Direct and Fast Ray Tracing of NURBS Surfaces

Oliver Abert* Markus Geimer† Stefan Müller‡
Universität Koblenz-Landau Forschungszentrum Jülich Universität Koblenz-Landau

Abstract

Recently it has been shown that Bézier surfaces can be used as a geometric primitive for interactive ray tracing on a single commodity PC. However, the Bézier representation is restricted, as a large number of control points also implies a high polynomial degree, thus reducing the frame rate significantly. In this work we will present a fast, efficient and robust algorithm to ray trace trimmed NURBS surfaces of arbitrary degree. Furthermore, our approach is largely independent of the number of control points of a surface with respect to the rendering performance. Additionally the degree and the number of control points of a surface do not influence the numerical stability of the intersection algorithm.

The desired high performance is achieved by taking a novel approach of surface evaluation, which requires only minimal preprocessing. We will present a method to transform the computationally expensive Cox-de Boor recursion into a SIMD suitable form that maximizes performance by avoiding the recursion and drastically reduces the number of executed commands.

Keywords: Ray tracing, NURBS, free-form surfaces, interactive rendering.

Index Terms:
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

1 Introduction

Free-form surfaces, including Bézier, B-Spline, or NURBS representations, offer a powerful way to easily and exactly describe curved surfaces for use in computer graphics applications. However, in most circumstances these surfaces are converted into triangular meshes before rendering, no matter whether ray tracing or scanline rendering is employed. Until today, graphic cards are not able to render geometry objects other than triangles. By contrast, with ray tracing it is generally possible to use any geometry with which a ray intersection can be computed.

Nevertheless, most ray tracing systems also handle only triangles as their geometric primitive. However, directly rendering free-form surfaces instead has several advantages. Real-world data sets, as they are found in the automobile industry for example, frequently require extensive triangulation preprocessing (manually as well as automatically) which can take up to more than a full day. Obviously, with ray tracing there is no need for such a time-consuming triangulation, which often introduces artifacts as well. Furthermore, memory consumption is reduced, since a single surface can represent the same geometry that needs to be modeled by thousands of triangles otherwise, depending on the desired detail. In addition, the accuracy is not limited to a certain distance from the observer, since the surfaces will always stay perfectly curved even from the shortest viewing distance. Last but not least, trimming curves are often used to cut out irrelevant parts of free-form surfaces. A triangulation of trimmed NURBS patches is still frequently causing problems as artifacts and long preprocessing times are introduced, whereas integrating an online trimming test in a ray tracing approach is quite efficient and feasible. In fact, the system presented in this paper is capable of efficiently handling trimmed surfaces as well.

A possible solution to avoid triangulation is to convert NURBS to Bézier surfaces as presented in [8] or [2]. Naturally, the Bézier representation can be used to describe curved surfaces, however, the Bernstein polynomials, which form the Bézier basis, have their degree determined by the number of control points. This increases computation and memory requirements significantly when increasing the number of control points, thus leaving only quite simple Bézier surfaces (like bicubic patches with 4×4 control points) suitable for interactive use. By employing only low order Bézier surfaces it would be necessary to use many patches for a surface that can be represented easily with a single NURBS surface. In the end an approach using Bézier surfaces suffers from similar problems as a classical triangulation approach: long preprocessing times as well as the introduction of artifacts, especially at the surface boundaries, although their number is reduced compared to a triangulation.

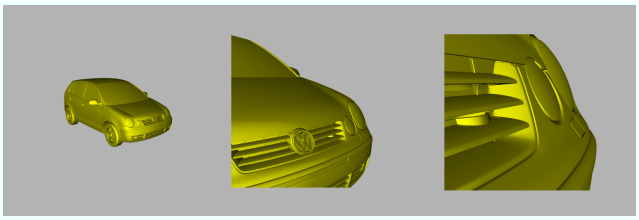

Figure 1: A model of the VW Polo from different viewpoints. The car remains perfectly curved, even from the shortest viewing distance, thanks to the direct ray tracing of NURBS.

With direct ray tracing of NURBS, we present in this paper a solution that features all advantages but does not suffer from the problems mentioned, at the cost of a justifiable portion of rendering speed. Nevertheless, achieved performance for non-trivial scenes is still above one frame per second on a single PC, though using multiple CPU cores. The actual intersection algorithm is basically the same Newton iteration-based approach as presented in [8], however, the NURBS surface evaluation is novel and explained in detail. Surface evaluation is essential for the performance of the entire system, since compared to a conventional ray tracer, a lot more time is spent in the actual intersection algorithm than in the traversal of an acceleration data structure. We found that roughly 60% of the CPU time is spend in the surface evaluation, including the calculation of partial derivatives. Hence the optimization of these algorithms is a performance critical task.

*e-mail: abert@uni-koblenz.de
†e-mail: m.geimer@fz-juelich.de
‡e-mail: stefanm@uni-koblenz.de

2 Previous Work

During the last years several algorithms that allow for ray tracing free-form surfaces have been developed, though most of them have not been targeted towards high frame rates. The realization of interactive ray tracing itself was presented just lately. Approaches in both fields will be discussed briefly in this section.

2.1 Free-Form Surface Ray Tracing

The pioneering work in the field of free-form surface ray tracing was presented by Toth [24] and Kajiya [13]. The former was first to employ an iterative Newton approach to find a ray-surface intersection. Initial estimates required for the Newton iteration to converge were determined analytically, which on the one hand is very reliable, but on the other hand is also very expensive to compute online, thus it is questionable if interactive frame rates can be achieved using this approach in the way presented. By contrast, Kajiya first solved the problem of ray and parametric surface intersection without performing a subdivision. By using numerical procedures he was generally able to solve the problem. However, interestingly he stated that for a real time computation approximately 33ns computation time per ray needs to be achieved - which was rather utopian back in 1982, whereas Table 1 shows that we are now not too far away from this mark.

Later, Sweeney et al. presented in [23] another approach based on a Newton iteration for B-Splines. The initial estimate, however, was acquired by computing the intersection of the ray and a refined control mesh of the surface. In [16], Martin et al. similarly used a Newton iteration approach, but in contrast they used a hierarchy of small bounding volumes which closely approximates the surface, thus yielding a reliable initial estimate, using the center of the parametric interval each bounding box encloses.

Nishita et al. presented in [20] the so-called *Bézier Clipping* approach. By exploiting the Bézier convex hull property, regions of the surface which are known not to intersect the ray are cut away iteratively, thus isolating the region where an intersection point is to be found. Campagna et al. [3] further improved and optimized this algorithm. Unfortunately, more complex surfaces such as NURBS can not be used directly with this approach. They have to be converted into a Bézier representation beforehand.

Later, Wang et al. combined the Newton iteration approach with Bézier Clipping in [28]. By taking advantage of the ray coherence between neighboring primary rays, they were able to improve the performance by up to a factor of three compared to the original implementation of Nishita.

Ray tracing of subdivision surfaces was investigated by Kobbelt et al. in [14] as well as Müller et al. in [18]. Both approaches are rather slow as they have to deal with a lot of special cases, especially at edges and surface boundaries.

2.2 Interactive Ray Tracing

Interactive ray tracing was first presented by Muuss et al. in [19], though their approach was limited to combinatorial solid geometry models. Four years later, Parker et al. [21] presented the first full-featured interactive ray tracer, which was even able to handle NURBS surfaces. However, very expensive shared memory supercomputers were required, i.e., 60 processors were necessary to speed up NURBS scenes of moderate complexity to interactive frame rates.

Wald et al. [27, 26] introduced a highly optimized ray tracer running on a cluster of standard PCs. By using SIMD instructions (**S**ingle **I**nstruction **M**ultiple **D**ata) and carefully paying attention to coherence, caching issues, and data layout, they were able to achieve a speed-up of more than an order of magnitude compared to conventional ray tracing systems. Though the system has even been further improved [25], it is still restricted to triangles.

Recently Benthin et al. [2] presented a method to ray trace Bézier and Loop subdivision surfaces at interactive frame rates. However, they use a subdivision method that refines the control mesh on the fly, yielding an approximate intersection point using a triangle mesh generated from the control points.

Lately, an approach to efficiently ray trace trimmed NURBS surfaces was presented in [6, 5]. Though Efremov et al. implemented a robust and numerically stable algorithm that achieves visually good results, they do not reach interactive frame rates. Furthermore, NURBS surfaces are not handled directly but converted to a rational Bézier representation beforehand. Simple scenes require between three and ten seconds to render on a fast PC, whereas more complex scenes even require up to several minutes.

Geimer and Abert presented in [8] the interactive ray tracing of bicubic trimmed Bézier surfaces. By using a Newton iteration similar to the one employed in [16] and a highly optimized implementation (using SIMD instructions, ray and cache coherence, iterative bounding volume hierarchy traversal), they were able to achieve more than three frames per second for non-trivial scenes with thousands of trimmed Bézier surfaces on a single processor. The underlying interactive ray tracing system, which is also capable of rendering triangle meshes, is presented in detail in [7].

Finally, Abert presented in [1] an approach, which is also based upon [7]. By using a power basis representation for NURBS surfaces, he was able to achieve interactive frame rates for scenes with NURBS surfaces of low complexity, due to the restriced accuracy of single-precision values. During a preprocessing step all basis functions are converted into a polynomial representation, which makes a fast evaluation possible. Furthermore, the use of GPUs for NURBS ray tracing was investigated. Unfortunately, it was shown that current GPUs are not yet powerful enough for NURBS surfaces of arbitrary complexity.

3 System Overview

This section gives a brief survey of the basic system architecture (see [7] for details), before going into the details of the ray-NURBS intersection algorithm.

All performance critical computations are completely performed using SIMD instructions by processing four rays in parallel. This includes the traversal of the acceleration data structure, the actual intersection and trimming test, as well as shading. Unlike [27], we have built our system upon our own SIMD abstraction layer, currently offering support for Intel's SSE [12] and Motorola's AltiVec [17] instruction sets. Additionally, an FPU mode is able to emulate a SIMD unit. Due to it's modular structure, the abstraction layer could be easily extended to support SIMD instruction sets of other architectures as well.

Just like for triangle-based ray tracing systems, it is important to reduce the number of intersection tests performed. Here, it is even more indispensable, as the test with a NURBS surface is obviously a lot more expensive than the test against a triangle. In this context, *kd*-trees are usually considered to be faster than bounding volume hierarchies, however, we use the latter due to the fact that an iterative traversal of axis aligned bounding boxes in depth-first order [22] is well suited for the usage of SIMD instructions and the hierarchy can additionally provide good initial estimates required for the Newton iteration. To achieve a good performance the hierarchy is created using the heuristic of Goldsmith and Salmon [9].

Naturally, ray tracing is well suited for parallel execution, and our system is no exception to that. The application can be run with an arbitrary number of threads, thus supporting multiprocessor and/or multicore PCs as well as Intel's HyperThreading technology [11].

Our system is currently limited to static scenes, which allows for interactive walk-throughs only. However, it can be easily extended to support dynamic scenes with hierarchical movements using the ideas presented in [15] and [26].

4 Our Approach

This section will give details about the approach we have taken. First, we explain the modifications to an axis-aligned bounding volume hierarchy necessary for the intersection test. Afterwards, we give insight into the measures taken during preprocessing, where an enhanced NURBS representation featuring a collection of *CDBItems* (**C**ox-**d**e **B**oor item - see Subsection 4.2) permits an extremely fast evaluation algorithm.

By paying attention to the characteristics of modern processor architectures (e.g., very long pipelines), we favour a rather "brute force" approach instead of a very complex algorithm. As was shown in [27] or [2], a carefully optimized implementation is often able to easily outperform more complex algorithms on current CPU architectures.

4.1 NURBS Surface Representation

A NURBS surface is given by the equation

$$S(u,v) = \sum_{i=1}^{n} \sum_{j=1}^{m} B_{i,j}^{h} N_{i,k}(u) M_{j,l}(v) \tag{1}$$

where the $B_{i,j}^{h}$ are the control points in homogeneous coordinate space. A control point is given by (x,y,z,w), where w denotes a weighting factor. The $N_{i,k}$ are the basis functions in u parametric direction and $M_{j,l}$ analogously for the v direction. Furthermore, n and m are the number of control points, whereas k and l denote the order of the basis function in u and v parametric direction, respectively.

The basis functions are given by the Cox-de Boor recursion formula [4]:

$$N_{i,k}(t) = \frac{(t-x_i)N_{i,k-1}(t)}{x_{i+k-1}-x_i} + \frac{(x_{i+k}-t)N_{i+1,k-1}(t)}{x_{i+k}-x_{i+1}} \tag{2}$$

The recursion will stop if k equals 1. In this case the following equation applies:

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } x_i \le t < x_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

In addition, the convention $\frac{0}{0} = 0$ is adopted for equation (2).

The x_i are elements of a knot vector, which is a monotonically increasing series of numbers with $n+k$ elements ($m+l$ respectively). Equations 2 and 3 do hold analogously for the parametric v direction. Knot vectors come in two flavours, open and periodic, where the former have multiplicity of knot values at the ends equal to the order k, while periodic knot vectors do not have this constraint. Additionally they can be uniform (equidistant inner knot values) or non-uniform (arbitrary positive distance between two consecutive values).

Obviously, the straight-forward evaluation of equation (1) is not advisable as the computational demand increases extremely fast with larger numbers of control points and/or a higher order. Furthermore, a recursive function call becomes expensive when executed in the innermost loop of an algorithm. Because a single intersection test requires several surface evaluations (i.e., one surface and two partial derivative evaluations per iteration step - see Subsection 4.4), one of our main objectives was to find a fast method to evaluate the basis functions by maintaining useful properties like numerical stability and robustness.

4.2 SIMD Cox-de Boor Representation

As a first step, we rewrite the Cox-de Boor recursion formula into a form that will allow for a more efficient execution and storage in memory using SIMD instructions. Equation (2) can be rewritten as

$$N_{i,k}(t) = c_1(t-x_i)N_{i,k-1}(t) + c_2(x_{i+k}-t)N_{i+1,k-1}(t)$$

where $c_1 = \frac{1}{x_{i+k-1}-x_i}$ and $c_2 = -\frac{1}{x_{i+k}-x_{i+1}}$. Further substitution yields

$$N_{i,k}(t) = \underbrace{\underbrace{(c_2 t + c_4)}_{simd\_madd} N_{i+1,k-1} + \underbrace{(c_1 t + c_3)}_{simd\_madd} N_{i,k-1}}_{simd\_mul}}_{simd\_madd}$$

where now $c3 = -x_i c_1$ and $c4 = -c_2 x_{i+k}$. The underbraces denote the SIMD commands used for computation during runtime. This way only three combined multiply-add (madd) and one multiply command is required, which is achieved by computing $c_i, i \in [1,4]$ during preprocessing as these are invariant with respect to t. However, all four c_i have to be precomputed for all basis functions $N_{i,k}$ and $M_{j,l}$. By contrast, the functions for $k=1$ and $l=1$ are an exception as these are always 1 in our interval of interest. Fortunately, the four c_i float values perfectly fit into a single SIMD variable[1], which we call *CDBItem*. Analogously the same approach of identifying constant values and calculating them in advance works for the derivatives as well.

As can be seen, a *CDBItem* stores all the information that is required to compute the value of a specific basis function $N_{i,k}$ or its derivative in parametric u direction, if

- the parameter value u is known (which is the case during runtime),
- both lower order basis functions $N_{i,k-1}$ and $N_{i-1,k-1}$ are known.

As mentioned above, all $k=1$ order basis functions are an exception to that. The same holds analogously for $M_{j,l}$.

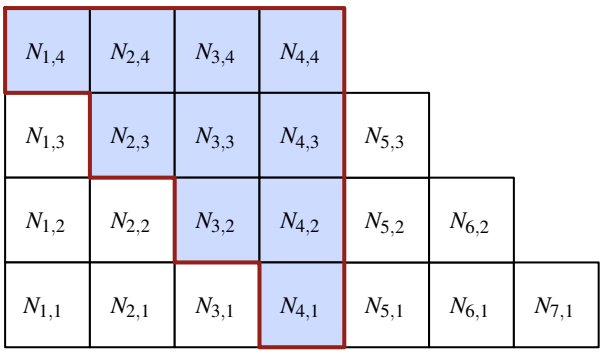

Figure 2: Dependencies of the basis functions (here: order $k=4$ and number of control points $n=4$). Only the elements within the red border are different from zero.

Figure 2 exemplarily shows the relevant basis functions $N_{i,k}$, shaded in light blue, for the open, uniform knot vector $[0\ 0\ 0\ 0\ 1\ 1\ 1\ 1]$, order $k=4$ and number of control points $n=4$.

[1] SIMD registers on SSE and AltiVec are 128 bit wide which is sufficient for storing four 32 bit floating point values.

The remainder do not need to be computed as they are always zero. Note that every $N_{i,k}$ with $k > 1$ basis function depends on basis functions $N_{i,k-1}$ and $N_{i+1,k-1}$, except for those at the beginning (if $i+k = n+1$ in this example) or end (if $i = n$) of a row, which are dependent only on either $N_{i,k-1}$ or $N_{i+1,k-1}$. For instance, if the Cox-de Boor formula has to be evaluated for a given parameter value of $t = 0.5$ then $N_{4,1}$ will be 1 (and all other $N_{i,1} = 0$). Resolving the dependencies from bottom to top, we can now compute $N_{3,2}$ and $N_{4,2}$. Afterwards, the three basis functions $N_{i,3}, i > 1$ and finally all four basis functions $N_{i,4}$ can be computed. Especially, all $N_{i,k}$ with maximum k are important as these are needed to solve equation (1).

The pyramid shown in Figure 2 will become broader for an increasing number of control points and higher for an increasing degree as depicted in Figure 3. However, evaluating the basis functions for a specific parameter value always results in exactly one particular $N_{i,1}$ to be different from zero, thus the processing time for a single surface evaluation is largely independent from the number of control points. As a matter of course there are still effects that may reduce performance (e.g., fewer cache hits or insufficient bus bandwidth). However, the only significant performance loss arises from shortened SIMD effectivity. Parameter value pairs that lie in different intervals have to be computed sequentially. The more control points a specific surface has, the smaller the intervals will become. However, if possible, the results will be merged into SIMD registers, which allows the second part of the computation (evaluating equation 1 with given N and M) to be processed in full parallelism, if the rays hit different surfaces with the same number of control points and equal degree. Even in the worst case, i.e., four rays hitting four different intervals, the performance will be better compared to a sequential FPU-based implementation due to the streaming architecture of the SIMD unit. The results presented in Section 6 verify, that even with a large number of control points on a single surface, the performance still remains high by all means. Obviously, the degree still influences computation time.

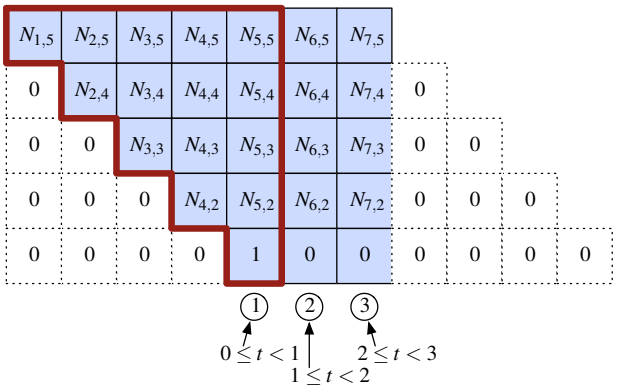

Figure 3: The red triangle marks all basis functions that need to be evaluated for a curve with order $k = 5$ and $n = 7$ control points. For the triangle to be positioned correctly, the parameter value t must satisfy $0 \le t < 1$, while assuming a knot vector of $[0\ 0\ 0\ 0\ 0\ 1\ 2\ 3\ 3\ 3\ 3\ 3]$. Depending on the parameter value and knot vector, the red triangle has to be positioned either at location 1, 2 or 3. For instance, if a parameter value of $t = 2.6$ would be used, the red triangle needs to be positioned at position 3 instead. Notice: For increasing n (or m) the triangle stays unaltered, merely increasing k (or l respectively) will heighten it and therefore increase the computational cost for an evaluation.

4.3 Preprocessing

In this section we describe the preprocessing step, where the acceleration data structure is generated, all *CDBItems* are calculated as described in 4.2 , and stored along with the appropriate surface.

As with every ray tracer, an acceleration data structure is very important. In our system, the need for an efficient data structure is even higher, as the intersection test is much more complex compared to well known triangle tests. Therefore, it is even more important to avoid unnecessary tests. Additionally, we need good initial estimates to start the Newton iteration, which can also be provided by the hierarchy.

In the majority of cases, conventional ray tracers use either a *kd*-tree or a bounding volume hierarchy with axis-aligned bounding boxes as acceleration data structure. In contrast to [2] we choose a bounding volume hierarchy, as it is more expensive to compute initial estimates from a *kd*-tree. In [2], Benthin et al. are subdividing the control mesh and computing the final intersection with a triangle approximating the surface at this point in space. However, our approach is quite different from that, as we are avoiding subdivision schemes and approximated triangle intersections. Employing a bounding volume hierarchy, we simply can use the center of the bounded parametric interval defined by the bounding boxes we are already using as acceleration data structure.

The major difference to an ordinary bounding volume hierarchy is that we do not store a whole or even several surfaces in a single volume, but the union of several volumes bound a single surface instead. This has two advantages, provided that the volumes are small enough:

1. The number of rays that hit the volume but miss the geometry is reduced significantly.

2. The center of the bounded parametric region can be used as a good initial estimate for the Newton iteration.

Figure 4 gives an example in 2D. Every bounding box stores the bounded parametric interval. Here, ray A and B hit the blue box and start the iteration using $t = \frac{0.45+0.6}{2} = 0.525$, whereas ray C intersects the yellow box using $t = \frac{0.75+1.0}{2} = 0.875$ instead. However, if only a single bounding box (dashed outer lines) would have been used, all iterations would start with $t = 0.5$. The iteration will most likely diverge or yield wrong results in regions that are too far away from the midpoint. In the given example, the regions in question are marked red, whereas the green area denotes the interval where $t = 0.5$ is still an acceptable guess.

Currently, the bounding volumes are created using a simple flatness criterion, i.e., if the evaluation of a predefined set of normal vectors indicate a surface with high curvature then recursively more and smaller bounding volumes are generated. However, artifacts, i.e., wrong or missed intersections, might be introduced, as this approach uses a user defined θ value to stop the generation process. There will be too few and inaccurate bounding volumes if this parameter is chosen to large. Nevertheless, it is quite easy to manually find a suitable θ value ensuring the convergence of the Newton iteration. Even by choosing a conservative value, the number of generated bounding boxes might be larger than necessary, but having no significant impact on the rendering speed. However, using the ideas presented in [24], it would be possible to analytically compute bounding volumes, thus improving this stage during preprocessing by removing a potential source for errors.

4.4 Intersection Test

Beyond the surface evaluation, the intersection test algorithm is similar to the approach taken by [16]. They generally solved the problem, but have not targeted at interactive frame rates.

In our approach, we represent the ray by two orthogonal planes $P_1 = (N_1, d_1)$ and $P_2 = (N_2, d_2)$, where the N_i are orthogonal vectors

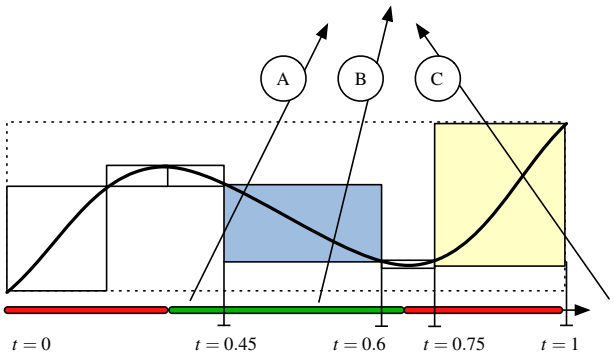

Figure 4: Example of a NURBS curve with exemplary associated bounding volumes.

of unit length, perpendicular to the ray direction D. The d_i are given by $d_i = N_i O$ with O denoting the ray origin. To find the intersection of a surface $S(u,v)$ and a ray, we have to find the roots of

$$R(u,v) = \begin{bmatrix} N_1 S(u,v) + d_1 \\ N_2 S(u,v) + d_2 \end{bmatrix} \tag{4}$$

R becoming zero indicates that the distance of the evaluated point on the surface to both planes is also zero, hence an intersection point is found.

There are several numerical methods with which this equation can be solved, however, we have chosen the Newton iteration due to its advantageous properties like (mostly) quadratic convergence, given a good initial estimate, and easy computation, even though the first derivative is required. Nevertheless, the partial derivatives can be computed in an efficient way as well. Additionally, this iteration is well suited for SIMD execution, unlike other approaches using a more complex control flow.

We have to solve a two-dimensional problem. Therefore, the Newton iteration step is then given by

$$\begin{bmatrix} u_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} u_n \\ v_n \end{bmatrix} - J^{-1} R(u_n, v_n)$$

where J is the Jacobian matrix of R given by

$$J = \begin{bmatrix} N_1 S_u(u,v) & N_1 S_v(u,v) \\ N_2 S_u(u,v) & N_2 S_v(u,v) \end{bmatrix}$$

Here, S_u and S_v denote the partial derivatives in the corresponding parametric direction.

The iteration will be continued until one of the following termination criteria is met:

1. If the distance to the real root falls below some user defined threshold ε, then an intersection point is found, i.e.

$$|R(u_n, v_n)| < \varepsilon$$

2. Whenever the iteration takes us further away from the root, the computation will be aborted, assuming divergence.

$$|R(u_{n+1}, v_{n+1})| > |R(u_n, v_n)|$$

3. A maximum number of iteration steps has been performed, also indicating divergence.

Due to the usage of SIMD instructions, the iteration is not finished until all four parallel computations are completed. Nevertheless, often, especially for coherent primary rays, an intersection is found during the same iteration step. Due to this fact, the performance is still improved by more than a factor of three compared to a sequential execution.

5 NURBS Surface Evaluation

As mentioned before, the NURBS surface evaluation is an integral part of the whole ray tracing system. Including the partial derivatives, approximately 60% of the processor time is spent in the surface evaluation, depending on the scene. In this section it is shown how the highest number of evaluations per second to date (to the knowledge of the authors) can be achieved.

Basically, the task is to solve equation (1) for any given u and v parameter values. A straight-forward evaluation is, as mentioned before, highly inadvisable, since there is a lot of redundancy due to the recursion, thus resulting in vast amounts of unnecessarily performed computations. This is where our Cox-de Boor cache (*i.e., CDBCache*) implementation will help to significantly improve the performance.

Our approach basically involves three important key routines, which are:

- Compute a single basis function using the appropriate *CDBItem* data set
- Fill a temporary cache for u and v parameter values
- Evaluate the NURBS equation using these two caches

Figure 5 shows an example for filling the cache with $k = 4$ and $n = 4$ for u parametric direction (this process has to be done a second time with the v direction data set in case of a surface). $N_{4,1}$ is known to be 1, since $x_i \leq t < x_{i+1}$. All basis functions are computed in the order given in Figure 5.

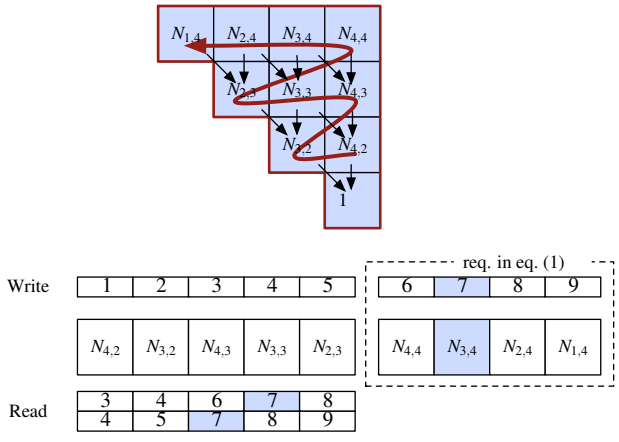

Figure 5: The red arrow denotes the order of computation of the individual *CDBItems* where the small black arrows denote data dependencies. The numbers at the bottom indicate when items are written or read. $N_{3,4}$ for example is written in step 7 and is dependent on $N_{4,3}$ (written at step 3) and $N_{3,3}$ (step 4). This approach minimizes data dependencies, and thus processor stalls, as much as possible.

The following is an example for computing a basis function $N_{i,k}$ in pseudo code

$$
\begin{aligned}
&N_{i,k} = simd\_madd(\\
&\ simd\_madd(c_1, t, c_3), nik_1, \\
&\ simd\_mul(simd\_madd(c_2, t, c_4), nik_2) \\
&);
\end{aligned}
$$

where t is the current parameter that must satisfy $x_i \leq t < x_{i+1}$. Furthermore, the c_i are the precomputed values of the corresponding *CDBItem* for $N_{i,k}$, where nik_1 corresponds to the result of $N_{i,k-1}$, analogously nik_2 to $N_{i+1,k-1}$. Note that if one of them lies outside the red triangle of Figure 3, the value for that particular nik_i is zero.

After the caches for both u and v parametric directions have been computed, i.e., all $N_{i,k}$ and $M_{j,l}$ with k and l being equal to the surface order, equation (1) can finally be solved. However, due to the local control property of B-Splines, only a limited area of the surface is affected, depending on the degree. It is not necessary to compute the whole equation, except if $n = k$ and $m = l$. Otherwise equation (1) reduces to

$$S(u,v) = \sum_{i=intU+1}^{intU+k+1} \sum_{j=intV+1}^{intV+l+1} B^h_{i,j} N_{i,k}(u) M_{j,l}(v) \qquad (5)$$

where $intU$ and $intV$ specify the current knot vector interval number being computed. For instance, given a knot vector of [0 0 0 1 2 2 2] would yield two intervals, i.e., $I_1 = [0,1]$ and $I_2 = [1,2]$.

Equation (5) regards the fact that

$$N_{i,k} = 0 \Rightarrow i < intU + 1 \vee i > intU + k + 1$$

respectively the same applies for $M_{i,k}$. Hence a great amount of work can be skipped, especially when intersecting surfaces with a small degree and a large number of control points. Such surfaces occur frequently in real-life CAD data, where spatially small surfaces with thousands of control points are no rarity. In particular this applies to trimming curves as well.

6 Results

In this section we present experimental results achieved using the techniques described above. See Figures 6 and 7 for some examples. All rendered images are of 512^2 pixels resolution. The tests were performed on a PowerMac G5 with two dual-core processors, running at 2.5 GHz (1.25 GHz bus speed, 64KB L1 instruction cache, 32KB L1 data cache, 1024 KB L2 cache per CPU) with 2.5 GB of PC4200 RAM using AltiVec and four threads to take advantage of all CPU cores. We also compared our results with two production systems in Table 2.

We claimed earlier that the performance of our algorithm is largely independent of the number of surface control points. Table 1 confirms our statement (compare column one and two in both rows). The recorded loss in performance is due to the reduced effectivity of the SIMD instructions. At a certain close distance the frame rate is increasing again, since in such a case the SIMD effectivity is growing again. Evaluations are most effective if all four parameter values lie within the same knot vector interval of their parametric direction. Therefore increasing the number of control points together with constant spatial expansion will result in smaller intervals, thus enforcing more sequential operations. Nevertheless, beyond that no significant performance loss is occurring.

Our system is also able to handle trimming curves in an efficient way. Since our research in the field of efficient computation of trimmed NURBS surfaces begun just lately, we are not focussing on that topic here. Currently we are basically using the same techniques described in this paper, which we adapted for the 2D case. By finding the closest intersection with a 2D trimming curve we can compute whether the trimming ray is leaving or entering a trimmed region. This is only possible if all trimming curves have a consistent orientation. Following this approach we are able to take advantage of the highly optimized algorithms we already developed. In Figure 7, a VW Polo model is shown which heavily uses trimming curves. As can be seen in Table 1, the overhead of trimming is only 6.7% for this model, though more than half of all surfaces have trimming curves assigned (38354 trimming curves in total). At present arbitrary NURBS curves are supported as trimming curves in contrast to other systems which in almost all cases only support Bézier curves.

Finally the results show, that the memory usage is quite low. Although we use several bounding boxes for a single surface, the memory usage for the bounding volume hierarchy alone is comparable to a triangle based ray tracer, as the latter uses a lot more triangles than we use surfaces. The memory required for the surfaces and all precomputed data is even lower. For example, the entire Polo scene uses less than 40 MB in total. With such a low memory usage, it is possible to easily ray trace very complex scenes.

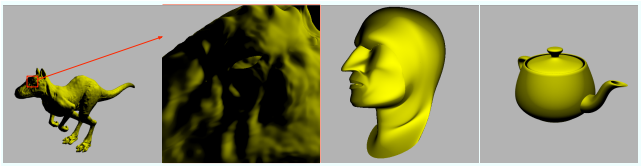

Figure 6: The Killeroo modell [10] with 56625 control points is shown on the left side, while the next shows a closeup shot from its head (See columns two and three in the first row of Table 1). Note, that the frame rate drops from 3.26 to 1.54 frames per second. However by taking into account the screen coverage increasing from 14.1% to 92.3% the result is quite satisfying. The average intersection time for a single test even decreased from 591 ns to 319 ns due to a more efficient SIMD usage at such a short distance. The same effect can be observed with the Polo model. Pictures on the right: A head modeled with 915 bicubic 4×4 NURBS patches and the well-known Utah teapot, which uses a NURBS representation instead of the original Bézier surfaces.

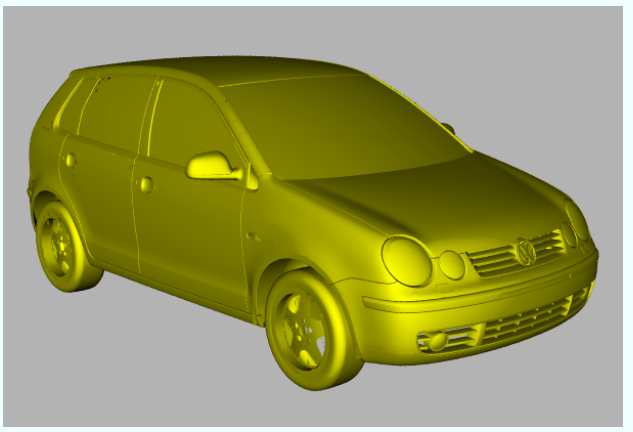

Figure 7: Trimmed model of a VW Polo. These renderings are based on a direct export from Catia V4 engineering data of the Volkswagen AG. The original data set included only half of the model, so Maya was used for mirroring only - no further optimizations were applied. Thus the model contains *"evil"* surfaces with thousands of control points and a high degree (up to 15) for visually less important tiny geometric details. As the results demonstrate, the system is able to handle even such surfaces efficiently.

Statistics	Killeroo	Killeroo	Killeroo closeup	Teapot	Head
NURBS Surfaces	89	89	89	32	915
Screen Coverage	14.1%	14.1%	92.3%	21.16%	43.19%
Average Order	4	4	4	4	4
Number of Control Points	17181	56625	56625	512	14640
Number of Bounding Volumes	104182	130865	130865	12984	13844
Frames per second	3.81	3.26	1.54	7.22	4.37
Intersection Tests/frame	329528	339812	1469232	265712	497000
Total Intersection Time	0.169 s	0.201 s	0.47 s	0.061 s	0.135 s
Average Intersection Time	512 ns	591 ns	319 ns	229 ns	287 ns
Memory Consumption (NURBS)	0.38 MB	1.07 MB	1.07 MB	0.02 MB	0.62 MB
Memory Consumption (BVH)	3.57 MB	4.49 MB	4.49 MB	0.44 MB	0.47 MB
Preprocessing Time	1.57 s	2.49s	2.49 s	0.14 s	0.49 s
Statistics	Single Surface	Single Surface	Polo (trimmed)	Polo (untrimmed)	Polo closeup
NURBS Surfaces	1	1	9659	9659	9659
Screen Coverage	32.6%	32.6%	11.9%	12.2%	81.1%
Average Order	4	4	5.47	5.47	5.47
Number of Control Points	16	162409	423416	42316	42316
Number of Bounding Volumes	58	58	604191	604191	604191
Frames per second	9.01	5.53	1.83	1.96	0.70
Intersection Tests/frame	134992	135324	520348	484856	2111640
Total Intersection Time	0.027 s	0.069 s	0.451 s	0.401 s	1.24 s
Average Intersection Time	200 ns	509 ns	866 ns	827 ns	587 ns
Memory Consumption (NURBS)	0.695 KB	0.695 KB	15.75 MB	15.75 MB	15.75 MB
Memory Consumption (BVH)	2.03 KB	2.03 KB	20.74 MB	20.74 MB	20.74 MB
Preprocessing Time	0.0025 s	1.477 s	52.54 s	52.54 s	52.54 s

Table 1: Achieved frame rates and measured timings with different scenes on a quad-processor PowerMac G5 running at 2.5 GHz with 512^2 image resolution. Four threads and SIMD instructions are used. Preprocessing is currently still single-threaded and therefore no advantage is taken of the additional processors. Note that for the Polo scenes the average intersection time includes the time required for the trimming test.

7 Comparison

Only few people have worked on interactive ray tracing of NURBS surfaces, thus a comparison is rather difficult. Efremov [5] presented some results in his master's thesis. On a 3.06 GHz Xeon CPU he achieved a frame rate of 0.22 frames per second using a 800^2 image resolution with a scene comparable to our *head* scene (976 surfaces, 15616 control points, average order of 4.0). By roughly taking the difference in resolution and processor speed into account, this still leaves us a speed-up factor of approximately three. However, due to the fact that rational Bézier surfaces rather than NURBS are used, his approach does not scale well with an increasing surface degree. Rendering a more complex scene with 64659 patches, 1261136 control points, and an average degree of 7.58 requires 8.61 minutes to compute. Although we currently do not have a comparable scene available, we expect our system to render such a scene in clearly less than 10 seconds.

By contrast, Martin et al. [16] also traced NURBS surfaces directly. However, the numbers they presented were measured on a 300 MHz MIPS R12000 processor, which makes it quite difficult to compare the results, due to the very different hardware platform.

Compared to our own Bézier based ray tracing system presented in [8] we achieve a lower frame rate, which is obvious as NURBS are a lot more complex than their Bézier counterparts. However, we were able to increase the image quality while decreasing the preprocessing time at the same time. The data provided by Volkswagen was not directly usable with the Bézier-based system since the NURBS surfaces had to be converted into a bicubic Bézier representation, which involved a lot of manual work. Additionally, due to the conversion, the Bézier based model suffers from visual artifacts (see Figure 8), mainly gaps between surfaces. The frame rate achieved for the Polo scene using the Bézier-based system is 6.92 frames per second, which means, the approach presented in this paper is currently *"only"* 3.7 times slower. However, note that there was a lot of manual preprocessing work included (i.e., converting the NURBS surfaces into bicubic 4×4 patches). Without such a conversion the rendering time would have been far from interactive due to the global control property of Bézier surfaces. Also note the problems of this approach shown in Figure 8.

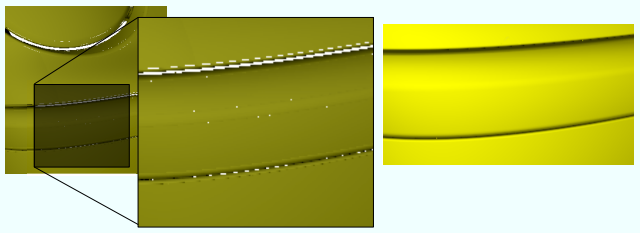

Figure 8: Artifacts occurring in a Bézier-based approach (left) [8]: Gaps between Bézier surfaces due to data conversion. Such artifacts can not appear in a NURBS based approach as presented in this paper (right). Note, that the gap between the reflector and car body is not an error, but modeled this way. However, white pixels shown in the magnified region around and on the bumper are indeed errors, resulting from the conversion of NURBS surfaces into bicubic Bézier patches

8 Conclusions & Future Work

Though most ray tracing systems use only triangles as their geometric primitive, recently it has been shown that it is feasible to ray trace Bézier surfaces at interactive frame rates on a single commodity PC.

System	Our approach	Maya	Rhino 3D
Load File	n.a.	4.12 min	3 min
Preprocessing	0.87 min	n.a.	21.75 min
Rendertime/frame	0.54 s	20 s	71 s
Memory usage	< 50 MB	> 1.0 GB	> 1.2 GB

Table 2: All three systems load and render the VW Polo scene once. Note that the time taken for loading the file is contained in the preprocessing time, while Maya does preprocessing, i.e., low detail triangulation in this case, on the fly while loading. All three resulting images were comparable in their visual quality. The Rhino 3D timings were measured on a Pentium IV 2.8 GHz with 1 GB of RAM running Windows XP Professional. Since all three approaches are using ray tracing for rendering the graphic cards are not important here.

In this paper, we have presented a novel approach to NURBS surface evaluation, which in conjunction with the interactive ray tracing system developed by Geimer [7], allows for ray tracing of arbitrary NURBS surfaces. Furthermore, the performance of our proposed algorithm is largely independent of the number of control points and is numerically stable for any degree and number of control points. Preprocessing time is short even for complex models, typically not exceeding one minute. An optimized bounding volume hierarchy provides good initial estimates for the Newton iteration, which is the foundation of the intersection test. In addition, we have shown that memory consumption is very low and can compete well with triangle based models, especially if high detail is desired.

The most important factor for the high performance is the extremely fast NURBS surface evaluation, which of course could also be useful in other contexts, like collision detection using NURBS or precise ray picking on such models.

In the context of this paper, we have only considered interactive walk-through scenarios. However, it is basically also possible to introduce dynamic interaction without too much effort, since known concepts should work seamlessly with our approach as long as no surface is deformed. In this case, some or all *CDBItems* need to be recomputed, however, investigating the usability of deformable NURBS surfaces in a realtime environment could be an interesting subject for further work.

Additionally, the bounding volume hierarchy creation can be improved by generating a box only for regions, which are known to converge based on the approach presented in [24].

If a high frame rate is considered more important than a short preprocessing time, it would also be feasible to integrate the Bézier intersection code described in [8] into our NURBS-based system.

Finally, memory consumption could be reduced even further, as any *CDBItem* set with identical k, and n, and uniform knot vector, will result in exactly the same precomputed c_i values. Very large scenes and/or very homogeneous scenes (only bicubic 4×4 surfaces for example) will benefit most. In these cases, surfaces would be able to use the same item set. This might also increase cache performance as well, depending on the scene.

Acknowledgements

The authors would like to thank all people who contributed to this paper with helpful discussions and comments, especially Stephan Palmer, Rodja Trappe, Thorsten Grosch, and Anneli Lundin. In addition, we would like to thank the Volkswagen AG for providing the Polo data set.

References

[1] O. Abert. Interactive ray tracing of NURBS surfaces by using SIMD instructions and the GPU in parallel. Master's thesis, Universität Koblenz-Landau, Germany, 2005.

[2] C. Benthin, I. Wald, and P. Slusallek. Interactive ray tracing of free-form surfaces. In *Proceedings of ACM AFRIGRAPH*, pages 99–106, Nov. 2004.

[3] S. Campagna, P. Slusallek, and H.-P. Seidel. Ray tracing of spline surfaces. 13(6):265–282, Aug. 1997.

[4] C. de Boor. On calculating with B-Splines. *Journal of Approximation Theory*, 6:50–62, 1972.

[5] A. Efremov. Efficient ray tracing of trimmed NURBS surfaces. Master's thesis, Saarland University, Saarbrücken, Germany, 2004.

[6] A. Efremov, V. Havran, and H.-P. Seidel. Robust and numerically stable Bézier clipping method for ray tracing NURBS surfaces. In *Proceedings of 21st Spring Conference on Computer Graphics*, pages 127–135, May 2005. (Budmerice, Slovakia, May 12–14, 2005).

[7] M. Geimer. *Interaktives Ray Tracing*. PhD thesis, Universität Koblenz-Landau, Germany, 2005.

[8] M. Geimer and O. Abert. Interactive ray tracing of trimmed bicubic Bézier surfaces without triangulation. In *WSCG'2005 Full Papers Conference Proceedings*, pages 71–78, Feb. 2005.

[9] J. Goldsmith and J. Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[10] headus (metamorphosis) Pty Ltd. Killeroo model. http://www.headus.com/au/cinefex/1.html, 2006.

[11] Intel Corp. Hyper-Threading Technology. http://www.intel.com/technology/hyperthread/, 2004.

[12] Intel Corp. *IA-32 Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, 2004.

[13] J. T. Kajiya. Ray tracing parametric surfaces. *Computer Graphics (Proceedings of SIGGRAPH 82)*, 16(3):245–254, July 1982.

[14] L. Kobbelt, K. Daubert, and H.-P. Seidel. Ray tracing of subdivision surfaces. In *Rendering Techniques '98 (Proceedings of the Eurographics Workshop on Rendering)*, pages 69–80, July 1998.

[15] J. Lext and T. Akenine-Möller. Towards rapid reconstruction for animated ray tracing. *Eurographics 2001 Short presentations*, pages 311–318, Sept. 2001.

[16] W. Martin, E. Cohen, R. Fish, and P. Shirley. Practical ray tracing of trimmed NURBS surfaces. *journal of graphics tools*, 5(1):27–52, 2000.

[17] Motorola, Inc. *AltiVec Technology Programming Interface Manual*, 1999.

[18] K. Müller, T. Techmann, and D. Fellner. Adaptive ray tracing of subdivision surfaces. *Computer Graphics Forum (Proceedings of Eurographics 2003)*, 22(3):543–552, Sept. 2003.

[19] M. J. Muuss. Towards real-time ray-tracing of combinatorial solid geometric models. In *Proceedings of BRL-CAD Symposium*, June 1995.

[20] T. Nishita, T. W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):337–345, Aug. 1990.

[21] S. Parker, W. Martin, P.-P. J. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive ray tracing. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 119–126, Apr. 1999.

[22] B. Smits. Efficiency issues for ray tracing. *journal of graphics tools*, 3(2):1–14, 1998.

[23] M. Sweeney and R. Bartels. Ray tracing free-form B-spline surfaces. *IEEE Computer Graphics and Applications*, 6(2):41–49, Feb. 1986.

[24] D. L. Toth. On ray tracing parametric surfaces. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3):171–179, July 1985.

[25] I. Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, Saarbrücken, Germany, 2004.

[26] I. Wald, C. Benthin, and P. Slusallek. Distributed interactive ray tracing of dynamic scenes. In *Proceedings of IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 77–86, Oct. 2003. (Seattle, Washington, October 20–21, 2003).

[27] I. Wald, P. Slusallek, C. Benthin, and M. Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum (Proceedings of Eurographics 2001)*, 20(3):153–164, Sept. 2001.

[28] S.-W. Wang, Z.-C. Shih, and R.-C. Chang. An efficient and stable ray tracing algorithm for parametric surfaces. *Journal of Information Science and Engineering*, 18(4):541–561, July 2001.