GPU-based trimming and tessellation of NURBS and T-Spline surfaces

Michael Guthe[*] Ákos Balázs[†] Reinhard Klein[‡]
Universität Bonn, Institute of Computer Science II

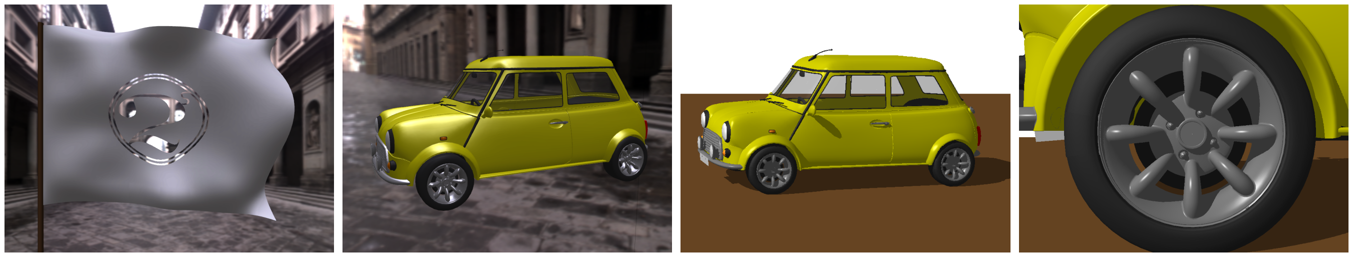

Figure 1: Rendering of NURBS models (from left to right): animated trimmed NURBS surface (degree 5×5, 100 control points) with environment mapping; Mini model consisting of 629 trimmed surfaces; with shadow maps; closeup onto the front wheel.

Abstract

As there is no hardware support neither for rendering trimmed NURBS – the standard surface representation in CAD – nor for T-Spline surfaces the usability of existing rendering APIs like OpenGL, where a run-time tessellation is performed on the CPU, is limited to simple scenes. Due to the irregular mesh data structures required for trimming no algorithms exists that exploit the GPU for tessellation. Therefore, recent approaches perform a pre-tessellation and use level-of-detail techniques. In contrast to a simple API these methods require tedious preparation of the models before rendering and hinder interactive editing. Furthermore, due to the tremendous amount of triangle data smooth zoom-ins from long shot to close-up are not possible. In this paper we show how the trimming region can be defined by a *trim-texture* that is dynamically adapted to the required resolution and allows for an efficient trimming of surfaces on the GPU. Combining this new method with GPU-based tessellation of cubic rational surfaces allows a new rendering algorithm for arbitrary trimmed NURBS and T-Spline surfaces with prescribed error in screen space on the GPU. The performance exceeds current CPU-based techniques by a factor of up to 1000 and makes real-time visualization of real-world trimmed NURBS and T-Spline models possible on consumer-level graphics cards.

CR Categories: I.3.3 [COMPUTER GRAPHICS]: Picture/Image Generation—Display algorithms; I.3.5 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling—Splines

Keywords: GPU-based algorithms, trimming, NURBS and T-Spline surfaces

[*]e-mail:guthe@cs.uni-bonn.de
[†]e-mail:edhellon@cs.uni-bonn.de
[‡]e-mail:rk@cs.uni-bonn.de

1 Introduction

The standard surface representation used in industrial CAD systems are trimmed non-uniform rational B-Spline (NURBS) surfaces.The main advantage of this representation is the ability to compactly describe a surface of almost any shape. Recently, the introduction of T-Splines extended this surface representation further with hierarchical concepts, making it even more attractive for design purposes. Although much effort has been spent to build specialized hardware for rendering these surfaces (e.g. [Abi-Ezzi and Subramanian 1994]), no hardware implementation is available for this purpose so far. The main reason for this are the irregular mesh data structures required to generate a trimmed mesh (see Figure 2). As special hardware is not available all current algorithms tessellate the surfaces, i.e. approximate them by triangle meshes on the CPU.

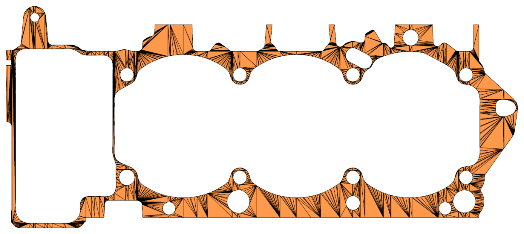

Figure 2: Tessellation of a planar surface with complex trimming (20 trimming loops) by explicit meshing.

Since high-quality tessellation of real world models is not possible in real-time using existing algorithms, it is performed in a preprocessing step in most systems. However, this preprocessing is a cumbersome task especially in industrial applications. In addition, due to the wide range of resolutions from cm to μm (see Figure 3) needed to allow both long shots and close-ups the tessellated models can become extremely complex at the finest resolution and contain billions of triangles. This requires out-of-core algorithms which all have a high latency and thus current systems restrict themselves to a maximum available accuracy preventing close-ups in turn. Recent approaches try to alleviate this restriction by tessellating patches on-the-fly if during a close-up a higher accuracy is needed. Due to the limited tessellation performance this increases the already high latency and inaccuracies of these approaches.

Looking at the CPU-based trimmed NURBS and T-Spline tessellation algorithms we identified memory access as a second limit-

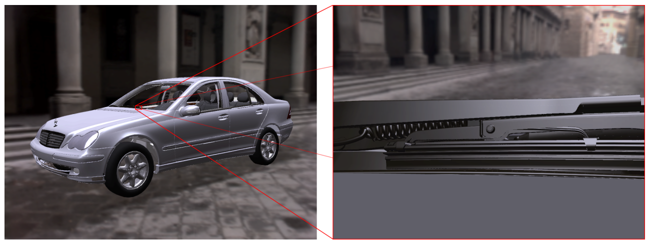

Figure 3: Range of resolutions from cm to μm.

ing factor besides the irregular data structures needed for trimming. This can be seen by the following example: consider an effective memory bandwidth of e.g. 2.08 GB/s for 200 MHz DDR RAM (PC3200), at most 83.2 MB are available per frame for real-time rendering. Since irregular meshes are generated, each triangle requires at least 24 bytes which need to be written to memory and then read again for rendering. Thus, at most 1.7 million triangles can theoretically be generated per frame. Due to additional memory access required for evaluation (e.g. 256 bytes per vertex for a bi-cubic patch), this is reduced to about 250 thousand triangles per frame. Trimming and triangulation of the irregular mesh reduces this performance further, typically by a factor of 10 to 20. Since at least several hundred thousands of triangles are necessary to render e.g. a car-model with acceptable quality, the corresponding tessellations cannot be generated in real-time. On a graphics card, the memory bandwidth is significantly higher due to the parallel architecture, and thus e.g. up to 20 million triangles are possible at real-time frame rates on a GeForce 5900 Ultra. If the tessellation could be performed on the GPU, the control points would be cached and thus the shader runtime instead of the bandwidth would be the limiting factor. Furthermore, if a regular grid would be used, only a single pass would be required for tessellation. In total, a tessellation based on a regular grid running on the GPU should be 60 to 1000 times faster than any algorithm running on the CPU with the above architecture. Therefore, a tessellation algorithm running completely on the GPU would provide a new API for direct NURBS and T-Spline rendering that overcomes all of these limitations.

The key to any GPU-based tessellation algorithm is an efficient solution for the trimming. However, existing trimming approaches cannot be implemented on the GPU due to the already mentioned irregular mesh data structures. In this paper we present a novel texture based trimming approach that does not perform any explicit meshing and only requires regular data structures. Therefore, it can efficiently be mapped to hardware and realized on current GPUs. We show that, based on this new algorithm, GPU-based rendering of trimmed NURBS and T-Spline surfaces becomes possible resulting in an up to 1000 times higher tessellation performance than on the CPU. By using a view-dependent resolution for the tessellation which guarantees a lower screen space error than a user specified threshold, runtime tessellation and high quality rendering of trimmed NURBS models are achievable. Finally, we show that our algorithm can seamlessly be integrated into the rendering pipeline and current graphics APIs e.g. OpenGL.

2 Related Work

As our new method exploits ideas of CPU based trimmed NURBS tessellation and evaluation on GPUs, we give a short overview of both fields. Since the restrictions of current GPUs make it impossible to efficiently evaluate higher order surfaces, we use degree reduction and consequently also review prior work in this field.

Tessellation: In the last two decades different approaches have been developed for the rendering of trimmed NURBS surfaces, for example ray-tracing (e.g. [Nishita et al. 1990]) or pixel-level subdivision (e.g. [Shantz and Chang 1988]). As direct trimmed NURBS rendering is not supported in hardware, the tessellation of the surfaces has become the most widely-used technique for interactive applications, since the resulting polygonal representation can be rendered efficiently. The basic approach used by these polygonal tessellation methods is to first approximate the surface itself with a mesh. For this approximation, typically a regular quad grid or an octree is used. Then the trimming curves are approximated as well and the intersections between those and the surface approximation are calculated. While early approaches (e.g. [Herzen and Barr 1987; Rockwood et al. 1989; Forsey and Klassen 1990]) dealt with individual curves or surfaces only, more recent approaches try to handle multiple patches as in [Kumar et al. 1997], where surfaces are stitched based on a priori known connectivity information. If a common representation of the trimming curves on both sides of adjacent patches is given, an individual view-dependent triangulation can be generated at run-time using the same sampling frequency on both patches to avoid cracks as in [Chhugani and Kumar 2001]. If no common curve representation is available, it can be generated and stored as in [Guthe et al. 2002]. However, these algorithms are restricted to the rendering of static models. For dynamic models, Balázs et al. [2004] developed an algorithm which independently tessellates the trimmed NURBS surfaces and visually closes the cracks on the GPU. However, due to the low number of surfaces that can be tessellated per second, all CPU based runtime tessellation algorithms need to distribute the re-tessellation among several frames which leads to popping artifacts during movement.

Evaluation on GPUs: Abi-Ezzi and Subramanian [1994] and Bóo et al. [2001] proposed an additional adaptive tessellation unit at the front of the rendering pipeline for NURBS and subdivision surfaces respectively. Bolz and Schröder [2003] developed an algorithm to evaluate Catmull-Clark subdivision surfaces on programmable graphics hardware. After the transmission of the tessellation textures to the GPU, only control points instead of triangles need to be send and thus the fragment shader can be saturated with marginal bus bandwidth consumption. With different tessellation textures this approach can also be used for bi-cubic B-Spline surfaces since they are equivalent to this subdivision scheme on a regular quad mesh. The algorithm generates an adaptive tessellation on a per-patch basis, which is rendered into an offscreen buffer – a so called pixel buffer or p-buffer – and then used as input for a second rendering pass. Theoretically this method could achieve up to 24 million vertices per second on recent GPUs, but the high number of p-buffer switches – one per patch is required – reduces the performance by several orders of magnitude. Based on this work, Kanai and Yasui [2004] developed an algorithm to calculate accurate per-pixel normals on a tessellated subdivision surface. Although the produced images are very convincing, it is too slow for real time rendering of more than about ten surfaces.

Degree Reduction: The idea of approximating high degree Bézier curves using degree reduction already came up more than 30 years ago [Forrest 1972]. As shown by Park and Choi [1995], the error can be reduced greatly by subdividing the curve before degree reduction. They also discovered error bounds on the degree reduced curve, which are used in this work. Using a standard degree reduction algorithm however, the degree of continuity between the composite curves cannot be controlled directly. Either, the continuity is preserved up to the maximum for the current curve degree (e.g. [Forrest 1972]), or completely lost (e.g. [Eck 1993]). Therefore, Zheng and Wang [2003] developed a method to explicitly control the continuity classes of the curve at its endpoints. However, all these algorithms preserve the parametric continuity which is not necessary when only dealing with shape e.g. for surface design and rendering. In this case, geometric continuity – a curve is *n*th-order geometrically continuous if it is *n* times differentiable with respect to arc length – is more appropriate.

3 Algorithm Overview

Rendering of trimmed patches consists of two major parts. On one hand the patch is mapped from its rectangular domain $\Omega \subset \mathbb{R}^2$ into $\mathbb{R}^3$ by evaluating it at sample vertices in the domain. The sampling has to be dense enough to guarantee an appropriate piecewise linear or bilinear approximation of the 3d patch. On the other hand, parts of the patch outside the trimming region, which is defined by closed parametric curves in the parameter domain, are cut away. While state-of-the-art evaluation algorithms are well suited for implementation on the GPU, current trimming methods cannot be transferred to highly parallel architectures like the GPU due to the explicit meshing. Our novel approach to solve this trimming problem is a GPU-based algorithm that allows to represent the trimming region by an appropriate black and white texture of sufficient resolution. For each texel the color determines if it is inside or outside the trimmed region. Therefore, a one-bit masking texture can be used for this purpose. While trimming by the use of textures is a known technique, the challenge is to find a parallelizable algorithm for the generation of this binary trim-texture that can be implemented on the GPU and such an algorithm is not available up to date. Having such a parallelizable algorithm, the following two questions must still be answered: first, how to choose the resolution of the trim-texture to guarantee a prescribed error (Section 5.3) and second, how to choose the sampling rates of the trimming curves and surfaces (Section 5.1 and 5.2 respectively).

The overall workflow is shown in Figure 4. First, the trimming curves are sampled with sufficient accuracy and evaluated on the GPU (1). Then the resulting polygons are rendered into a texture of appropriate size using the p-buffer extension (2). In the second rendering pass, the patch is sampled using a regular grid of sufficient resolution. The resolution is chosen in a way that a given screen space error is guaranteed. Since generating an appropriate grid on the CPU for each patch is contradictionary to a GPU-based approach, predefined grids of different resolutions are stored on the graphics card in advance. At runtime only the grid index is calculated on the CPU and then sent to the GPU. For rendering of the patch, it is evaluated at all grid vertices on the GPU (5). For the trimming, we simply bind the trim-texture and all pixels outside the trimming region are removed in the fragment stage by a lookup into this trim-texture (6).

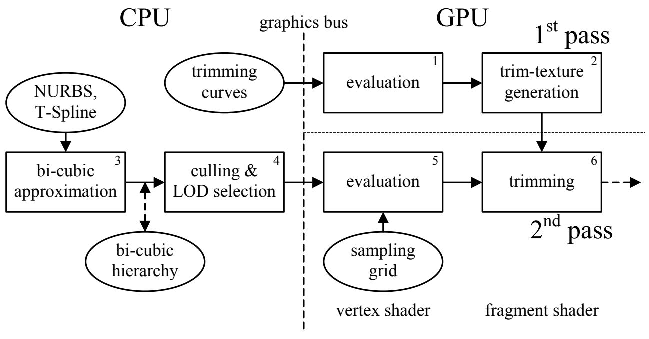

Figure 4: Main workflow of our algorithm.

When developing an algorithm for the GPU numerous restrictions have to be taken into account. Due to the highly parallel architecture global hierarchies or irregular data structures (e.g. for stitching) cannot be used. Instead, each surface needs to be treated individually. As data dependent loops are only supported by very recent GPUs, a conversion from NURBS or T-Spline to piecewise rational Bézier representation is necessary, since the current knot spans needed to calculate the sample points differ. Furthermore, for cards not having texture access in the vertex shader, the amount of input data for a vertex program is limited to 16 vertex attributes and 8 program matrices and thus only low degree Bézier patches can be evaluated. Since we want our algorithm to work with any graphics card supporting at least vertex shader 1.0, we restrict ourselves to this extension that only supports 12 temporary registers and thus limit the maximum degree to bi-cubic. Thus, our overall algorithm first approximates each NURBS or T-Spline surface and its trimming curves with a coarse hierarchy of rational bi-cubic Bézier patches, or cubic rational Bézier curves respectively, on the CPU (3). During rendering this hierarchy is traversed and patches with sufficient accuracy are selected to guarantee a given screen space error (4). If the traversal reaches a leaf node, additional bi-cubic patches are generated. Then the control points of each patch are sent to the GPU before selecting a grid of appropriate resolution for evaluation.

4 Trimming on the GPU

After converting the approximating cubic trimming curves into a suitable polygonal representation (see Section 4.1) the trim-texture is generated from these polygons with holes, by an algorithm similar to the one used for the area calculation of polygons. The main idea is, that when spanning a triangle fan from the first vertex of each trimming loop, a point inside the trimming region will be covered an odd number of times by the triangles of these fans, while a point outside the trimming region will be covered by an even number of times as shown in Figure 5. Instead of counting the coverages, it is possible to simply consider the lowest bit and toggle between black and white. A major advantage of this approach is that we do not need to take care of the orientation and nesting of trimming loops and thus error prone special case handling is avoided.

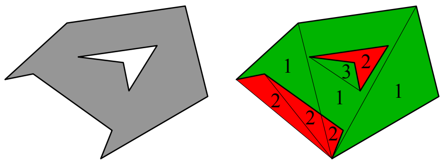

Figure 5: Left: Concave polygon with hole. Right: Texel coverage using our algorithm (green regions are inside and red outside).

4.1 Trimming Curve Conversion

The generation of the trim-texture is illustrated in Figure 6. For each trimming loop a triangle fan is generated. The vertices $C_k(t_i)$ at the parameter values t_i of this triangle fan are calculated using the control points of the corresponding curve segment C_k. This is done by initializing the vertex attributes with the control points P_j and then sending the sampling parameter values $t_1, \ldots, t_n$ as 1d vertices. These values are used by a vertex program which takes the control points as constants and evaluates the corresponding curve at the parameter values t_i. This way the vertices of the triangle fan are generated curve by curve and the resulting triangles are rasterized. The toggling of the pixels is performed in the blending stage of the rendering pipeline. It is important to note, that this way the entire trim-texture generation is performed in a single rendering pass.

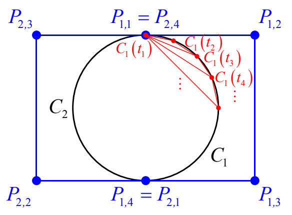

Figure 6: Trim-texture generation.

Similarly to the bi-cubic approximation of the surfaces we approximate the trimming curves with rational cubic Bézier curves. For evaluation we chose the deCasteljau algorithm since it only requires 12 assembly operations while the direct evaluation needs 13.

4.2 Surface Evaluation

In principle previous adaptive GPU based tessellation algorithms developed for subdivision surfaces could be adopted to tessellate the rational bi-cubic Bézier patches. However, these algorithms have the already mentioned drawback that for each patch a p-buffer switch is required as the tessellation is performed in a fragment shader which makes them useless in practical applications. This switch can only be removed if the connectivity is already defined before rendering, since then only the evaluation on the GPU is required which can already be performed in the vertex shader. Therefore, we store predefined grids of different resolutions on the graphics card. In order to span a wide spectrum of grid resolutions we start with different types of simple base grids that are subdivided in either of the two parameter directions as required, yielding four hierarchies up to a maximum resolution depending on the maximum screen resolution. To achieve a target resolution of e.g. 17×350 we would use the 3×3 base grid and subdivide it three times in x-direction and seven times in y-direction leading to a resolution of $(3 \cdot 2^3) \times (3 \cdot 2^7) = 24 \times 384$.

From the many different algorithms for evaluating Bézier patches, we have to choose the one that can be implemented most efficiently on current GPUs. First of all, the power basis form is not reasonable since its numerical instability is even more severe on low accuracy GPUs. This leaves the choice between the deCasteljau algorithm and direct evaluation. For rendering, the vertex normal required for shading needs to be calculated in addition to the vertex position. The deCasteljau algorithm needs a total of 74 assembly operations, while the direct evaluation only requires 61 operations. Additionally direct evaluation only requires 12 temporary registers while the deCasteljau algorithm needs 17.

4.3 Rendering

After the trim-texture is constructed, it is bound and the trimming is performed in the fragment shader. When the patch is rendered, simply all fragments are killed for which the intensity of the trim-texture is lower than a threshold value. If fragment shaders are not supported, the trim-texture is used as alpha texture with an alpha test. Although using a p-buffer in combination with the render target extension for the trim-texture is the fastest possibility, the render target has to be changed – a so called p-buffer switch occurs – twice for each patch. Since this requires a complete state reload and is therefore a very expensive operation, we focus on reducing the number of such render target changes as much as possible.

4.4 Multiple Trimmed Patches

In order to reduce the number of p-buffer switches a trim-texture atlas is generated for multiple patches, which contains all trim-textures of these patches. When several trimmed patches are rendered at once, first the required sizes of all stencil textures of the corresponding patches are calculated and sorted by their height. Then the rectangular trim-textures are placed beside each other at the bottom line of the atlas. When the next texture would exceed the maximum texture width, a new row is started. Although this algorithm is very simple it is sufficiently efficient for the rectangular textures used here. After the texture atlas is filled (i.e. adding the next trim-texture would exceed the maximum texture height) or all trim-textures have been added, the trim-textures are rendered into the atlas. For each patch only the viewport needs to be set according to the position and resolution of its trim-texture. When all trim-textures are generated, the algorithm switches back to the screen buffer and renders all patches for which the trimming is contained in the current texture atlas. Note that untrimmed patches can immediately be rendered before generating the first trimming atlas. If the texture atlas was filled before all trim-textures could be added, the algorithm continues with the next texture atlas.

In industrial models trimming is often used to cut out small parts of large surfaces. This means that after the conversion of the NURBS or T-Spline surface to rational Bézier patches, some of these patches lie completely outside the trimming region and only a small region of the trim-texture is used at all. Therefore, we calculate the bounding box of the trimming region and apply knot insertion at the minimum and maximum u and v parameter values. Finally, we remove all Bézier patches outside this region. If the trimming is only used to cut out a rectangular region of the surface domain, no trimming is necessary at all after removing the unused domain regions. This is the case, if only a single loop exists and each trimming curve lies completely on one side of the domain boundary. Then the patch can be rendered without a trim-texture.

5 Sampling

As we approximate all curves with piecewise rational cubic curves and all surfaces with rational bi-cubic patches, we limit our discussion to this type of curves and surfaces, but a generalization to higher degree curves and patches is possible. For the rendering of these cubic curves or bi-cubic surfaces, the required sampling resolution to guarantee a specified error ε needs to be calculated.

5.1 Trimming Curves

According to Filip et al. [1986], the number of required samples n for a piecewise linear approximation of a function $f(t)$ over the interval $[a,b]$ (which is always $[0,1]$ for Bézier curves) with a maximum deviation of ε can be calculated by:

$$n = \left\lceil (b-a)\sqrt{\frac{\sup_{a \le t \le b} \|f''(t)\|}{8\varepsilon}} \right\rceil$$

A rational bi-cubic Bézier curve projected to the hyperplane $w = 1$ can be written as

$$C(t) = \frac{\sum_{i=0}^{3} P_i B_i^3(t)}{\sum_{i=0}^{3} w_i B_i^3(t)} = \frac{P(t)}{w(t)},$$

and its second derivative can be written as a rational Bézier curve with a degree seven nominator $\check{P}(t) = \sum_{i=0}^{7} \check{P}_i B_i^7(t)$ and a degree nine denominator $\check{w}(t) = \sum_{i=0}^{9} \check{w}_i B_i^9(t)$. Since all w_i are positive by construction, all $\check{w}_i$ are also positive. Therefore, an upper bound of the second derivative is given by:

$$\sup_{0 \le t \le 1} \|C''(t)\| \le \frac{\max(\|\check{P}_0\|, \ldots, \|\check{P}_7\|)}{\min(\check{w}_0, \ldots, \check{w}_9)}$$

5.2 Surfaces

To generate less rendering primitives (e.g. for cylindrical surfaces), the sampling resolution in u- and v-direction is calculated independently. According to Filip et al. [1986], the error when approximating a C^2-continuous surface with two triangles spanning the bilinear parameter space rectangle $D = [(u_0, v_0), (u_1, v_1)]$ is bounded by

$$\sup_{p \in D} \|f(p) - l(p)\| \le \frac{1}{8}(\Delta u^2 M_u + 2\Delta u \Delta v M_{uv} + \Delta v^2 M_v),$$

with

$$M_u = \sup_{p \in D} \left\| \frac{\partial^2 S}{\partial u^2} \right\|, \; M_{uv} = \sup_{p \in D} \left\| \frac{\partial^2 S}{\partial u \partial v} \right\|, \text{ and } M_v = \sup_{p \in D} \left\| \frac{\partial^2 S}{\partial v^2} \right\|$$

Now we can separate the sampling densities by exploiting the fact that $ab \le \frac{1}{2}(a^2 + b^2)$ and thus the approximation error is bound by

$$\sup_{p \in D} \|f(p) - l(p)\| \le \frac{1}{8} \left(\Delta u^2 (M_u + M_{uv}) + \Delta v^2 (M_v + M_{uv}) \right),$$

which is a simple addition the two approximation errors in u- and v-directions. Thus ε is an upper bound for the approximation error if the error in both directions is not greater than $\frac{\varepsilon}{2}$.

When a patch has no trimming and the parameter value is not used for texturing, it is not necessary to preserve its parametrization. In this case an upper bound for the distance of a curve to an evenly parameterized line segment is not required. Instead, any reparametrization of this degree elevated line segment can be used. This means that the middle control points can freely move between the two end points of the line segment. Therefore, the closest point on the line segment is calculated for each control point of the curve. These points then define a re-parameterized line segment and the difference vectors to the control points define the difference curve. Using the maximum second derivative of this difference curve the required sampling resolution for a purely geometric approximation can be calculated which is lower.

5.3 Trim-Texture

For trimmed NURBS or T-Spline surfaces the required trim-texture resolution has to be calculated additionally to the grid resolution. To guarantee a desired error of ε along the trimming curves, both the surface and the trimming curves need to be approximated with an accuracy of $\frac{\varepsilon}{2}$. Therefore, the distance between two neighboring pixels of the texture has to be at most ε on the evaluated surface. Thus the texture resolution can be calculated from the maximum absolute value of the first surface derivatives:

$$res_u = \left\lceil \frac{(u_1 - u_0)}{\varepsilon} \sup_{p \in [(0,0),(1,1)]} \left\| \frac{\partial S(p)}{\partial u} \right\| \right\rceil$$

and analogously for the v-resolution, where $[(u_0, v_0), (u_1, v_1)]$ is the domain interval of the current bi-cubic Bézier patch.

Again, the maximum surface derivative in u- or v-direction is bound by the maximum derivative of the corresponding iso-parameter curves, which can be written in rational Bézier form with degree five nominator and degree six denominator. An upper bound for the absolute value is:

$$\sup_{0 \le t \le 1} \|C'(t)\| \le \frac{\max(\|\check{P}_0\|, \ldots, \|\check{P}_5\|)}{\min(\check{w}_0, \ldots, \check{w}_6)}$$

A problem occurs, when the viewer moves very close to a surface. In this case the patch size becomes much larger than the screen and therefore, the required trim-texture resolution would exceed the maximum possible texture resolution by orders of magnitude. To overcome this limitation, the traversal of the bi-cubic patch hierarchy is continued until the trim-texture of each patch is small enough. For this it is only necessary to modify the bi-cubic approximation error. Note, that for trimmed surfaces only a trim-texture for the domain region covered by visible patches needs to be generated. This optimization does not only increase the rendering performance but also the accuracy of trimming.

6 Bi-cubic Approximation

In order to approximate a given NURBS or T-Spline surface with rational bi-cubic Bézier patches we first convert the surface into its piecewise rational Bézier representation. For NURBS surfaces the Oslo algorithm [Cohen et al. 1980] is used and for the recently developed T-Spline surfaces the knot insertion algorithm of Sederberg et al. [2004] is applied. Afterwards each of these initial Bézier patches which can be of arbitrary degree is approximated with a bi-cubic patch as described in Section 6.1. Since the error of this approximation may exceed a desired error bound, we build a binary hierarchy of bi-cubic patches during rendering by recursive subdivision of the initial Bézier patches (blue subtrees in Figure 7). To reduce the number of rendered bi-cubic patches these separate hierarchies are also combined into a single binary tree (shown in green in Figure 7) using the median cut algorithm [Heckbert 1982]. After the tree is built, we hierarchically simplify the bi-cubic patches – approximate the two child patches with a single bi-cubic patch – starting from the level of the initial Bézier patches. This simplification process is performed once when the surface is rendered for the first time. A detailed description is given in Section 6.2.

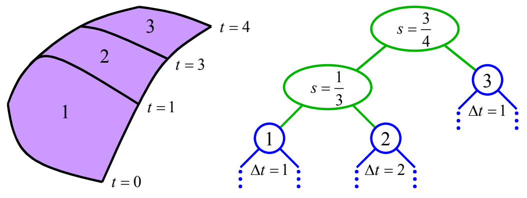

Figure 7: NURBS surface with its bi-cubic patch hierarchy.

6.1 Approximation of a Single Bézier Patch

To find a sensible bi-cubic approximation of a single rational Bézier patch contained in a leaf node we use a novel constrained degree reduction. We derive our approximation algorithm completely from a generalized degree reduction. Therefore, we can simply apply it to rational curves by using the homogeneous representation of the control points $P_i = [\, w_i x_i \;\; w_i y_i \;\; w_i z_i \;\; w_i \,]^T$.

As Bézier surfaces are tensor product surfaces, degree reduction of the surface in one direction is equal to degree reduction of all curves in this direction. Previous degree reduction algorithms like [Forrest 1972] are equivalent to Hermite interpolation in the cubic case and thus preserve C^1 continuity. However, in the context of rendering, preserving G^1-continuity would be sufficient since only the direction of the tangent vector needs to be preserved. This leads to the following definition of the new control points:

$$\tilde{P}_0 = P_0 \qquad \tilde{P}_1 = P_0 + \lambda_0(P_1 - P_0)$$
$$\tilde{P}_2 = P_n + \lambda_1(P_{n-1} - P_n) \qquad \tilde{P}_3 = P_n$$

The two free parameters λ_0 and λ_1 can now be used to minimize the total approximation error. The distance between two Bézier curves $C_1(t)$ and $C_2(t)$ of the same degree is bound by the maximum distance between their corresponding control points. Therefore, we elevate the degree of the approximating curve $\tilde{C}(t)$ to that of the original curve $C(t)$ to construct $\bar{C}(t)$ and then minimize the control point distances:

$$\sum_{i=0}^{n} \|P_i - \bar{P}_i\|^2 \rightarrow \min$$

For this minimization problem the analytical solution is:

$$\lambda_0 = \frac{(\sum_{i=0}^m a_i b_i)(\sum_{i=0}^m c_i^2) - (\sum_{i=0}^m a_i c_i)(\sum_{i=0}^m b_i c_i)}{(\sum_{i=0}^m b_i^2)(\sum_{i=0}^m c_i^2) - (\sum_{i=0}^m b_i c_i)^2}$$
$$\lambda_1 = \frac{(\sum_{i=0}^m a_i c_i)(\sum_{i=0}^m b_i^2) - (\sum_{i=0}^m a_i b_i)(\sum_{i=0}^m b_i c_i)}{(\sum_{i=0}^m b_i^2)(\sum_{i=0}^m c_i^2) - (\sum_{i=0}^m b_i c_i)^2}$$

with

$$\begin{aligned} \vec{a}_i &= P_i - P_0(\gamma_{i,0} + \gamma_{i,1}) - P_n(\gamma_{i,2} + \gamma_{i,3}) \\ \vec{b}_i &= -(P_1 - P_0)\gamma_{i,1} \\ \vec{c}_i &= -(P_{n-1} - P_n)\gamma_{i,2}, \end{aligned}$$

where $\gamma_{i,j}$ is the contribution of the simplified control point $\tilde{P}_j$ to the control point $\bar{P}_i$ after degree elevation (i.e. they represent the degree elevation matrix). As the direction of the tangent vector flips when λ_0 or λ_1 becomes negative, a minimum value is used for each of them. In addition, when $w_1 < w_0$ or $w_{n-1} < w_n$, a maximum value for λ_0 and λ_1 is given by $\frac{w_0}{w_0 - w_1}$ and $\frac{w_n}{w_n - w_{n-1}}$ respectively, as only positive weights $\tilde{w}_1$ and $\tilde{w}_2$ should be produced.

After constructing the degree reduced curve an upper bound for the introduced error needs to be calculated. For this purpose we use the non-homogeneous representation of the control points P_i and $\bar{P}_i$ as we are interested in the error after projection. The approximation error ε_c is then:

$$\varepsilon_c = \max_{i=0}^{n} \left\| [\, x_i\ y_i\ z_i \,]^T - [\, \bar{x}_i\ \bar{y}_i\ \bar{z}_i \,]^T \right\|$$

As the approximation error ε_c is not known in advance, additional subdivisions are performed to extend the hierarchy until the approximation error is low enough for the current screen space error.

6.2 Simplification of Two Bi-cubic Patches

To fill the upper part of the bi-cubic Bézier hierarchy described above we perform pairwise approximation of two bi-cubic Bézier patches by a single bi-cubic patch. Similarly to the approximation of a single Bézier patch we derive this simplification from subdivision and thus rational patches are accounted for by using the homogeneous control points. Since the simplification of two Bézier patches into a single one can be viewed as the inverse of subdivision we are able to calculate λ_0 and λ_1 by considering a subdivision of the simplified patch at the parameter value s. As this subdivision has to preserve the knot intervals of the two child patches the parameter s is given by

$$s = \frac{\Delta t_1}{\Delta t_1 + \Delta t_2},$$

where Δt_1 and Δt_2 are the lengths of the knot intervals of the two child patches in the partition direction (see Figure 7). Now we can set up the same minimization problem as for the approximation of a single rational Bézier patch. The $\gamma_{i,j}$ are then defined by the subdivision matrix of s instead of the degree elevation matrix. Finally, an upper bound of the error introduced by simplification is calculated by subdividing the simplified patch at s and then exploiting the convex hull property of the difference curves.

7 Rendering

While the evaluation and rendering of the rational bi-cubic patches are performed completely on the GPU, the selection of sufficiently accurate rational bi-cubic Bézier patches is done on the CPU by traversing the hierarchy associated with the surface starting at the root node. When a patch with sufficient accuracy is found, it is rendered and the rest of the subtree is skipped, similar to standard HLOD algorithms. During the traversal hierarchical view-frustum culling based on the bounding box of the current Bézier patch is also performed. If the patch is visible, the required object space error ε to guarantee a screen space error of ε_{img} is calculated using the distance of the viewer to this bounding box. This object space error is then split equally between the bi-cubic approximation error and the sampling error (see Sections 5 and 6). To increase the performance for very small surfaces, we also check if ε is larger than the bounding box diagonal of the surface. In this case a simple (untrimmed) quad is sent to the GPU instead of the possibly trimmed surface.

7.1 Crack Filling

When selecting the bi-cubic patches for rendering neighboring patches are subdivided independently which can introduce cracks between bi-cubic patches. Trimmed NURBS and T-Spline surfaces are also tessellated individually, which may introduce cracks between neighboring surfaces as well. Both types of cracks need to be closed. To achieve this, we build upon the fat border algorithm [Balázs et al. 2004] which conceals the cracks by rendering appropriately shaded triangle strips behind each trimming loop. Although this seems to be a simple solution it has the drawback that it requires tangent vectors along the trimming curves which need to be calculated in the vertex stage. Since the vertex shader would need more than the 12 temporary registers available in the vertex shader 1.0 extension to calculate the position, normal and tangent vectors for a point on the trimming curve, we cannot use this approach directly.

Cracks between Bézier patches: To fill the cracks between the bi-cubic patches, a simple line strip is rendered around each sampling grid resulting in only a slightly lower quality than the original approach. For a screen space error of ε_{img} the width of this line strip is $2\varepsilon_{img}$, e.g. one pixel for a screen space error of half a pixel.

Cracks between trimmed NURBS patches: For untrimmed NURBS patches the cracks along their boundary do not need to be filled explicitly, since they are already filled by the Bézier patch crack filling algorithm. Therefore, only the cracks along trimming curves need to be filled. One possibility would be to render a line strip along each trimming loop. For this however, the trimming curves would need to be restricted to the currently rendered bi-cubic patches which would increase the CPU computation time significantly. Therefore, we chose a slightly different approach which fills the cracks already when generating the trim-texture. After generating the trim-texture with the algorithm described above, an additional line strip is rendered along each trimming loop. The width of this line strip is always one pixel since the accuracy of the trimming curves in texture space is 0.5 pixel.

8 OpenGL API Integration

We propose the API shown in Figure 8 that is even simpler to use than the original OpenGL NURBS rendering API.

```
int gluGenNurbsObjectsEXT(int count);
void gluControlPoints4fvEXT(int object, int usize, int vsize, float *cp);
void gluKnotVectorUfvEXT(int object, int size, float *knots);
void gluKnotVectorVfvEXT(int object, int size, float *knots);
int gluAddTrimmingLoopEXT(int object);
void gluAddTrimmingCurve3fvEXT(int object, int loop, int size, float *cp,
                               int knotsize, float *knots);
void gluDrawNurbsObjectEXT(int object, float error);
void gluDrawNurbsObjectsEXT(int first, int count, float error);
void gluDrawNurbsObjectsivEXT(int *objects, int count, float error);
void gluNurbsSpecialProgram(int specialprogram);
const char* gluGetNurbsEvaluateShader();
const char* gluGetNurbsTrimmingShader();
```

Figure 8: Proposed API calls (for trimmed NURBS only).

One of the major advantages of our algorithm is the seamless integration into the rendering pipeline. When using the fixed function pipeline, the integration is simple since the vertex and fragment shader can emulate it after tessellation and trimming. To provide a simple mechanism for combining trimmed NURBS and T-Spline rendering with custom shaders using the GL shading language, we give the programmer access to the Bézier evaluation vertex program function and the trimming fragment program function. Then any shader can be used for trimmed NURBS and T-Spline rendering by simply calling these functions at the beginning of the vertex and fragment program and binding the new shader for our extension. Considering this simple mechanism, an integration into the graphics driver is also possible and has the further advantage that custom evaluation hardware can be used when available. As examples we combined our trimmed NURBS and T-Spline rendering algorithm with high dynamic range environment mapping [Debevec 1998] and light space perspective shadow maps [Wimmer et al. 2004].

9 Results

To evaluate our new algorithm we perform all benchmarks on an Athlon 3000+ with 1.5 GByte memory and a GeForce 5900 Ultra at a resolution of 1280×1024 with 0.5 pixel screen space error.

First, we compare the tessellation performance of the GPU-based method to the current OpenGL API using a single bi-cubic trimmed and untrimmed patch (see Figure 9). To investigate the tessellation performance we render these patches at different screen-sizes where a larger screen-size implies a higher sampling rate. For a pixel sized patch all algorithms simply render a quad resulting in the same performance of about 0.01ms mainly due to the pipeline flush. As shown by these results we get a performance gain of a factor of about 1000 for bi-cubic patches across all other sampling resolutions which is even higher than our estimates based on the memory bandwidth. The additional 1ms required by our algorithm for the rendering of trimmed patches is mainly due to the p-buffer switch.

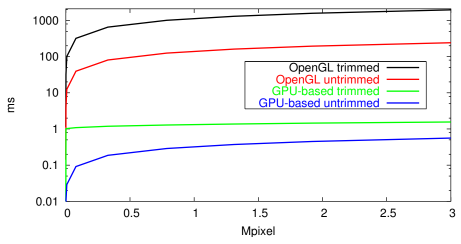

Figure 9: Tessellation performance in dependance of screen size for OpenGL and the GPU-based algorithm. Note the logaritmic scale.

As second example we evaluate the performance of the bi-cubic approximation for surfaces of different degrees. In Figure 10 we show the performance for a single animated trimmed NURBS surface with 100 control points and degrees of 3×3, 5×5, and 7×7 respectively. The reason that the 7×7 degree surface renders faster than the one with 5×5 degree is that it consists of 9 Bézier patches while the 5×5 degree surface consists of 25 Bézier patches. About 70% of the additional time that the higher degree surfaces need compared to the bi-cubic one is required for the bi-cubic approximation (dashed lines in Figure 10). The tessellation time is only slightly higher, since they need approximately the same number of quads. This approximation time scales with $O(\sqrt[4]{n})$ (which is proportional to the number of necessary bi-cubic patches) due to the excellent convergence of this bi-cubic approximation. For T-Spline surfaces, the performance is equal to that of the according NURBS surfaces, since the time for conversion into piecewise Bézier representation is neglectable and afterwards they are identical. Note, that if a second rendering pass is required e.g. for the light space perspective shadow map algorithm [Wimmer et al. 2004] (see Figure 1) the approximation time is required only once.

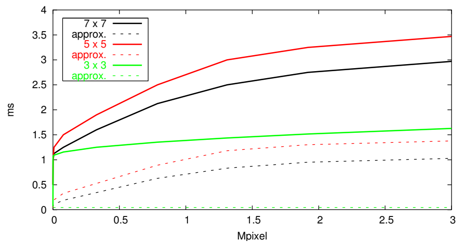

Figure 10: Total rendering performance of a single animated trimmed NURBS surface with 100 control points and of different degrees. Dashed lines show bi-cubic approximation time.

Finally we evaluate the performance when rendering real world models, e.g. the Mini model shown in Figure 1 as well as the Golf and C-Class models shown in Figure 11. Here a fair comparison to existing methods is hardly possible, since they are either too slow, cannot guarantee a certain screen space error and can easily produce a high error especially when zooming in (see Figure 12), or are limited to a certain maximum accuracy in object space due to pre-computations.

Table 1 shows the results of rendering the different car models ranging from 600 to 70,000 surfaces of which 20% to 50% are trimmed. With our proposed new method even complex NURBS models can be rendered interactively. The frame rates for the Golf and the Mini model are similar although the Golf has much more NURBS surfaces, because the bi-cubic representations of both models contain approximately the same number of patches. The reason for

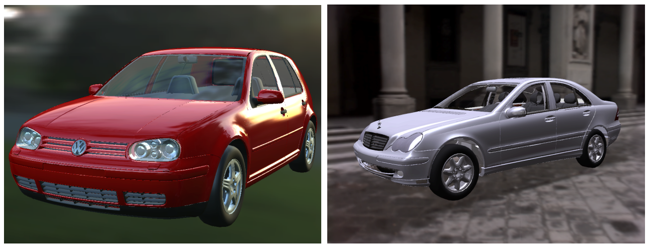

Figure 11: Golf model consisting of 8,138 trimmed surfaces; C-Class model consisting of 67,571 trimmed surfaces.

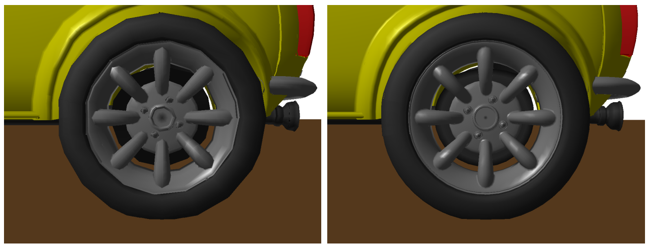

Figure 12: 5 pixel screen space error when latency hiding (left) vs. accurate rendering with our algorithm (right).

	Mini	Golf	C-Class
NURBS surfaces	629	8,138	67,571
non-trivially trimmed	203	1,486	35,230
Bézier patches	25,648	17,936	396,535
our alg.	7 fps	6 fps	1 fps
OpenGL	0.04 fps	0.03 fps	—

Table 1: Details of the static models and the performance of the different rendering algorithms.

the relatively high performance of the C-Class model is the use of the bi-cubic hierarchy described in Section 6 and thus only about 100,000 bi-cubic patches are actually rendered. Note, that the frame rates shown in Table 1 are minimum frame rates and increase when zooming in due to the view-frustum culling.

10 Conclusion

We have presented a novel algorithm to perform trimming of surfaces on the GPU that makes GPU-based tessellation of trimmed NURBS and T-Spline surfaces possible. The advantages of our new approach can be summarized as follows: The tessellation performance is up to 1000 times higher than that of CPU-based tessellation, it can be performed at runtime and thus no preprocessing is required, all resolutions are available, and a simple API can be used. Although upcoming data dependent loops and texture access in the vertex stage will allow evaluation of higher order Bézier surfaces on the GPU, the approximation with rational bi-cubic patches will still be reasonable since it significantly reduces the shader runtime.

11 Acknowledgements

We thank SGI, Volkswagen and DaimlerChrysler for providing us with the trimmed NURBS models, Paul Debevec for the high dynamic range environments, Radomír Měch for his hints on how to reduce p-buffer switches when rendering multiple trimmed surfaces, and Andreas Schilling for his valuable comments.

References

ABI-EZZI, S. S., AND SUBRAMANIAN, S. 1994. Fast dynamic tessellation of trimmed nurbs surfaces. *Computer Graphics Forum 13*, 3, 107–126.

BALÁZS, Á., GUTHE, M., AND KLEIN, R. 2004. Fat borders: Gap filling for efficient view-dependent lod rendering. *Computers & Graphics 28*, 1, 79–86.

BOLZ, J., AND SCHRÖDER, P., 2003. Evaluation of subdivision surfaces on programmable graphics hardware.

BÓO, M., AMOR, M., DOGGETT, M., HIRCHE, J., AND STRASSER, W. 2001. Hardware support for adaptive subdivision surface rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, 33–40.

CHHUGANI, J., AND KUMAR, S. 2001. View-dependent adaptive tessellation of spline surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, 59–62.

COHEN, E., LYCHE, T., AND RIESENFELD, R. F. 1980. Discrete b-spline and subdivision techniques in computer aided geometric design and computer graphics. *Computer Graphics and Image Processing 14*, 2, 87–111.

DEBEVEC, P. 1998. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of ACM SIGGRAPH 98*, ACM Press, 189–198. Computer Graphics Proceedings, Annual Conference Series.

ECK, M. 1993. Degree reduction of bézier curves. *Computer Aided Geometric Design 10*, 3-4, 237–252.

FILIP, D., MAGEDSON, R., AND MARKOT, R. 1986. Surface algorithms using bounds on derivatives. *Computer Aided Geometric Design 3*, 4, 295–311.

FORREST, A. 1972. Interactive interpolation and approximation by bézier polynomials. *The Computer Journal 15*, 1, 71–79.

FORSEY, D. R., AND KLASSEN, R. V. 1990. An adaptive subdivision algorithm for crack prevention in the display of parametric surfaces. In *Graphics Interface '90*, Canadian Information Processing Society, 1–8.

GUTHE, M., MESETH, J., AND KLEIN, R. 2002. Fast and memory efficient view-dependent trimmed nurbs rendering. In *proceedings of Pacific Graphics 2002*, IEEE Computer Society, 204–213.

HECKBERT, P. 1982. Color image quantization for frame buffer display. *Computer Graphics (Proceedings of ACM SIGGRAPH 82) 16*, 3 (July), 297–307.

HERZEN, B. V., AND BARR, A. H. 1987. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 89) 21*, 4 (July), 103–110.

KANAI, T., AND YASUI, Y. 2004. Per-pixel evaluation of parametric surfaces on gpu. In *ACM Workshop on General Purpose Computing Using Graphics Processors (also at SIGGRAPH 2004 poster session)*.

KUMAR, S., MANOCHA, D., ZHANG, H., AND HOFF, K. E. 1997. Accelerated walkthrough of large spline models. In *1997 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 91–102. ISBN 0-89791-884-3.

NISHITA, T., SEDERBERG, T. W., AND KAKIMOTO, M. 1990. Ray tracing trimmed rational surface patches. *Computer Graphics (Proceedings of ACM SIGGRAPH 90) 24*, 4 (August), 337–345. ISBN 0-201-50933-4.

PARK, Y., AND CHOI, U. J. 1995. Degree reduction of bézier curves and its error analysis. *J. Austral. Math. Soc. Ser. B 36*, 399–413.

ROCKWOOD, A. P., HEATON, K., AND DAVIS, T. 1989. Real-time rendering of trimmed surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 89) 23*, 3 (July), 107–116.

SEDERBERG, T. W., CARDON, D. L., FINNIGAN, G. T., NORTH, N. S., ZHENG, J., AND LYCHE, T. 2004. T-spline simplification and local refinement. *ACM Transactions on Graphics 23*, 3, 276–283.

SHANTZ, M., AND CHANG, S.-L. 1988. Rendering trimmed NURBS with adaptive forward differencing. *Computer Graphics (Proceedings of ACM SIGGRAPH 88) 22*, 4 (August), 189–198.

WIMMER, M., SCHERZER, D., AND PURGATHOFER, W. 2004. Light space perspective shadow maps. In *Rendering Techniques 2004 (Proceedings of Eurographics Symposium on Rendering)*, A. Keller and H. W. Jensen, Eds. Eurographics Association, June, 143–152.

ZHENG, J., AND WANG, G. 2003. Perturbing bézier coefficients for best constrained degree reduction in the l_2-norm. *Graphical Models 65*, 351–368.