Direct Trimming of NURBS Surfaces on the GPU

Andre Schollmeyer* Bernd Fröhlich†
Bauhaus-Universität Weimar

Abstract

This paper presents a highly efficient direct trimming technique for NURBS surfaces, which is applicable to tessellation-based rendering as well as ray tracing systems. The central idea is to split the trim curves into monotonic segments with respect to the two parameter dimensions of the surface patches. We use an optimized bisection method to classify a point with respect to each monotonic trim curve segment without performing an actual intersection test. Our hierarchical acceleration structure allows the use of a large number of such curve segments and performs the bisection method only for points contained in the bounding boxes of the curve segments.

We have integrated our novel point classification scheme into a GPU-based NURBS ray casting system and implemented the entire trimmed NURBS rendering algorithm in a single OpenGL GLSL shader. The shader can handle surfaces and trim curves of arbitrary degrees, which allows the use of original CAD data without incorporating any approximations. Performance data confirms that our trimming approach can deal with hundreds of thousands of trim curves at interactive rates. Our point classification scheme can be applied to other application domains dealing with complex curved regions including flood fills, font rendering and vector graphics mapped on arbitrary surfaces.

CR Categories: I.3.5 [Computer graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations I.3.7 [Computer graphics]: Three-Dimensional Graphics and Realism—Raytracing

Keywords: parametric surfaces, trimmed NURBS, point classification, ray casting, root finding, programmable graphics hardware

1 Introduction

Computer Aided Design (CAD) modeling tools are widely used for the industrial design of models for prototyping and production. The standard surface representation within these systems are trimmed non-uniform rational B-Spline (NURBS) surfaces. Trimmed NURBS surfaces are capable of compactly describing almost any shape and additionally provide local and intuitive control. These patches are commonly visualized by rendering a triangle-based approximation (tessellation) on graphic processing units (GPU), since there is no specialized NURBS rendering hardware readily available. The process of generating a high-quality tessellation is time consuming, especially for trimmed surfaces requiring fine sampling of the trim boundaries. Recent on-the-fly tessellation techniques propose trimming approaches for the CPU [Balázs et al. 2004] as well as for the GPU [Guthe et al. 2005]. However, since they either use mesh-based or texture-based techniques, updates of these representations are required to maintain pixel-accuracy during interactive viewpoint manipulations.

Figure 1: *Per-pixel accuracy for arbitrary zoom levels.*

In this paper we present a highly efficient direct trimming technique for NURBS surfaces, which is applicable to tessellation-based rendering as well as ray tracing systems. Trimming is accomplished by employing the ray-based point-in-curve test with respect to the NURBS trim curves. Our central idea is to split the trim curves into monotonic segments such that there is at most a single intersection with a horizontal ray. The split points correspond to the extrema of the trim curves and can easily be found. Within each single-intersection curve segment we use an optimized bisection method to classify a point as being inside or outside of the trim region without performing an actual ray-curve intersection. The set of monotonic trim curve segments for each patch suggest a two-level hierarchical acceleration structure, which organizes the entire trim region into cells containing, in most cases, one trim curve segment or none at all. During NURBS rendering the acceleration structure is searched for the cell containing the point to be classified and only the contained trim curve segments are further considered with our optimized bisection method.

Common trimming implementations use Bézier Clipping as a robust technique for finding all the intersections of a ray and a trim curve. However, Bézier Clipping is originally a recursive technique, which is difficult to efficiently implement on a GPU. An iterative GPU implementation of Bézier Clipping ([Pabst et al. 2006]) was limited to a small number of curves due to GPU programming constraints and the lack of an acceleration structure. Furthermore, Bézier Clipping requires quite a number of computationally non-trivial steps such as convex hull computations and Bézier polygon subdivisions. A robust implementation also needs to consider numerous special cases ([Wang et al. 2000; Efremov et al. 2005]).

The main contribution of this paper is a new point classification scheme for curved regions with holes. We apply this scheme for efficiently handling a large number of trim curves in real-time rendering systems. Our approach does not use any approximations of the trim curves, nor is the degree of the curves limited. Due to our trim-curve preprocessing we are able to employ an optimized bisection method for classifying a point with respect to a bi-monotonic curve segment. Problematic ray-curve intersection cases such as multiple zeros, minima and maxima are resolved during a preprocessing step, all of which contribute to the robustness of our method. A complexity estimation conveys that our classification of a point contained in the bounding box of a trim curve segment requires on aver-

*e-mail:andre.schollmeyer@uni-weimar.de
†e-mail:bernd.froehlich@uni-weimar.de

age less than two evaluations of the curve. This estimation was also empirically confirmed. A comparison reveals that the commonly employed Bézier Clipping technique is about five times slower than our technique for the investigated real world examples. However, due to the precision of our approach, inaccuracies of the actual trim curve representations may be revealed, which results in pin holes or cracks along the trim boundary of two adjacent surface patches. For evaluation purposes, we integrated our trimming method into a GPU-based ray casting system. The entire rendering algorithm for trimmed NURBS surfaces is implemented as a single fragment program which can handle surfaces and trimming curves of arbitrary degree. Due to our two-level acceleration structure we achieve interactive frame rates for models containing more than a hundred thousand trim curves. The efficiency of our approach makes it applicable to a variety of other domains requiring point classification with respect to complex curved regions including font rendering, flood filling, and vector graphics mapped on arbitrary surfaces.

2 Related Work

We provide a short overview of existing rendering techniques for trimmed NURBS surfaces with a focus on the applied trimming method. The surface rendering techniques can be classified as methods based on tessellation or on ray tracing.

Tessellation-based approaches convert the NURBS surface representation into a set of triangles. [Rockwood et al. 1989] and [Kumar and Manocha 1995] use uniform subdivision methods to generate piece-wise linear approximations of the trimming curves, and use them as polygonal boundaries for the surface tessellation. [Balázs et al. 2004] propose an elevation of the trimming curves into Euclidian space to constrain the error of a piece-wise linear approximation of the trim curves.

In contrast to these preprocessing-based meshing methods, [Guthe et al. 2005] adaptively triangulate the NURBS surface on the GPU and perform the actual trimming task on a per-pixel basis using a lookup-texture. They generate a texture representation of the trimming area by rendering a polygonal representation of each trim loop. The resolution of the trim texture representations is adapted to the resolution of the trimmed surface on the screen and updated on the fly if necessary. All of these approaches use a polygonal approximation of the trim curves and a tessellation of the NURBS surfaces. Furthermore, for high screen resolutions these techniques need to create an enormous number of triangles and a large number of high resolution trim textures, both of which may need to be continuously updated during viewpoint navigation. While these techniques may produce similar quality images as our approach, we require significantly less memory on the GPU since only the control points of the trimming curves and a small acceleration structure need to be stored and potentially updated during surface editing.

Pioneering work in the field of parametric surface ray tracing [Kajiya 1982; Toth 1985; Joy and Bhetanabhotla 1986] focuses on different ray-surface intersection methods, but does not provide a solution for the trimming problem. [Farouki 1987] was the first to introduce a set of generalized algorithms for processing trimmed surfaces. The seminal paper on "Ray Tracing Trimmed Rational Surface Patches" by [Nishita et al. 1990] presented the Bézier Clipping method as a robust numerical root-finding algorithm, which can be used for ray-surface intersections as well as for ray-curve intersections. It exploits the convex hull property of Bézier representations and recursively subdivides curve segments containing potential roots. [Wang et al. 2000] and [Efremov et al. 2005] proposed several enhancements to further improve the stability and applicability of the Bézier Clipping algorithm.

Several NURBS ray tracing systems, including [Geimer and Abert 2005; Benthin et al. 2004; Efremov et al. 2005], make use of Bézier Clipping for trimming. [Efremov et al. 2005] utilizes an adapted version of the ray-casting-based point-in-polygon test for trim loops. They apply Bézier Clipping for intersecting a ray with the trim curves. Additionally, a trim loop hierarchy accelerates the trimming test by avoiding unnecessary intersections. [Benthin et al. 2004; Geimer and Abert 2005] also use the point-in-polygon test and Bézier Clipping. They almost reach interactive frame rates due to the clever use of SIMD extensions and Kd-tree-based acceleration structures. However, they only deal with cubic trimming curves.

[Pabst et al. 2006] presented a GPU-based NURBS ray casting system. They provided an iterative formulation of the originally recursive Bézier Clipping algorithm to directly perform the trimming task on the GPU during fragment processing. However, the system was limited to a small number of trimming curves of low degree due to several constraints of current graphics hardware and the lack of an acceleration structure.

Splitting curves into monotonic segments is used in many works. The idea of preprocessing curves into bi-monotonic segments was introduced by [Mudur and Koparkar 1984] for use in an algorithm for intersecting two planar parametric curves. [Rockwood et al. 1989] split trim curves into monotonic segments to robustly generate the tessellation of trimmed NURBS patches. For this process, they also developed their own root finder, which has some similarities to Bézier Clipping. [Keyser et al. 1999] base their MAPC library for manipulating algebraic curves (e. g. intersecting two curves) on monotonic segments. They call the process of splitting a curve into monotonic segments "resolving the curve topology". [Qin et al. 2008] split curves into monotonic segments to simplify distance computations.

3 Trimming Approach

Trimming is a common method to overcome the topological limitation of a rectangular parameter domain. Trimming curves are often generated by the intersection of two surface patches in 3D space. These intersection curves are algebraic space curves of very high degree (e. g. the intersection curve of two bicubic patches is of degree 324). NURBS curves of limited degree (e. g. up to degree 15) are often used as an approximation to these high degree space curves, which, in general, does not create watertight surfaces along the trim curves in 3D space. This is because approximating the intersection curve creates two slightly different trim curves on each of the adjacent surfaces. This problem is well known (e. g. [Song et al. 2004]), but is typically either neglected or camouflaged by drawing a fat line along the actual trim curve. While our point classification approach precisely evaluates NURBS-based trim curves, it does not address these approximation inaccuracies, which may show as pin holes or cracks along trim curves.

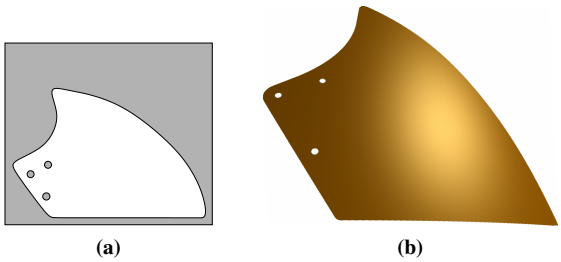


Figure 2: *2a) The domain of a surface including one outer and three nested trimming regions and 2b) the resulting surface*

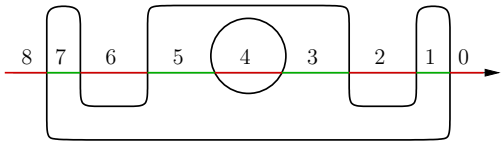


Figure 3: *Even-odd-rule for ray-based point-in-closed-curve test. Points on the ray shown in red have an even number of intersections with the trimming curve and lie outside of the trimming region. Green points are classified as inside.*

The actual domain of the parametric surface is defined by sets of non-intersecting, closed NURBS curves. Nested trimming regions allow the definition of holes and islands as shown in Figure 2.

Trimming approaches based on rational polynomial representations of trimming curves commonly use a ray-based point-in-closed-curve method ("even-odd-rule") to determine if a particular point should remain or has to be trimmed (Figure 3). This method has to compute and count all ray intersections with the trimming curves, which is basically a root-finding problem. Since trimming curves can be of high degree, they are sometimes approximated by cubic segments for this test. In any case, a general root finding method capable of enumerating all the intersections of a ray and a NURBS curve must be applied. Bézier Clipping has been the method of choice in all previous work. However, Bézier Clipping is computationally expensive and requires treating numerous special cases in order to avoid numerical problems.

Our idea suggests avoiding the general root-finding problem by splitting the trimming curves into monotonic segments, which are guaranteed to have at most one intersection with a ray parallel to one of the parameter domain axes. This preprocessing step allows us to use simple root-finding methods at run time, which are ideally suited to be implemented on the GPU. In addition, the subdivision also leads to an acceleration structure, which very efficiently deals with a large number of such trimming curve segments.

3.1 Preprocessing of Trimming Curves

In most systems and CAD data exchange formats, trimming curves are represented as NURBS curves, which map from their one-dimensional parameter space t into the two-dimensional (u, v)-domain of the parametric surface. We transform the NURBS representation into its equivalent rational Bézier representation to reduce the run time computation for evaluating a point on the curve. The resulting rational Bézier curves are defined by

$$\mathbf{C}(t) = \frac{\sum_{i=0}^{n} w_i \mathbf{b}_i B_i^n(t)}{\sum_{i=0}^{n} w_i B_i^n(t)} \tag{1}$$

with two-dimensional control points $\mathbf{b}_i$, scalar weights w_i and the n-th degree Bernstein polynomials

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \qquad t \in [0,1] \tag{2}$$

The classification of a point $\mathbf{p}$ lying in the (u, v)-parameter domain of the parametric surface is accomplished by generating a ray originating at $\mathbf{p}$ and intersecting it with the surrounding trim curves. In general, the intersection of such a ray and a Bézier curve of degree n may result in up to n intersections. This intersection problem can be transformed into a root-finding problem of the same degree. The transformation is greatly simplified by considering only rays pointing in positive u-direction ($v = v_p$) as shown in Figure 4a.

This ray $\mathbf{r}$ is defined by

$$\mathbf{r}(\tau) = \begin{pmatrix} u_p \\ v_p \end{pmatrix} + \tau \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{3}$$

For finding the ray-curve intersections with rays parallel to the u-axis, only the roots of the non-parametric explicit representation of the v-component of the trimming curve shifted by $-v_p$ have to be found:

$$C_v(t) - v_p = 0 \tag{4}$$

Instead of finding all these roots for each potential point on the surface patch, we are splitting the non-parametric trim curve $C_v(t)$ into segments such that each segment contains at most one real root. To obtain these segments, we apply Rolle's theorem. When applied to our context, the theorem says that for continuous, differentiable functions, real roots are separated by an extremum. Thus, the idea is to find all the extrema and make use of the fact that real roots only occur between subsequent extrema of different signs or at the extreme points. This is only valid for continuous functions. Rational Bézier curves are guaranteed to be continuous within their domain if their weights remain positive. Figure 4b illustrates that the split process does not depend on the actual ray coordinates ($v = v_p$) since they do not affect the location of the extrema. Once the curve is split into these segments, simple root finders such as the robust bisection method can be used.

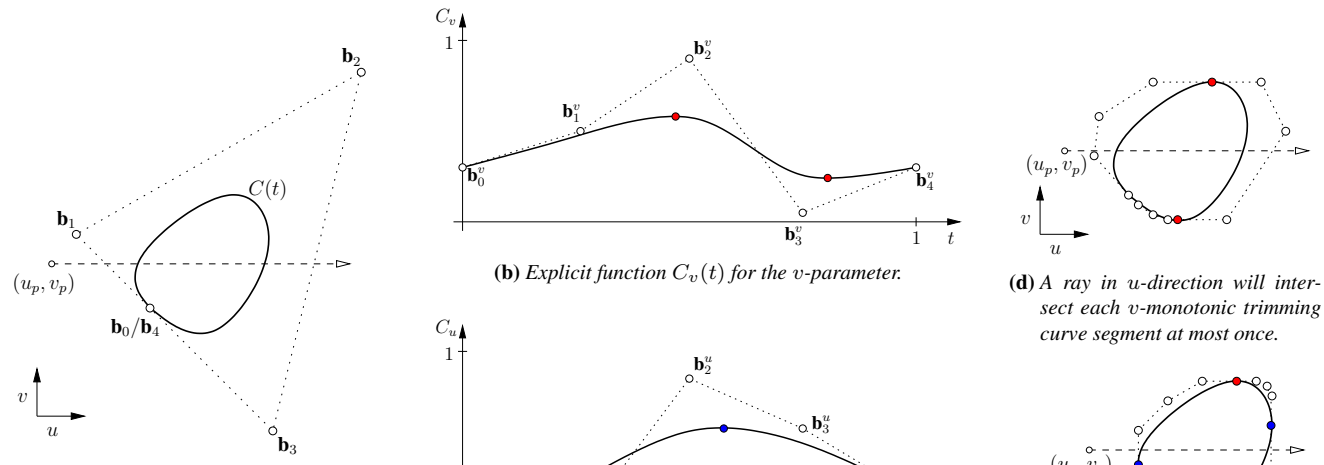
The extrema of the curve $C_v(t)$ can be found by differentiation and solving $C_v'(t) = 0$. For non-rational trimming curves the derivative can be directly determined [Farin 1993]. For rational curves the non-rational enumerator has to be used as described in [Sederberg and Wang 1987]. In both cases, the resulting derivative is an integral explicit polynomial function in Bézier form for which all roots are to be obtained. While turning points also fulfill $C_v'(t) = 0$, they are not required for finding the roots of the original curve. However, we also split at the turning points to avoid dealing with an additional case.

For solving $C_v'(t) = 0$ any conventional root-finding algorithm can be applied. A particular constraint of our problem is that we are only interested in finding the roots in the interval $[0..1]$. Bézier Clipping would work well since it directly considers the interval boundaries. However, as stated before, a robust implementation of this algorithm is quite involved. We use a much simpler alternative based on Rolle's theorem and recursive differentiation similar to the approach proposed by [Collins and Loos 1976]. The implementation only consists of a few lines of code and worked well for all our real-world data.

Once the extrema of the curve $C_v(t)$ are determined it is subdivided at all locations t satisfying $C_v'(t) = 0$. The resulting monotonic curve segments are shown in Figure 4d. For splitting a curve at the extreme points we employ the de Casteljau algorithm.

3.2 Trimming Curve Intersection

Once the curve $C_v(t)$ is split into monotonic segments containing only single roots, simple root finders can be used to find the roots of $C_v(t) - v_p = 0$. We will use the bisection method which converges to a unique root within an interval $[a, b]$, if the function is continuous in $[a, b]$ and the functional values $f(a)$ and $f(b)$ differ in sign. Even though the bisection method only converges linearly, the algorithm has some major advantages. In contrast to Newton's method it is numerically stable and one iteration step only requires a single curve evaluation. As the curve's derivative is not required, the evaluation can be accomplished using a method similar to the one



(a) *A trivial example of a trimming region consisting of a single trimming curve.*

(b) *Explicit function $C_v(t)$ for the v-parameter.*

(c) *Explicit function $C_u(t)$ for the u-parameter.*

(d) *A ray in u-direction will intersect each v-monotonic trimming curve segment at most once.*

(e) *For bi-monotonic trimming curve segments, the optimized bisection can be applied.*

Figure 4: *This example illustrates the processing for a single closed trim curve (4a). After the preprocessing steps, we obtain the segments shown in (4e). The original curve is subdivided at its extrema with respect to v (red colored) and u (blue colored).*

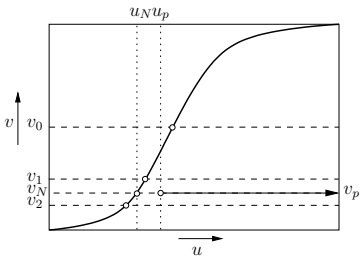


Figure 5: *The naive bisection algorithm.*

presented by [Pavlidis 1982], also referred to as the Horner scheme in Bernstein basis. The algorithm exploits the recursive derivation of the binomial coefficients and evaluates a Bézier curve of degree n in $O(n)$ steps by using nested multiplications. The small and constant number of registers required by Horner's scheme enables the evaluation of arbitrary degree curves on GPUs.

As shown in Figure 5, a naive version of our root finder would start solving $C_v(t) = v_p$ by subsequently bisecting a given interval until the resulting u-interval containing the intersection point is smaller than a given epsilon. The achieved accuracy of the intersection point with respect to the curve parameter t is $\Delta t_N \leq \frac{1}{2^N}$ assuming that the algorithm started with the interval $[0..1]$ and N iterations were required. We still have to classify the intersection regarding the ray direction. The ray intersects the curve in a positive direction only if $u_p < u_N$.

The ray-based point-in-closed-curve test only requires the number of intersections and it does not necessarily require the actual coordinates of the intersection points. Hence, the iteration should stop once the origin of the ray has been classified as being left or right of the curve. This can be effortlessly added into the bisection algorithm if each bisection step can easily calculate the curve's u-interval for each subsequent v-interval to be considered. If the ray origin is not contained in the u-interval, no further bisection is necessary. For ray origins smaller than the u-interval, there is an intersection. Otherwise no intersection exists.

Our preprocess splits the curve $C_v(t)$ at the extreme points to generate monotonic curve segments. These splits also imply v-monotonic curve segments for $\mathbf{C}(t)$, which contain at most a single intersection with a horizontal ray. By additionally splitting the curve at the extreme points of $C_u(t)$, we generate monotonic curve segments with respect to both directions, v and u. As each curve segment is now additionally monotonic in u-direction, each split can directly compute the corresponding u-interval for each v-interval during the bisection process (see Figure 6a) as described by [Mudur and Koparkar 1984]. Thus the algorithm can terminate early for most rays after a small number of iterations N as shown

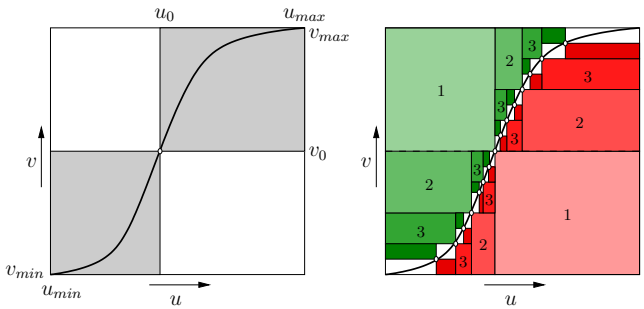
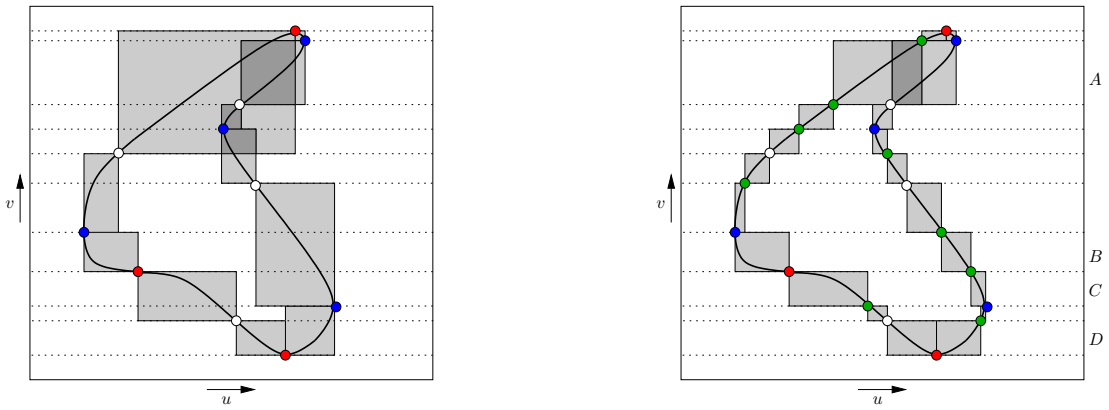


Figure 6: *6a) Each point on a bi-monotonic curve defines a corner of the subsegment's bounding boxes. 6b) Optimized bisection for bi-monotonic curves. Regions with definite intersections are green and also show the number of iterations after which the decision was taken. Non-intersecting regions are marked red.*



(a) (b)

Figure 7: *7a) The end points of the bi-monotonic curve segments create a set of intervals along the v-axis. The endpoints of the original Bézier curves are shown in white. The curve subdivisions at the curve's u- and v-extrema are marked in blue and red respectively. 7b) All curve segments are clipped to the v-intervals created by the bi-monotonic curve segments. These additional curve subdivisions are shown as green points.*

in Figure 6b. In fact, the average number of curve evaluations for evenly distributed test points within the bounding box of the curve segment can be easily estimated for a particular case. Assuming that the v-parameter corresponds to the t-parameter of the curve (Figure 6b), then the first bisection step classifies half of the area of the bounding box as being inside or outside with a single curve evaluation. The next bisection step classifies one quarter of the area and requires then a total of two evaluations, etc. Thus the average number of curve evaluations is given by the following sum with N being the number of iterations:

$$\sum_{i=1}^{N} \frac{i}{2^i} < 2 \tag{5}$$

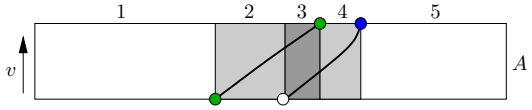
4 Trimming Curve Acceleration Structure

We have shown that trimming curves can be preprocessed in a way such that the point-in-closed-curve test can be efficiently computed for a single bi-monotonic curve segment. It requires less than two curve evaluations on average for points contained in the bounding box of a curve segment. However, real world models contain up to hundreds of trimming curves per patch. It is therefore inevitable to build an acceleration data structure to limit the intersection candidates for a given ray.

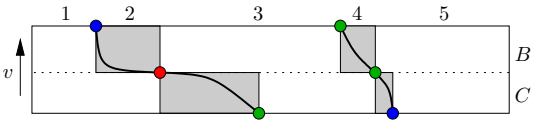
The bi-monotonic curve segments generated during the preprocessing stage provide a good starting point for the acceleration structure. They split the v-axis into a set of intervals as shown in Figure 7a. Obviously, rays in u-direction only have to test curve segments with an appropriate v-interval. If such a candidate curve segment is found, there is a definite intersection of the line corresponding to the ray and the trimming curve segment contained in the curve segment's bounding box. This is due to the fact that all preprocessed curve segments are v-monotonically increasing. As a result, curve segments only have to be considered in more detail if the ray's origin is contained inside a curve segment's bounding box. Otherwise, depending on the actual location of the ray origin, a definite intersection is reported if one exists.

The v-intervals generated by the preprocessed curve segments' end points imply further splits of other curve segments such that we end up with curve segments clipped to the given v-intervals as shown in Figure 7b. This clipping step requires ray-curve segment intersections, which can be handled by the unoptimized bisectioning algorithm presented in the previous chapter.

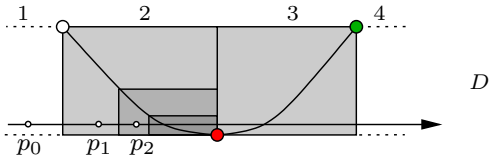
Each v-interval encloses a number of curve segments. The minimal



(a) *If two curves' bounding boxes overlap in u-direction both curves need to be intersected in the inner u-interval (A3).*

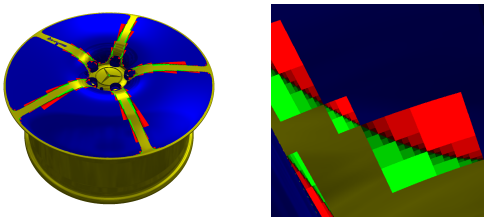


(b) *At turning points curves are subdivided. As the resulting segments are in disjoint v-intervals (B, C) and the binary search will return one v-interval only, incorrect multiple intersections with both curve segments are impossible.*



(c) *The split at the curve's extrema results in two curve segments with their according u-intervals (D2, D3). A horizontal ray with different potential starting points p_0, p_1 and p_2 is shown. p_0 is directly classified in interval D1 as having two intersections. p_1 is classified in interval D2 after one step of the bisection method. p_2 requires two bisection steps. For each of these starting points the curve segment in interval D3 is never investigated, since there is one definite intersection previously found.*

Figure 8: *This figure shows how the acceleration structure deals with typically problematic cases such as extrema and turning points.*



(a) (b)

Figure 9: *This figure shows the number of required curve evaluations for points inside the trim region (green) and points outside the trim region (red) for the rim model. Darker colors indicate a larger number of curve evaluations. Within yellow and blue regions there are no curve evaluations required, due to the acceleration data structure.*

and maximal u-coordinates of the bounding boxes of these curve segments need to be sorted in u-direction, which leads to an ordered list of disjoint u-intervals. We assign the number of potential and definite intersections to each created u-intervals as shown in Figure 8. Thus, the optimized bisectioning algorithm from the previous chapter only needs to be triggered for curve segments, where the ray originates in the curve segment's bounding box as can be seen in Figure 6.

The acceleration data structure for each trimmed surface is stored in a two-level hierarchy. The upper level includes a sorted list of v-intervals and the boundaries of the entire trimming region. The list of u-intervals per v-interval constitutes the second level. Each u-interval entry comprises the number of definite intersections and a list of curve segments which have to be considered by the bisectioning scheme. The traversal of the acceleration structure for a point $\mathbf{p} = (u_p, v_p)$ (respectively a ray with origin $\mathbf{p}$ and pointing in positive u-direction) is realized as follows:

1. If $\mathbf{p}$ lies within the boundaries of the entire trimming region, find the according v-interval using binary search, otherwise $\mathbf{p}$ is discarded.

2. Find the respective u-interval within the current v-interval by binary search. Set the number of total intersections to the number of definite intersections of the found u-interval.

3. Iterate over the u-interval's list of potentially intersecting curve segments. Intersect each curve using the optimized bisectioning method and add the number of found intersections to the total number of intersections.

4. Classify $\mathbf{p}$ using the even-odd-rule based on the total number of intersections.

The acceleration method scales well with an increasing number of curves, since we employ two binary searches to find the respective u-interval. Even further splits of trimming curve segments with large bounding boxes could be considered to reduce the number of curve evaluations. The robustness of our point classification method is a result of the preprocessing. The problematic curve points such as minima, maxima and turning points (independent of their multiplicity) are identified during the preprocess and used to split up the trim curves in single-intersection segments. Since the bounding boxes of two adjacent curve segments never overlap, the point classification for an individual point has to deal at most with either the segment to the left or to the right of an extrema or turning point – never with both segments. Thus incorrect classifications of points on rays in the vicinity of extrema or turning points are avoided.

5 Implementation and Results

While our trimming approach would be also very effective for CPU-based ray tracing systems, we have focused on an evaluation within a tessellation-based NURBS rendering system as well as within a GPU-based NURBS ray casting system. Our trimming algorithm is implemented as a single fragment program using the OpenGL API and GLSL. All benchmarks have been performed on an Intel(R) Core(TM)2 Quad CPU running at 2.66Ghz with 8GB memory, two Nvidia GeForce GTX280 in SLI mode and a resolution of 1024x768. Our models were created using direct IGES export from CATIA V5 without further optimizations.

A preprocessing stage converts all NURBS trimming curves into an equivalent rational Bézier curve representation. The resulting set of curves is then split to construct bi-monotonic curve segments, which are then further clipped to the v-intervals. Table 1 gives an impression of the number of curves involved when processing a large model.

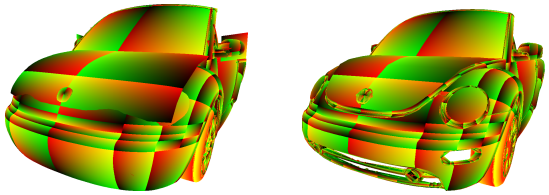
preprocessing stage	number of curves
original NURBS	125,055
after Bézier conversion	241,186
after splitting at the extrema	271,309
after removing redundancies	215,971
acceleration structure	412,512

Table 1: *This table shows the number of trimming curves that are generated during the preprocessing stage for the VW Beetle model (Figure 11c). The actual number of necessary splits at the extrema is quite low. The conversion of NURBS patches into rational Bézier patches often results in completely trimmed Bézier surfaces, which are removed along with their trimming curves. Linear trimming segments in the u-direction are also removed since they cannot be intersected by a ray in the u-direction.*

Our single-pass NURBS ray casting system is based on the approach proposed by [Pabst et al. 2006]. Our implementation differs in various aspects and thus we will briefly describe the complete algorithm. Initially, all NURBS surfaces are converted into an equivalent rational Bézier patch representation. The convex hulls are generated for these patches and passed to the rendering pipeline. The fragment program generates a ray for each resulting fragment and intersects it with the parametric surface representation. The ray-surface intersection is computed using Newton's method. For surface evaluations we replaced the de Casteljau algorithm used in [Pabst et al. 2006] by a Horner-like evaluation method described by [Sederberg 1995]. Sederberg's method can be implemented on the GPU with constant register usage and a complexity of $O(n^2)$ operations, whereas the register usage of the de Casteljau algorithm depends on the degree of the surface and requires $O(n^3)$ operations.

Once a ray-surface intersection is found, the resulting point (u_p, v_p) in the parameter domain is passed to our trimming algorithm and classified with respect to the trimming region of the patch. All fragments classified to be outside the domain of the surface are discarded. We must point out that the intersection as well as the trimming algorithm for all surfaces is realized as a single fragment program. Thus we neither have to limit the degree of the patches or trimming curves, nor do we need to use lower degree approximations. Furthermore, this single-pass implementation can be integrated with any rasterization-based rendering system.

We have also implemented a simple tessellation-based NURBS renderer, which uses texture coordinate interpolation to assign a (u, v)-



(a) (b)

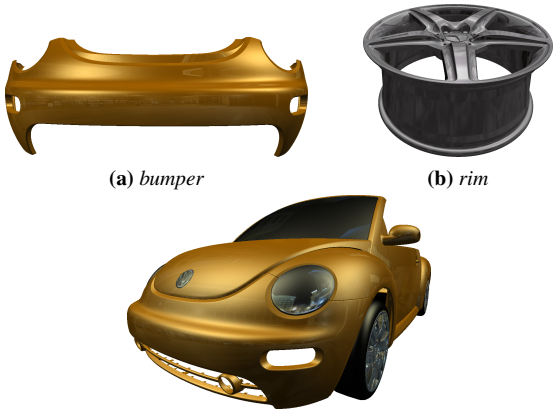
Figure 10: *This figure shows the tessellation of a VW Beetle Cabrio colored using the interpolated parameter values. The model contains more than 200,000 trimming curve segments. Figure 10a shows the untrimmed model. Using our trimming algorithm the frame rate is still above 60Hz (10b).*

coordinate to each fragment (Figure 10). The fragment is then processed by our trimming approach in a single fragment program as described for the NURBS ray casting system.

During all benchmarks our application has been parameterized as follows. If no early classification can be done, the bisection continues until either the remaining u-interval or the parameter interval reaches floating-point precision, obtaining an error bound of $1 \cdot 10^{-7}$. Experiments have shown that the actual error bound has almost no influence on the execution time, since the average number of iterations is limited by two as shown in Equation 5. Newton's method as part of the ray-surface intersection was limited to 5 iterations at most. The convergence threshold is set to $1 \cdot 10^{-3} mm$ in object space.

As a first test we measured the performance of our trimming algorithm by rendering tessellated models with an increasing number of surfaces and trimming curves. The draw times, as shown in Table 2, suggest that our algorithm is basically limited by the number of generated fragments and thus is image resolution dependent. With an increasing number of trimmed surfaces and an almost constant screen coverage, the draw time increases only slightly. The performance for the close-up of the bumper is further improved as the coherent texture reads can exploit texture caching hardware. Tests with alternative root finding algorithms such as the secant method and regula falsi have shown that though they generally converge faster, the average number of necessary iterations and therefore curve evaluations is almost the same. The regula falsi method is even slower than the bisection method, since it needs to treat special cases to preserve numerical robustness. The secant method seems to perform slightly faster than the bisection method, but may suffer from slow convergence in rare cases.

A direct performance comparison of our results to the trim-texture-based approach of [Guthe et al. 2005] is challenging due to the implementation complexity of the system and differences in scenes and parameter choices (e. g. texture size). Guthe et al's method is certainly faster for a constant view point, since they only need a single texture look-up to classify a domain point. However, if the view point changes trim textures need to be recomputed, which may cause frame rate drops and intermediate visual artifacts. Our trim curve representation is view-independent and only needs updates if control points of surfaces or trimming curves change. This feature leads to quite constant frame rates in an interactive context. Both approaches may result in visual artifacts due to the approximative nature of trim curves. In addition, Guthe et al's method employs approximations of the trim curves in early stages of the pipeline, while we use the curved representation down to the pixel level. The quality of Guthe et al's method mainly depends on a given screen space error, which also defines the required texture size for the discretized trim curve representations. Thus, memory usage may highly vary



(a) *bumper* **(b)** *rim*

(c) *VW Beetle*

Figure 11: *This figure shows the different models used for our test scenarios. The number of the contained surfaces and trim curves can be found in Table 3 and 4.*

depending on the actual view point, the screen resolution and the depth complexity of the scene. In contrast, our compact trim curve data structure including the two-level hierarchy is computed once for each trimmed surface and the size only depends on the number of monotonic trimming curve segments.

The iterative Bézier Clipping approach by [Pabst et al. 2006] has lower memory requirements than our technique, since it simply uses a plain list of trim curves for each patch. However, the algorithm lacks an acceleration data structure and suffers from current GPU programming constraints. On the GPU, the temporary local storage needs to be a shader-compile-time constant. Bézier Clipping uses the de Casteljau algorithm to compute intermediate control polygons. The amount of required auxiliary points during subdivision as well as the intermediate control polygons depend on the curve degree. For such an approach the maximum curve degree is limited to about 5 on current graphics hardware. In contrast, our method only requires one curve evaluation per iteration. The Horner-like evaluation scheme directly accumulates from texture look-ups. Thus it has constant register usage and can handle curves of arbitrary degree.

	bumper	rim	VW Beetle
Bézier patches	380	1,202	53,554
Bézier curves	1,212	5,128	215,971
triangles	11,590	54,464	1,434,808
screen fill	60%	70%	70%
tessellation w/o trimming	0.3	0.6	11.7
tessellation w/ bisection	1.0	2.1	14.2
tessellation w/ secant method	1.0	2.0	14.0
tessellation w/ regula falsi	1.7	3.3	15.2
ray casting w/ bisection	18.8	105	93.6

Table 2: *This table shows the draw times in ms for our trimming method applied to tessellated NURBS models as well as to a ray casting approach. The cost for the per-pixel trimming is quite low and mainly depends on the fill rate. We also evaluated the performance of alternative root-finding algorithms such as secant method and regula falsi as alternatives to the bisection approach.*

degree	1	2	3	4	5	≥ 7
bumper	658	4	14	6	88	442
rim	2,275	-	2,853	-	-	-
VW Beetle	84,496	113	14,130	970	110,481	5,711

Table 3: *This table shows a degree distribution of the trimming curves of the models shown in Figure 11.*

degree	2	3	4	5	≥ 6	avg.
bumper	20	82	116	94	68	4.5
rim	30	10	10	245	913	5.9
VW Beetle	1,595	3,490	1,724	18,810	6,767	5.3

Table 4: *Degree distribution of Bézier surfaces for test models.*

For comparison reasons we implemented a CPU-version of the Bézier Clipping approach for trim curves as well as a CPU-version of our algorithm. Both approaches were used with bi-monotonic curve segments and our acceleration data structure to avoid root finding for points outside the bounding box of a curve segment. This implementation is not entirely fair towards our algorithm, since Bézier Clipping usually does not use a curve splitting approach and has to resolve multiple ray-curve intersections. Nevertheless, Bézier Clipping was about five times slower than our approach for real-world datasets such as the Beetle model. This result is based on the classification of a set of evenly distributed (u, v) point samples in the domain of each trimmed patch of the model. No actual rendering was performed. [Nishita et al. 1990] reported an average number of 1.7 iterations for Bézier Clipping. Using our acceleration data structure it dropped to 1.2. However, each Bézier Clipping iteration is quite involved including a convex hull generation, a de Casteljau subdivision and the handling of special cases. In contrast, our approach only needs one curve evaluation per iteration and typically required about 1.9 iterations for the tested models. This empirical result also confirms our estimate of an average of less than 2 iterations, which was derived in Equation 5.

Finally, we evaluated the performance of our complete GPU-based NURBS ray casting system including the presented trimming algorithm. The ray casting as well as the trimming algorithm deal with the original Beetle model without the use of any degree reductions or approximations. Thus, the model includes surfaces of up to degree 15 and trimming curves of up to degree 13. The performance depends highly on the surface degree and image resolution - more precisely on the number of generated fragments – since the Bézier surface evaluation within Newton's method has a complexity of $O(n^2)$. The performance is lowest for the rim, which mainly consists of high degree surfaces.

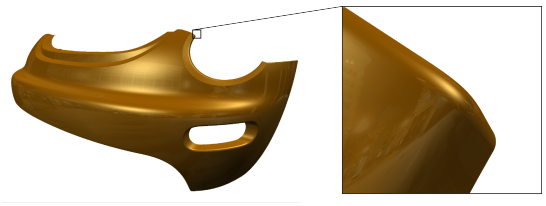


Figure 12: *This figure shows the bumper of a VW Beetle consisting of 380 Bézier surfaces. The frame rate for both views, the complete model as well as the close-up view, is about 50Hz. The close-up view shows the view-independent pixel-accuracy of our algorithm. The convex hull only consists of 11,590 triangles.*

6 Conclusions and Future Work

In this paper we presented a novel point classification method for curved regions with holes and applied this technique to direct trimming of NURBS surfaces. We demonstrated how to split the original trimming curves into segments such that the commonly used computationally expensive Bézier Clipping method can be replaced by an adapted bisectioning algorithm. Furthermore, we make use of the fact that the intersection point for the ray-based point-in-closed-curve test itself is not necessary. Our optimized bisectioning algorithm requires on average less than two trimming curve evaluations to classify a point contained in the curve's bounding box. A highly adapted acceleration data structure effectively limits the intersection candidates such that we can deal with complex trimming regions. Our point classification implementation supports trimming curves of arbitrary degree. It is integrated into a GPU-based single pass NURBS ray casting system, which is realized as a single OpenGL GLSL program. Our experience with the system shows that complex real-world models can be rendered at interactive frame rates.

Nevertheless, there is still room for improving the trimming approach. In particular, the combination with a coarse (u, v)-grid could avoid unnecessary binary searches in our hierarchical acceleration structure. If the lookup into the grid indicates the cell to be completely inside or outside the trimming domain no further computations have to be done. Otherwise the regular algorithm has to be used. Furthermore, we could further split the trimming curves to reach a statistically optimal computation time for the point-in-close curve test under the assumption of an even distribution of the point tests over the (u, v)-domain. The cost for a trimming curve evaluation is dominated by the number of texture fetches of the coefficients and thus known during the preprocessing phase. Since we know that we need on average about two curve segment evaluations for points in the bounding box of a curve segment, the overall influence on the number of texture fetches by introducing a split can be computed. The binary searches also need to be considered in this cost function. Their cost is dominated by the lookup of the v- and u-interval boundaries. Thus an optimization of the overall cost will create an optimal split of the trim curve segments under the given assumptions.

As pointed out by [Song et al. 2004], it is a hopeless task to exactly represent the intersection of two NURBS patches. This exactness is not essential for most applications, provided that trimmed surfaces agree exactly along approximate intersection curves without gaps or overlaps between them. They suggest control point perturbation schemes to enforce watertight patch intersections. However, this suggestion has not yet found its way into classical CAD systems as we found out: with large magnifications of adjacent trimmed patches, we sometimes find small pinholes along the trim curves or with even larger magnifications, gaps occur. While this problem should be resolved within the CAD systems, it could be also handled by the rendering system. One could even allow the user to switch between the two visualizations.

The most important next step is the integration of our approach into a CAD system. The small acceleration data structure can be generated and downloaded into the graphics card on the fly while trim curves are manipulated. Thus, the system can always deal with a pixel-precise representation of the trimming curves. For a large number of NURBS patches the performance of our ray-casting-based surface rendering algorithm is still limited on current GPUs, but the addition of occlusion culling techniques and the use of an early-ray termination test will significantly improve performance for scenes with medium to high depth complexity. We are convinced that direct GPU-based rendering of trimmed NURBS surfaces is a worthwhile alternative to the current tessellation-based

rendering pipeline. Trimming is just one, albeit very useful, application of our point classification technique. It could also be used for other applications where point classification with respect to curved regions with holes is required, including collision detection, stenciling, flood fills and vector graphics mapped onto arbitrary surfaces.

Acknowledgements

We thank Hans Pabst and Jan P. Springer for the many discussions during the early phase of this work and the Siggraph reviewers for their constructive feedback. The VW New Beetle model is courtesy of Volkswagen AG. The rim model is courtesy of Daimler AG.

References

BALÁZS, Á., GUTHE, M., AND KLEIN, R. 2004. Efficient trimmed nurbs tessellation. *Journal of WSCG 12*, 1 (Feb.), 27–33.

BENTHIN, C., WALD, I., AND SLUSALLEK, P. 2004. Real-time rendering: Interactive ray tracing of free-form surfaces. In *AFRIGRAPH '04: Proceedings of the 3rd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, 99–106.

COLLINS, G. E., AND LOOS, R. 1976. Polynomial real root isolation by differentiation. In *SYMSAC '76: Proceedings of the third ACM Symposium on Symbolic and Algebraic Computation*, ACM, New York, NY, USA, 15–25.

EFREMOV, A., HAVRAN, V., AND SEIDEL, H.-P. 2005. Robust and numerically stable bézier clipping method for ray tracing nurbs surfaces. In *SCCG '05: Proceedings of the 21st Spring Conference on Computer Graphics*, ACM, New York, NY, USA, 127–135.

FARIN, G. 1993. *Curves and Surfaces for Computer Aided Geometric Design (3rd ed.): A practical guide*. Academic Press Professional, Inc., San Diego, CA, USA.

FAROUKI, R. T. 1987. Trimmed-surface algorithms for the evaluation and interrogation of solid boundary representations. *IBM J. Res. Dev. 31*, 3, 314–334.

GEIMER, M., AND ABERT, O. 2005. Interactive ray tracing of trimmed bicubic bézier surfaces without triangulation. In *Proceedings of WSCG*, 71–78.

GUTHE, M., BALÁZS, A., AND KLEIN, R. 2005. Gpu-based trimming and tessellation of nurbs and t-spline surfaces. In *ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, 1016–1023.

JOY, K. I., AND BHETANABHOTLA, M. N. 1986. Ray tracing parametric surface patches utilizing numerical techniques and ray coherence. In *SIGGRAPH '86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, 279–285.

KAJIYA, J. T. 1982. Ray tracing parametric patches. *SIGGRAPH Comput. Graph. 16*, 3, 245–254.

KEYSER, J., CULVER, T., MANOCHA, D., AND KRISHNAN, S. 1999. Mapc: a library for efficient and exact manipulation of algebraic points and curves. In *SCG '99: Proceedings of the Fifteenth Annual Symposium on Computational Geometry*, ACM, New York, NY, USA, 360–369.

KUMAR, S., AND MANOCHA, D. 1995. Efficient rendering of trimmed NURBS surfaces. *Computer-Aided Design 27*, 7, 509–521.

MUDUR, S., AND KOPARKAR, P. 1984. Interval methods for processing geometric objects. *IEEE Comput. Graph. Appl. 4*, 2, 7–17.

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, New York, NY, USA, 337–345.

PABST, H.-F., SPRINGER, J., SCHOLLMEYER, A., LENHARDT, R., LESSIG, C., AND FROEHLICH, B. 2006. Ray casting of trimmed nurbs surfaces on the gpu. *Symposium on Interactive Ray Tracing*, 151–160.

PAVLIDIS, T. 1982. *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, Maryland.

QIN, Z., MCCOOL, M. D., AND KAPLAN, C. 2008. Precise vector textures for real-time 3d rendering. In *SI3D '08: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, 199–206.

ROCKWOOD, A., HEATON, K., AND DAVIS, T. 1989. Real-time rendering of trimmed surfaces. In *SIGGRAPH '89: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, 107–116.

SEDERBERG, T. W., AND WANG, X. 1987. Rational hodographs. *Computer Aided Geometric Design 4*, 4 (Dec.), 333–335.

SEDERBERG, T. W. 1995. Point and tangent computation of tensor product rational bézier surfaces. *Computer Aided Geometric Design 12*, 1, 103–106.

SONG, X., SEDERBERG, T. W., ZHENG, J., FAROUKI, R. T., AND HASS, J. 2004. Linear perturbation methods for topologically consistent representations of free-form surface intersections. *Computer Aided Geometric Design 21*, 3, 303–319.

TOTH, D. L. 1985. On ray tracing parametric surfaces. *SIGGRAPH Computer Graphics 19*, 3, 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. *The Journal of Visualization and Computer Animation 11*, 4 (Sept.), 209–219.