P_2 &: \; a_2x + b_2y + c_2z + \underbrace{(-a_2o_x - b_2o_y - c_2o_z)}_{d_2} = 0 \end{aligned} \tag{6}$$

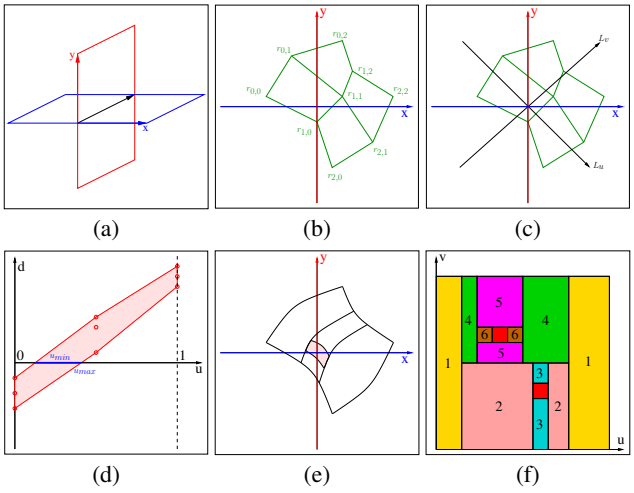


Figure 2: Original Bézier clipping method. (a) Representing a ray as an intersection of two orthogonal planes. (b) An example of a projected patch. (c) Determining $\bar{L}_u$ and $\bar{L}_v$. (d) A clipping iteration in u direction. (e) The projected patch segment after two clipping iterations. (f) An example of multiple intersections.

We can compute the normal vectors by rotation of the initial ray direction vector $\bar{d}$ by 90°:

$$\begin{aligned} \bar{n}_1 &= (a_1, b_1, c_1) = (-d_y, d_x, 0) \\ \bar{n}_2 &= (a_2, b_2, c_2) = (0, -d_z, d_y) \end{aligned} \tag{7}$$

Step 2. The intersection of plane P_k and the rational Bézier patch can be represented by substituting Equation (5) into Equation (6) and clearing the denominator

$$h^k(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u) B_j^m(v) h_{i,j}^k = 0 \tag{8}$$

where $h_{i,j}^k = w_{i,j}^p (a_k p_{x_{i,j}} + b_k p_{y_{i,j}} + c_k p_{z_{i,j}} + d_k) \in R$, $k = [1,2]$.

Note that $S(u,v)$ lies on the plane k iff $h^k(u,v) = 0$. Note also that $h_{i,j}^k$ is the weighted distance from control point $p_{i,j}$ to plane k. We can now project the surface into a $2D$ (x,y) coordinate space by taking the projected point coordinates to be

$$\bar{r}_{i,j} = (x_{i,j}, y_{i,j}) = (h_{i,j}^1, h_{i,j}^2) \in R^2 \tag{9}$$

The projected surface is then defined by equation

$$R(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u) B_j^m(v) \bar{r}_{i,j} \tag{10}$$

In this projection, the plane P_1 becomes the x axis, the plane P_2 becomes the y axis, and the ray projects to the coordinate system origin. Figure 2 (b) shows an example of a projected surface $R(u,v)$. The ray-patch intersection problem now becomes one of finding

$$\{(u,v) | R(u,v) = 0; \;\; a \le u \le b; \;\; c \le v \le d\}. \tag{11}$$

Step 3. We determine two normalized vectors $\bar{L}_u$ and $\bar{L}_v$ in the $2D$ (x,y) coordinate space in such a way, that $\bar{L}_u$ represents the u direction of parameterization of the projected patch and $\bar{L}_v$ represents the v direction of parameterization of the projected patch as shown in Figure 2 (c):

$$\begin{aligned} \bar{L}_u &= \left[(\bar{r}_{2,0} - \bar{r}_{0,0}) + (\bar{r}_{2,2} - \bar{r}_{0,2})\right] \\ \bar{L}_v &= \left[(\bar{r}_{0,2} - \bar{r}_{0,0}) + (\bar{r}_{2,2} - \bar{r}_{2,0})\right] \end{aligned} \tag{12}$$

Such a choice of $\bar{L}_u$ and $\bar{L}_v$ vectors leads to faster convergence of Bézier clipping method. Now, in order to solve Equation (11), we alternate the vectors $\bar{L}_u$ and $\bar{L}_v$ and iterate on the following steps (for brevity we consider only one iteration with the $\bar{L}_u$ vector).

Step 4. We have to determine the signed distances $d_{i,j} \in R$ from the projected control points $\bar{r}_{i,j}$ to $\bar{L}_u$. This gives a Bézier representation of the "distance to $\bar{L}_u$" function that can be used to determine the distance of an arbitrary point on the projected patch to $\bar{L}_u$:

$$d(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u) B_j^m(v) d_{i,j} \tag{13}$$

Step 5. Using the following property for evenly spaced control parameters $c_i = \frac{i}{n}$:

$$\sum_{i=0}^{n} B_i^n(u) c_i = u \tag{14}$$

we can define an *explicit patch* in a $2D$ (u,d) coordinate space, whose control points $\bar{D}_{i,j} = (u_{i,j}, d_{i,j}) \in R^2$ are evenly spaced in u parameter direction: $u_{i,j} = \frac{i}{n}$. A point on the explicit patch has coordinates

$$D(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u) B_j^m(v) \bar{D}_{i,j} = (u, d(u,v)) \tag{15}$$

Step 6. We plot the control points $\left(\frac{i}{n}, d_{i,j}\right)$ of the explicit patch in the $2D$ (u,d) diagram. This is shown in Figure 2 (d).

Step 7. We determine the convex hull of the points in this diagram and intersect it with the u axis. This gives us an *interval of interest* $[u_{min}, u_{max}]$. We conclude that $d(u,v) \neq 0$, and therefore $R(u,v) \neq 0$, for regions $u < u_{min}$ and $u > u_{max}$.

Step 8. The de Casteljau subdivision algorithm is applied to the projected patch to clip away these regions, leaving a patch segment, which corresponds to the interval $[u_{min}, u_{max}]$. This is referred to as the *Bézier clipping* in u.

Note that $[u_{min}, u_{max}] \subseteq [0,1]$ is the interval of interest of a patch segment on the current iteration. In order to determine the corresponding interval in the domain of the initial projected patch, we have to apply the following coordinate mapping on each iteration in u direction:

$$\begin{aligned} u_{min,d} &= u_{min,d} + (u_{max,d} - u_{min,d}) \cdot u_{min} \\ u_{max,d} &= u_{min,d} + (u_{max,d} - u_{min,d}) \cdot u_{max} \end{aligned} \tag{16}$$

where $[u_{min,d}, u_{max,d}]$ initially is the parameter domain of the projected patch in u direction, i.e., $[a,b]$. The v parameter direction is handled analogously.

Now we need to switch directions ($\bar{L}_u \to \bar{L}_v \to \bar{L}_u \to \cdots$) and repeat Steps 4 to 8. The parametric coordinates of intersection points (if an intersection exists) must be within the parameter subdomain $[u_{min,d}, u_{max,d}] \times [v_{min,d}, v_{max,d}]$. Figure 2 (e) shows the projected patch $R(u,v)$ after two clipping iterations. The Bézier clipping method terminates in two cases:

1. the convex hull on the corresponding iteration does **not** intersect the appropriate axis – this indicates that there is **no** intersection of $S(u,v)$ with the ray $R(t)$;
2. the size of the box in each parameter direction is smaller than a threshold value ε – in this case the intersection (u_{int}, v_{int}) is assumed to exist in the center of the current parameter subdomain:

$$u_{int} = \frac{1}{2}\left(u_{min,d} + u_{max,d}\right)$$
$$v_{int} = \frac{1}{2}\left(v_{min,d} + v_{max,d}\right) \qquad (17)$$

Note that if the size of the parameter subdomain in one direction is smaller than ε, this direction is not considered any more, i.e., only the second direction is proceeded iteratively without the alternation. Note also that in this case it is not absolutely necessary to apply Bézier clipping in this direction, and the second direction may be considered either with the same patch segment or with the clipped one.

If multiple intersections exist, Bézier clipping method does not converge to a single solution. Therefore, if a Bézier clipping fails to reduce the parameter interval width by at least 20%, we have to split the current patch segment in half in the direction of the current iteration. The case of multiple intersections is illustrated in Figure 2 (f). First, Bézier clipping in u discards regions 1. In attempting to clip in v, it turns out that $v_{max} - v_{min} > 0.8$. Therefore, the remaining domain is subdivided in half at $v = 0.5$. Two patch segments can be proceeded iteratively. The regions 2 of the first patch segment are discarded after one more iteration in u domain. Without further subdivision we can compute the intersection, which lies between regions 3 within tolerance. Two more iterations are needed to proceed the second patch segment. After clipping away the regions 4 and 5, one more intersection, which lies between regions 6 within tolerance, can be computed without further subdivision.

4 Detected Problems

The original Bézier clipping method does not suffer from the initial guess problem as Newton's iteration method does, and is therefore more suitable for finding ray-rational Bézier patch intersection points. However, it still suffers from the problem of reporting wrong intersections, which has been detected by Campanga et al. [1997]. An example of this problem is illustrated in Figure 3.

Figure 3 (a) shows an example of a projected rational Bézier patch and the calculated vector $\bar{L}_u$. One of the patch control points lies slightly above the vector and the others lie below. After constructing the $2D$ (u,d) diagram (see Section 3, Step 6), we get very small interval of interest as shown in Figure 3 (b). We stop to consider the u direction and start new iteration in v direction. We can consider the v direction either with the same patch segment or with the clipped one. For brevity we consider only the second case with

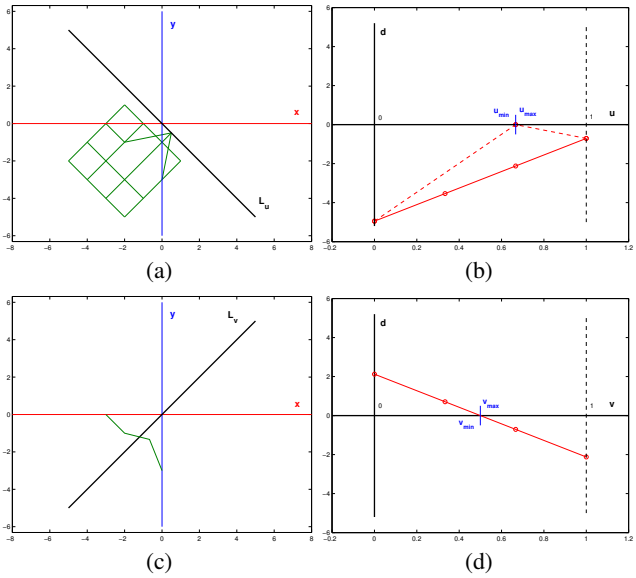


Figure 3: Problem of reporting wrong intersections. (a) The initial projected patch. (b) The $2D$ (u,d) diagram. (c) The patch segment after the Bézier clipping in u. (d) The $2D$ (v,d) diagram.

the clipped patch segment, but the problem of wrong intersections arises in both cases. Figure 3 (c) shows the clipped patch segment obtained after Bézier clipping in u direction together with the vector $\bar{L}_v$. Figure 3 (d) shows the corresponding $2D$ (v,d) diagram. Very small interval of interest in v direction leads to reporting a wrong intersection.

The reason for reporting wrong intersections is in the fact, that both vectors $\bar{L}_u$ and $\bar{L}_v$ can intersect the projected patch, but these intersections may be not in the vicinity of the point of origin $(0,0)$. Several suggestions of how to avoid this problem have been proposed [Campagna et al. 1997]. As the improved version of the original Bézier clipping method proposed in Section 5 does not suffer from this problem, detailed explanations are skipped. We refer interested readers to the original paper [Campagna et al. 1997].

5 Robust and Numerically Stable Bézier Clipping Method

This section describes novel improvements of the original Bézier clipping method proposed by Nishita et al. [1990]. First, the *multiple equivalent intersections problem* is described in Subsection 5.1. The reasons for necessity to change the coordinate space of the termination criteria ε are described in Subsection 5.2 together with related modifications of the method. An efficient view dependent computation of the new termination criteria ε for primary rays (rays shot from the virtual camera) is described in Subsection 5.3. A problem of computing the vectors $\bar{L}_u$ and $\bar{L}_v$ and an efficient modification of their computation are described in Subsection 5.4. Subsection 5.5 shows how an acceleration data structure can improve the performance of Bézier clipping method.

5.1 Multiple Equivalent Intersections Problem

The original Bézier clipping method has a problem, which has not been detected by any of researchers so far. Let us call this prob-

lem the *multiple equivalent intersections problem* – sometimes the method may result in thousands of intersections, which actually correspond to the same intersection point. Let us consider an example in Figure 4.

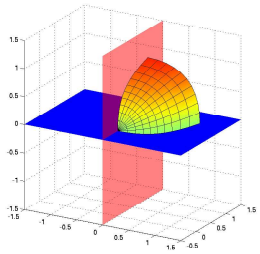


Figure 4: Testing a ray against a sphere patch.

Suppose that a ray is tested against a section of a sphere, which is represented by a rational Bézier patch and has the following mesh of control points and weights:

$$\begin{aligned} p_{00} &= (0,0,0),\ p_{01} = (0,0,1),\ p_{02} = (0,1,1); \\ p_{10} &= (0,0,0),\ p_{11} = (1,0,1),\ p_{12} = (1,1,1); \\ p_{20} &= (0,0,0),\ p_{21} = (1,0,0),\ p_{22} = (1,1,0); \\ w_{00} &= 1,\ w_{01} = \sqrt{2}/2,\ w_{02} = 1; \\ w_{10} &= 1,\ w_{11} = \sqrt{2}/2,\ w_{12} = 1; \\ w_{20} &= 1,\ w_{21} = \sqrt{2}/2,\ w_{22} = 1. \end{aligned} \tag{18}$$

The ray is represented as an intersection of two planes, which are shown in Figure 4. After projecting this problem in the $2D$ (x,y) coordinate space, we obtain a projection, which is shown in Figure 5 (a). Suppose that the initial rational Bézier patch is defined on the parameter domain $[0,1]\times[0,1]$. Then, after determining distances from its control points to the vector $\bar{L}_u$, we obtain a convex hull, which is shown in Figure 5 (b). Apparently, the interval of interest (the intersection with the u axis) is less than ε. This means that we stop to consider the u direction and continue to consider iteratively only the v direction. Notice that if we now applied Bézier clipping in u direction, we would obtain a very small patch segment (actually it is almost one point). So, it would be a good idea to stop the iterations at this point and report the intersection, because the obtained patch segment is already negligible in the $2D$ (x,y) coordinate space. But the method does not have any termination criteria like this and the next iteration is executed.

Notice that next iterations cause the multiple equivalent intersections problem. Suppose for simplicity that after obtaining the interval of interest in u direction, we do not subdivide the initial projected patch and consider the same one in v direction. If we subdivide the patch, the scale of figures is much smaller, but the problem remains. After constructing the convex hull in v direction (see Figure 5 (c)), we see that the size of the v axis span is almost equal to 1. So, the multiple intersections are supposed to exist, and the initial patch is subdivided in half in v direction as shown in Figure 5 (d). One subpatch is now defined on the v interval $\left[0,\frac{1}{2}\right]$ and another one is defined on the v interval $\left[\frac{1}{2},1\right]$.

After the subdivision, each half of the patch is considered separately in v direction only. But Figures 5 (e) and (f) show that after the convex hulls construction the problem remains. So, the both halves of the patch are assumed to have multiple intersections too and are subdivided once more in half in v direction. This gives us

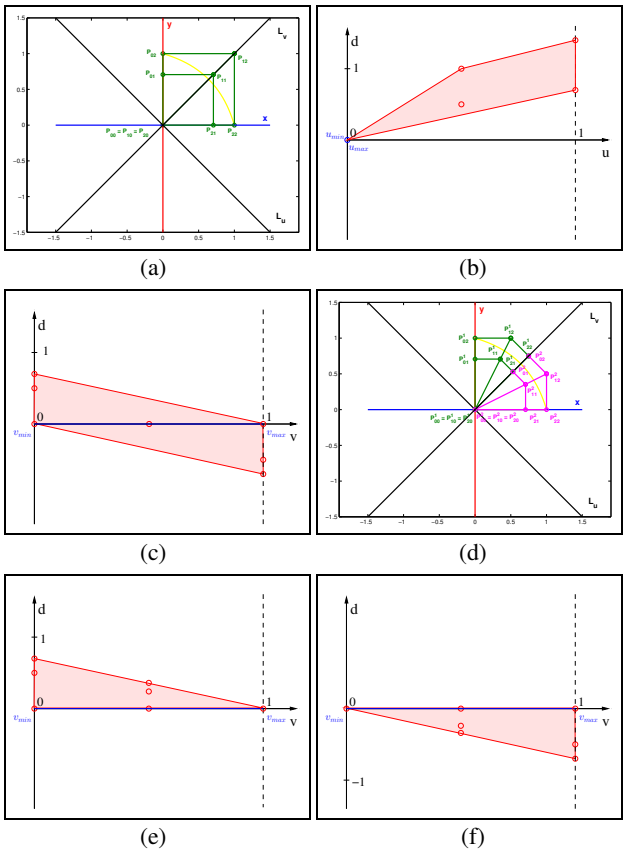


Figure 5: Multiple equivalent intersections problem. (a) The initial projected patch. (b) The $2D$ (u,d) diagram. (c) The $2D$ (v,d) diagram. (d) The patch segments after the Bézier clipping in v. (e) $2D$ (v,d) diagram for the first patch segment. (f) $2D$ (v,d) diagram for the second patch segment.

four patch segments to be considered, which are defined on the v intervals $\left[0,\frac{1}{4}\right]$, $\left[\frac{1}{4},\frac{1}{2}\right]$, $\left[\frac{1}{2},\frac{3}{4}\right]$, and $\left[\frac{3}{4},1\right]$.

Two of these patch segments report intersections on the next iteration in v direction. Another two patch segments are subdivided in half once more, and so on. This process of subdivision repeats many times until the size of the parameter domain of each patch segment "adjacent to $\bar{L}_v$" is very small in v direction (less than ε). As the result, many intersections are reported, which actually correspond to the same intersection point.

This shows that a special procedure has to be applied when processing a new intersection. In order to do it efficiently, we can store the intersections in a sorted list (which is sorted by distances to the ray origin). When a new intersection is found, we locate a position in the sorted list, where the intersection has to be inserted, and compare the new intersection with two (at most) neighbor intersections in the list. If the difference in distances is less than the value ε, the new intersection is discarded, otherwise the new intersection is inserted in the sorted list. We implemented this approach as outlined in the following pseudocode:

```
void PROCESS_INTS(ints_list, ints)
{
  /*  Process a new intersection  */
  /*
    Input: @ints_list - the list of all found intersections;
      @ints - a new intersection to be processed;
    Output: modified (is necessary) @ints_list;
  */

  pos = ints_list.Locate(ints);
  if (pos > 0)
    if (fabsf(ints.dist - ints_list[pos - 1].dist) < EPSILON)
      return;
  if (pos < ints_list.size())
    if (fabsf(ints.dist - ints_list[pos].dist) < EPSILON)
      return;
  ints_list.Insert(pos, ints);
}
```

We have found out that the problem of multiple equivalent intersections occurs often in practice for available real world models. Notice that depending on the threshold value ε, one may compute thousands of unnecessary intersections. It can significantly increase the computation time. Although the problem of multiple equivalent intersections cannot be avoided in general, using termination criteria in the $2D$ (x,y) coordinate space of a projected patch rather than in its $2D$ (u,v) parameter space, can highly decrease the number of such intersections. This idea is described in the next subsection.

5.2 Changing the Coordinate Space of the Termination Criteria

The original Bézier clipping method terminates depending on the termination criteria ε in the $2D$ (u,v) parameter space of a projected patch. It is the main bottleneck of the method, because depending on the speed of the parameterization of different segments of the patch, we can achieve different precision of the obtained result in the $3D$ (x,y,z) object space of the patch. In other words, we can overestimate or underestimate the obtained result.

When looking for an intersection point, we are mostly interested in the precision of the result in the $2D$ (x,y) coordinate space of a projected patch. In other words, we are looking for an approximate intersection point, whose deviation from the real intersection point is within the given tolerance ε in the $2D$ (x,y) coordinate space. The original Bézier clipping method has to be improved in order to work properly with the termination criteria ε in the $2D$ (x,y) coordinate space.

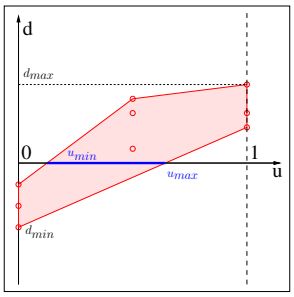


Figure 6: Determining the maximum distance span.

In order to improve the original Bézier clipping method, we have to make the following modifications:

- on each iteration, when determining the signed distances $d_{i,j}$ of the projected control points $\bar{r}_{i,j}$ to the vectors $\bar{L}_u$ or $\bar{L}_v$ (see Section 3, Step 4), we determine the minimum and maximum distances d_{min} and d_{max}. This gives us the *maximum distance span*. Figure 6 shows the d_{min} and d_{max} distances together with the convex hull and the interval of interest for an iteration in u direction.

- we stop to consider the corresponding direction if the size of the maximum distance span in this direction is less than the value ε:

$$(d_{max} - d_{min}) < \varepsilon \tag{19}$$

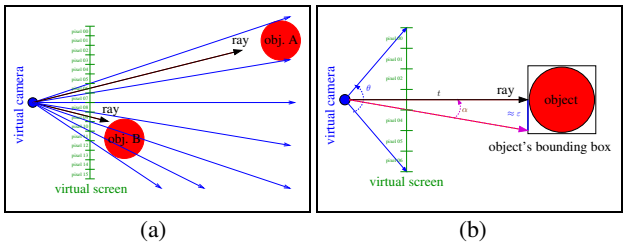
  In this case we do not perform the Bézier clipping and consider the second direction with the same patch segment (or report the intersection if the iteration process in the second direction is already stopped). Notice that we do not construct the convex hull and do not calculate the interval of interest in this case – an intersection is supposed to exist in the middle of the corresponding parameter domain direction of the current patch segment.

- if the size of the maximum distance span is greater than threshold value ε, we construct the convex hull, determine the interval of interest, perform the Bézier clipping and execute the next iteration (see Section 3, Steps 5 to 8).

Notice that problems detected by Campanga et al. [1997] described in Section 4 cannot arise with the modified version of Bézier clipping method, because the method terminates depending on the size of the maximum distance span for both vectors $\bar{L}_u$ and $\bar{L}_v$. However, these vectors are not orthogonal in general and can be of any directions. In some cases they can even coincide. It may cause reporting of wrong intersections because the obtained intersection point can lie far away from the point of origin $(0,0)$. This problem is addressed in Subsection 5.4.

5.3 View Dependent Computation of the Termination Criteria

The modified in Subsection 5.2 Bézier clipping method is more preferable than the original one, because its $2D$ (x,y) coordinate space termination criteria ε is more intuitive and yields more robust results. The idea of an efficient ε computation for primary rays (rays shot from the virtual camera) is explained in this subsection by a simple example in Figure 7. The idea follows the ray differentials [Igehy 1999].



(a) (b)

Figure 7: The determination of value ε for primary rays. (a) The difference in the necessary result precisions. (b) The scheme of value ε computation.

Suppose we are given two objects: *object A* and *object B* of the same shape (represented by rational Bézier patches) as show in Figure 7 (a). Apparently, the *object A* covers just one pixel on the virtual screen, and the *object B* covers three pixels on the virtual screen. It is obviously that an intersection of a ray with the *object A* can be calculated with three times less precision than an intersection with the *object B*. Considering the *object A*, we can take any

point on its surface because all of them are projected on the same pixel of the virtual screen.

This shows that in order to optimize the performance of Bézier clipping method, we have to make the termination criteria ε dependent on the $3D$ (x,y,z) coordinates of an object, the position of the camera, and the resolution of the virtual screen.

A scheme of computing the termination criteria ε is depicted in Figure 7 (b). Suppose t is the distance from the ray origin to the point of intersection with the object's axis aligned bounding box, θ is the camera *field of view* parameter, and x and y are the image width and height in pixels respectively. We express the smallest possible angle α between the ray, which goes through the center of a pixel and the ray, which goes through the pixel's border by equation

$$\sin\alpha = \frac{\sin\theta}{\sqrt{4\cdot(max\{x,y\} - \sin^2(\frac{\theta}{2}))^2 + \sin^2\theta}} \tag{20}$$

Then the value ε can be computed by equation

$$\varepsilon = t\cdot(k\cdot\sin\alpha) \tag{21}$$

The coefficient $k \in R$ is used to control the exact accuracy and can be set to a value from the interval $(0,1]$. When antialiasing techniques are not involved, we can set k to 1. If regular supersampling is used to prevent aliasing, we can use the following equation:

$$k = \sqrt{\frac{1}{n}} \tag{22}$$

where $n \in I$ is the number of rays, which are shot through one pixel. Notice that the coefficient $k\cdot\sin\alpha$ can be precomputed on the preprocessing stage.

This approach works well if we consider only rational Bézier patches with all weights equal to 1. Termination criteria ε of Bézier clipping method must correspond to the $2D$ (x,y) coordinate space of a projected patch, whose control points coordinates are weighted by weight values of the corresponding rational Bézier patch in the $3D$ (x,y,z) object space (see Equation (8)). If we do not take these values into account, we can oversestimate or underestimate the precision. To avoid this problem we must multiply the value ε calculated by Equation (21) by the factor of the minimal weight value of the control mesh of the rational Bézier patch:

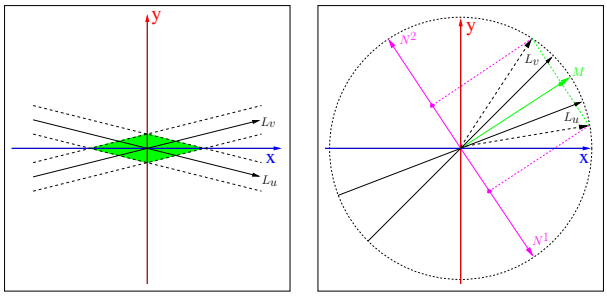
$$\varepsilon = t\cdot min\{w^p_{i,j}\}\cdot(k\cdot\sin\alpha) \tag{23}$$

The proposed approach makes the value ε more intuitive and optimizes the termination criteria of Bézier clipping method. The problem of multiple equivalent intersections, discussed in Subsection 5.1, still exists, but the number of such intersections is significantly reduced and in general is restricted by only few ones.

Notice, however, that the proposed approach works only for primary rays. For secondary rays (reflected, refracted, and shadow rays) we have to use a predefined constant value ε (probably multiplied by the factor of $min\{w^p_{i,j}\}$), which must correspond to the size of the scene bounding box, the precision of computations, etc.

5.4 Efficient Computation of $\bar{L}_u$ and $\bar{L}_v$

Termination criteria in the $2D$ (x,y) coordinate space guarantees that distances from obtained intersection points to both vectors $\bar{L}_u$ and $\bar{L}_v$ (see Subsection 5.2) are less than the threshold value ε. However, the vectors $\bar{L}_u$ and $\bar{L}_v$ are not orthogonal and in general can be of any directions. Figure 8 (a) shows a weak case of the method when the angle between these two vectors is small. Obviously, intersection points can be found everywhere inside the shown parallelogram, i.e., can lie far from the point of origin $(0,0)$.



(a) (b)

Figure 8: Efficient computation of $\bar{L}_u$ and $\bar{L}_v$. (a) Problem of small angle between $\bar{L}_u$ and $\bar{L}_v$. (b) At least 60° between $\bar{L}_u$ and $\bar{L}_v$.

Actually, the problem becomes more serious when the angle between $\bar{L}_u$ and $\bar{L}_v$ decreases. Notice that the efficient threshold value ε computed in Subsection 5.3 corresponds to the case when $\bar{L}_u$ and $\bar{L}_v$ are orthogonal and coincide with $\bar{x}$ and $\bar{y}$ basis vectors of the $2D$ (x,y) coordinate space. In order to satisfy two arbitrary orthogonal vectors, the value ε must be multiplied by the factor of $\frac{1}{\sqrt{2}}$. In order to satisfy two arbitrary vectors of any directions, the value ε must be additionally multiplied by the factor of $\tan(\frac{\varphi}{2})$, where φ is the angle between $\bar{L}_u$ and $\bar{L}_v$. Putting it all together, we have:

$$\varepsilon' = \varepsilon\cdot\frac{\tan(\frac{\varphi}{2})}{\sqrt{2}} = t\cdot min\{w^p_{i,j}\}\cdot\tan(\frac{\varphi}{2})\cdot(k\cdot\frac{\sin\alpha}{\sqrt{2}}) < \varepsilon \tag{24}$$

Calculated in such a way ε' value guarantees that the distance from any obtained intersection point in the $2D$ (x,y) coordinate space to the point of origin $(0,0)$ is at most ε, i.e., any obtained intersection point in the $3D$ (x,y,z) coordinate space projects to the same pixel on the virtual screen.

If the angle φ between $\bar{L}_u$ and $\bar{L}_v$ is small, we can experience the convergence problem, because the value of ε' is also small. To prevent this problem, we have to guarantee, that the angle φ is above some predefined value. During experiments we have found out that $\varphi \geq 60^\circ$ is a good choice. As $\tan(\frac{60^\circ}{2}) = \frac{1}{\sqrt{3}}$, we can calculate the final value of the termination criteria of the modified Bézier clipping method:

$$\varepsilon'' = t\cdot min\{w^p_{i,j}\}\cdot(k\cdot\frac{\sin\alpha}{\sqrt{6}}) \leq \varepsilon' < \varepsilon \tag{25}$$

Notice that the coefficient $k\cdot\frac{\sin\alpha}{\sqrt{6}}$ can be precomputed on the preprocessing stage. The only one problem remains is how to guarantee at least 60° angle between $\bar{L}_u$ and $\bar{L}_v$. This can be done in several steps, which are explained below (see Figure 8 (b)).

Step 1. After determining the vectors $\bar{L}_u$ and $\bar{L}_v$ (see Section 3, Step 3), we compute $\cos\varphi = \bar{L}_u\cdot\bar{L}_v$. If $\cos\varphi < 0$, we invert the direction of $\bar{L}_u$ vector. It guarantees that the angle φ between $\bar{L}_u$ and $\bar{L}_v$ is less than 90°.

Step 2. If $|\cos\varphi| > \frac{1}{2}$, i.e., $\varphi < 60^\circ$, we have to go through the next steps. Otherwise we stop here.

Step 3. We compute the bisector vector $\bar{M}$ by the equation $\bar{M} = \frac{1}{2}\cdot(\bar{L}_u + \bar{L}_v)$. We also compute two vectors, which are orthogonal

to $\bar{M}$ as follows

$$\bar{N}^1 = (M_y, -M_x)$$
$$\bar{N}^2 = (-M_y, M_x) \qquad (26)$$

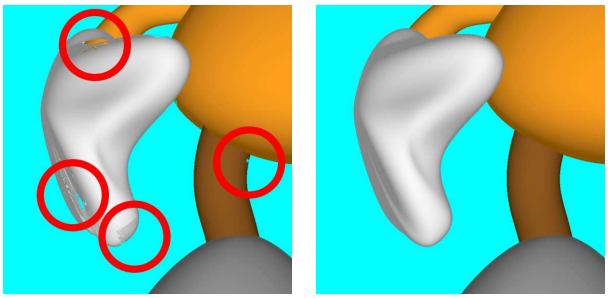
Step 4. If the vector $\bar{L}_u$ makes a right turn with respect to the vector $\bar{M}$, we compute new $\bar{L}_u$ and $\bar{L}_v$ vectors as follows

$$\bar{L}_u = \sin 30^\circ \cdot \bar{N}^1 + \cos 30^\circ \cdot \bar{M} = \frac{1}{2} \cdot \bar{N}^1 + \frac{\sqrt{3}}{2}\bar{M}$$
$$\bar{L}_v = \sin 30^\circ \cdot \bar{N}^2 + \cos 30^\circ \cdot \bar{M} = \frac{1}{2} \cdot \bar{N}^2 + \frac{\sqrt{3}}{2}\bar{M} \qquad (27)$$

Otherwise we swap vectors $\bar{N}^1$ and $\bar{N}^2$ in Equation (27).

Notice that sometimes the initial computation of the vectors $\bar{L}_u$ and $\bar{L}_v$ (see Section 3, Step 3) can give us vectors of zero length. It can happen if some control points of a projected patch are coincide. To avoid the problem of normalization of such vectors, we initially set $\bar{L}_u = (1,0)$ and $\bar{L}_v = (0,1)$ and update them only if we compute vectors of nonzero length. After that we go to the Step 1 of this subsection and correct the direction of $\bar{L}_u$ and $\bar{L}_v$ if the angle between them is less that 60°.

Computing $\bar{L}_u$ and $\bar{L}_v$ only once before the Bézier clipping iterations, as it is suggested in the original method (see Section 3, Step 3), can lead to visible artifacts caused by bad convergence. Moreover, the bad convergence decreases the performance of Bézier clipping method. An example of this problem is shown in Figure 9 (a).



(a) (b)

Figure 9: The problem of convergence of the original Bézier clipping method demonstrated on Gnom.wrl model (*courtesy of Blaxxun Interactive*). (a) The vectors $\bar{L}_u$ and $\bar{L}_v$ are precomputed only once. (b) The vectors $\bar{L}_u$ and $\bar{L}_v$ are recomputed on each iteration.

The problem arises because, after few Bézier clipping steps, the vectors $\bar{L}_u$ and $\bar{L}_v$ do not correspond to the u and v directions of parameterization of obtained patch segments any more. To avoid this problem we have to recompute the vectors $\bar{L}_u$ and $\bar{L}_v$ on each iteration, i.e., iterate on Steps 3 to 8 (see Section 3). Figure 9 (b) shows the result without visible artifacts obtained after the modification of the method. Notice, however, that we recompute the vectors $\bar{L}_u$ and $\bar{L}_v$ only until one of the parameter directions reports an intersection within the tolerance ε''. Afterwards, we consider the remaining direction without recomputing the vectors to guarantee the correct result.

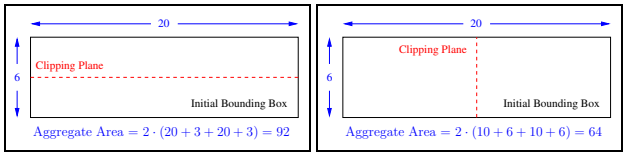
5.5 Acceleration Data Structure for the Initial Clip

This section shows how an acceleration data structure, constructed for a rational Bézier patch on the preprocessing stage, improves the performance of Bézier clipping method. We construct the acceleration data structure as follows.

Step 1. We subdivide the initial rational Bézier patch in the middle of the u parameter direction and construct temporary axis aligned bounding boxes for each half of the patch utilizing the convex hull property.

Step 2. We subdivide the initial rational Bézier patch in the middle of the v parameter direction and construct temporary axis aligned bounding boxes for each half of the patch utilizing the convex hull property.

Step 3. We select that pair of the subpatches (either from Step 1 or from Step 2), which gives us two bounding boxes of smaller aggregate surface area. Such an approach guarantees that the leaves of the acceleration data structure enclose surface segments of almost of the same size in both parameter directions. This idea is shown on a simple $2D$ example in Figure 10.



(a) (b)

Figure 10: The idea of the acceleration data structure construction. (a) Bad choice of a clipping plane. (b) Good choice of a clipping plane.

Step 4. We repeat Steps 1 to 3 recursively for each half of the patch until we reach an appropriate level of recursion or the surface area of the bounding box of a patch segment is below a predefined threshold value. In this case we construct a leaf of the acceleration data structure, which stores the axis aligned bounding box and the parameter domain of the enclosed patch segment. Notice that we do not store the control points and the weights themselves. By this way a binary tree is constructed.

Step 5. When the binary tree construction is finished, we build a $3D$ kd-tree for the leaves of the binary tree. The bounding box of the rational Bézier patch (the initial box for the $3D$ kd-tree) is computed as the union of bounding boxes of all leaves of the binary tree. Notice that such approach yields tight bounding boxes for rational Bézier patches.

When a ray is tested against a rational Bézier patch, ray-axis aligned bounding box test is executed first. If the ray intersects the bounding box, we compute the t distance (see Subsection 5.3), calculate the termination criteria ε'' (see Subsection 5.4), and shoot the ray through the $3D$ kd-tree. By this way we determine all leaves, which are intersected by the ray. As each leaf stores the parameter domain of the enclosed patch segment, we can apply the *initial clip* - cut away those parts of the rational Bézier patch, which are guaranteed not to have intersections with the ray. Notice that we apply the initial clip in the $2D$ (x,y) coordinate space of the projected patch rather than in the $3D$ (x,y,z) object space. After applying the initial clip, we execute Bézier clipping method with the clipped patch segment.

6 Results

The proposed modifications of Bézier clipping method have been implemented within a library for NURBS surfaces evaluation. The library has been integrated into a working ray tracing system. The

goal of this paper was not to implement optimized framework of the modified Bézier clipping method. The implementation of the method is rather general than optimized – it supports raytracing NURBS surfaces of any degree and any knot vectors. Moreover, the library is supposed to be easily integrated in any ray tracing system, and therefore is not optimized for a particular ray tracing application framework. Many different scene models, which consist of NURBS surfaces, have been rendered. The rendered images do not contain visible artifacts. Some rendered images of a complete Mercedes C-class model in trimmed NURBS representation are shown in Figure 11.

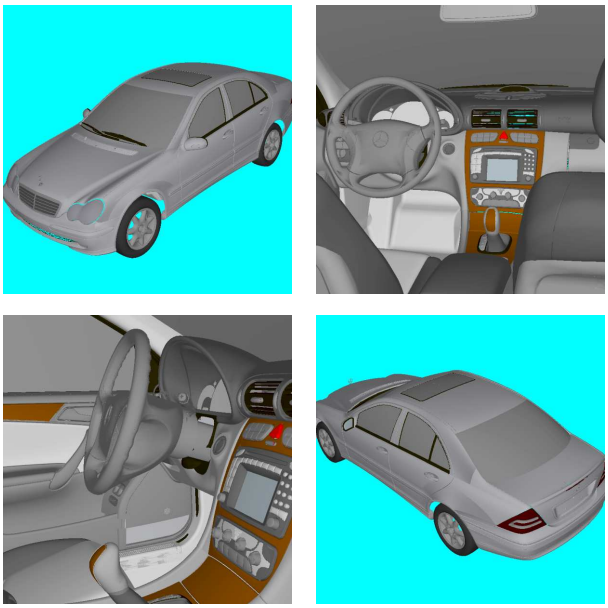


Figure 11: Rendered images of a complete Mercedes C-class model (*courtesy of DaimlerChrysler AG*).

7 Conclusion and Future Work

In this paper we have presented improvements of Bézier clipping method for raytracing NURBS surfaces, originally proposed by Nishita et al. [1990]. The multiple equivalent intersections problem has been detected and demonstrated on a simple example. The reasons for changing the coordinate space of the termination criteria have been explained and an efficient view dependent scheme for computing the termination criteria for primary rays have been proposed. It has been shown how to avoid the convergence problem, caused by small angles between the $\bar{L}_u$ and $\bar{L}_v$ vectors. The advantage of utilizing an acceleration data structure for the initial clip has been demonstrated.

The proposed improvements deal only with the core of Bézier clipping method. Different extensions of the original Bézier clipping method, such as utilizing the ray coherence [Wang et al. 2000], work with the modified method as well.

There are several possibilities for further improving the implementation. SIMD processor instructions can be utilized [Benthin et al. 2004; Geimer and Abert 2005] in order to improve the performance. As the performance of the method depends on the degree of rational Bézier patches, a special kind of the degree reduction technique, which preserves the surface geometry within a given tolerance, can be used on the preprocessing stage. Though, this paper does not deal with trimmed NURBS surfaces, a trimming test framework has also been implemented and can be further optimized.

8 Acknowledgement

We would like to thank Charles Adams, Blaxxun Interactive and DaimlerChrysler AG for providing models in NURBS representation. This work was partially supported by the European Union within the scope of project IST-2001-34744, “Realtime Visualization of Complex Reflectance Behaviour in Virtual Prototyping” (RealReflect).

References

BARTH, W., AND STÜRZLINGER, W. 1993. Efficient ray tracing for Bezier and B-spline surfaces. *Computers & Graphics 17*, 4, 423–430.

BENTHIN, C., WALD, I., AND SLUSALLEK, P. 2004. Interactive Ray Tracing of Free-Form Surfaces. In *Proceedings of Afrigraph 2004*.

CAMPAGNA, S., SLUSALLEK, P., AND SEIDEL, H.-P. 1997. 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.

GEIMER, M., AND ABERT, O. 2005. Interactive Ray Tracing of Trimmed Bicubic Bézier Surfaces without Triangulation. In *Proceedings of WSCG*.

GUTHE, M., MESETH, J., AND KLEIN, R. 2002. Fast and Memory Efficient View-Dependent Trimmed NURBS Rendering. In *Pacific Conference on Computer Graphics and Applications*, 204–213.

IGEHY, H. 1999. Tracing ray differentials. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 179–186.

LISCHINSKI, D., AND GONCZAROWSKI, J. 1990. Improved techniques for ray tracing parametric surfaces. *The Visual Computer 6*, 3, 134–152.

MARTIN, W., COHEN, E., FISH, R., AND SHIRLEY, P. 2000. Practical ray tracing of trimmed NURBS surfaces. *J. Graph. Tools 5*, 1, 27–52.

NISHITA, T., SEDERBERG, T. W., AND KAKIMOTO, M. 1990. Ray tracing trimmed rational surface patches. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, ACM Press, 337–345.

PIEGL, L., AND TILLER, W. 1995. *The NURBS book*. Springer-Verlag.

QIN, K., GONG, M., YOUJIANG, G., AND WANG, W. 1997. A new method for speeding up ray tracing NURBS surfaces. *Computers & Graphics 21*, 5, 577–586.

TOTH, D. L. 1985. On ray tracing parametric surfaces. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, 171–179.

WANG, S.-W., SHIH, Z.-C., AND CHANG, R.-C. 2000. An improved rendering technique for ray tracing Bézier and B-spline surfaces. *Journal of Visualization and Computer Animation 11*, 4, 209–219.