Vision, Modeling, and Visualization (2021)
B. Andres, M. Campen, and M. Sedlmair (Eds.)

Real-Time Curvature-aware Re-Parametrization and Tessellation of Bézier Surfaces

C. Buchenau and M. Guthe

University of Bayreuth, Visual Computing, Germany

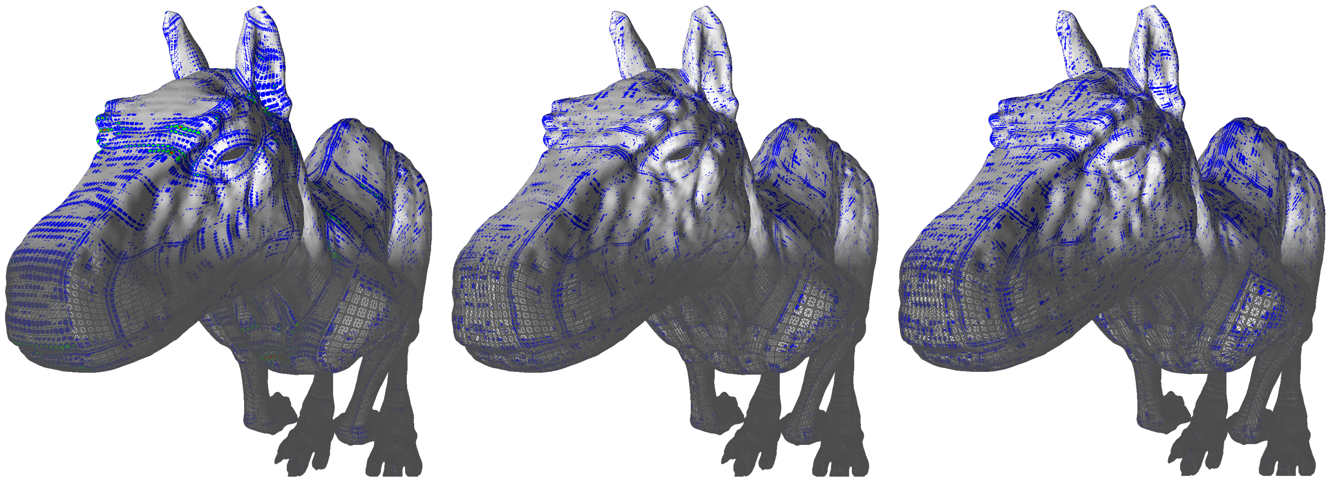

Figure 1: *Visualization of the screen space error and its tessellation for an naive distance-based level-of-detail approach (left), a uniform tessellation based on Filip et al. [FMM86] (middle) and our approach (right).*

Abstract

Interactive tessellation of parametric surfaces has many applications in both engineering and entertainment computing. The most common primitives are bi-cubic Bézier patches which are, among others, an intermediate representation of subdivision surfaces for rendering. The current state-of-the-art employs hardware tessellation where a uniform subdivison pattern is used per patch. If the curvature varies strongly over a patch, this results in an over-tessellation of flat areas.

Based on the observation that the second derivative changes linearly over the patch, we show that it is possible to re-parameterize the patches such that the tessellation adapts to the curvature. This way, we reduce the number of primitives by an average of 15% for the same error bound.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ ***Parametric curve and surface models; Rasterization;***

1. Introduction

In the field of computer-aided Design (CAD), parametric surfaces are the most wide used object representations. While their mathematical properties permits an engineer to meet precise technical specification the visualization or computation requires additional regard. For rendering a common method is to convert the object of interest into a polygonal form (usually triangles or quadrilaterals) and finally render this object representation.

One of the disadvantages of those approaches is the synthesized geometry: On the one hand the patches are partially over-tessellated, which results in an overhead of the memory consumption and higher processing times and on the other hand they are under-tessellated which can result in visual or computational artifacts. Especially, patches of high variance in curvature are affected by those issues.

Currently, the state-of-the art is using the uniform patterns of

DOI: 10.2312/vmv.20211370

the hardware tessellation unit for rendering. When aiming to guarantee a given parametric error in screen space, the approximation error is computed from bounds on the second derivatives. While this is very efficient, using uniform tessellation leads to the aforementioned over-tessellation of flat regions. Our main contributions are

- a re-parameterization of the sampling pattern to account for local curvature and reduce over-tessellation
- and an improved upper bound for the mixed second derivative, further reducing the number of primitives.

In combination, we can significantly reduce the number of primitives while still guaranteeing the same parametric error.

2. Related Work

In order to guarantee a given geometric or screen space error when tessellating curved surfaces, different upper bounds for piecewise linear approximations have been proposed. Based on the convex hull property of Bézier curves and surfaces, Filip et al. [FMM86] derived upper bounds computed from second derivatives of the surface. This also works for rational surfaces with finite positive weights. While these can be efficiently computed from the control points, there is a significant over-estimation of the error in some cases. Zhen et al. [ZS00] extended this approach for rational surfaces by directly using the homogeneous coordinates to compute tighter bounds. Methods based on the second derivatives give an upper bound for the parametric error which in turn is an upper bound for the geometric error. If the geometric error alone is required, slefes [Lut00] can be used as a tighter bound. The error is calculated as the distance between an upper and lower enclosing polygon which produces tighter bounds, especially for coarse tessellations.

GPU-based tessellation of parametric surfaces was introduced by Boubekeur et al. [BS05]. The approach is based on precomputed tessellation patterns that are rendered using instancing. The surface points are then computed in the vertex shader. Simultaneously, Guthe et al. [GBK05] also proposed using a fixed set of tessellation patterns. In addition, the tessellation factors are computed on the CPU for a given screen space error using bounds on the derivative of the surface [FMM86]. As independent tessellation of patches causes cracks which cannot be avoided when using fixed patterns, these where filled with line strips.

Since using a regular pattern for a complete surface patch leads to over-tessellation in flat regions, Eisenacher et al. proposed a recursive subdivision of the patches [EML09]. The subdivision is executed in parallel on the GPU, until a given screen space error is reached. While this significantly reduces the number of rendered triangles, it is only efficient, if enough sub-patches can be processed in parallel.

After introduction of the tessellation unit [Mic09] into the graphics pipeline, several methods using hardware tessellation where developed. Based on the approximation error computed from slefes, Yeo at al. [YBP12] propose a pixel accurate rendering of parametric surfaces. Later, they improved the approach to use tighter bounds based on variance of the surface [YBP14]. As the approximation error from slefes can only be used for Bézier patches, Hjelmervik et al. [Hje14] proposed an approach based on projecting bounds on the derivatives of the surface to screen space. For Bézier surfaces, they use the bounds proposed by Filip et al. [FMM86].

3. Preliminaries

For our proposed approach we consider bi-cubic tensor product Bézier surface patches using the Bernstein polynomial

$$\mathbf{s}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{c}_{i,j}\, B_i^3(u) B_j^3(v)\,. \tag{1}$$

where $\mathbf{c}_{i,j}$ are the control points of the patch which is defined over the parametric space $u, v \in D = [0,1] \times [0,1]$. Furthermore, the second order partial derivatives along u, v and the second order mixed derivatives are defined as:

$$\mathbf{s}_{u^2}(u,v) = \frac{\partial^2 \mathbf{s}(u,v)}{\partial u^2} = \sum_{i=0}^{1}\sum_{j=0}^{3} \mathbf{c}_{i,j}^{u^2} B_i^1(u) B_j^3(v)\,, \tag{2}$$

$$\mathbf{s}_{v^2}(u,v) = \frac{\partial^2 \mathbf{s}(u,v)}{\partial v^2} = \sum_{i=0}^{3}\sum_{j=0}^{1} \mathbf{c}_{i,j}^{v^2} B_i^3(u) B_j^1(v)\,, \tag{3}$$

$$\mathbf{s}_{uv}(u,v) = \frac{\partial^2 \mathbf{s}(u,v)}{\partial u \partial v} = \sum_{i=0}^{2}\sum_{j=0}^{2} \mathbf{c}_{i,j}^{u,v} B_i^2(u) B_j^2(v)\,. \tag{4}$$

Hereby, $\mathbf{c}_{i,j}^{u^2}$, $\mathbf{c}_{i,j}^{v^2}$ and $\mathbf{c}_{i,j}^{u,v}$ are the corresponding control points of $\mathbf{s}_{u^2}(u,v)$, $\mathbf{s}_{v^2}(u,v)$ and $\mathbf{s}_{uv}(u,v)$ with:

$$\mathbf{c}_{i,j}^{u^2} = 6(\mathbf{c}_{i,j} - 2\mathbf{c}_{i+1,j} + \mathbf{c}_{i+2,j})\,, \tag{5}$$

$$\mathbf{c}_{i,j}^{v^2} = 6(\mathbf{c}_{i,j} - 2\mathbf{c}_{i,j+1} + \mathbf{c}_{i,j+1})\,, \tag{6}$$

$$\mathbf{c}_{i,j}^{u,v} = 9(\mathbf{c}_{i,j} - \mathbf{c}_{i+1,j} - \mathbf{c}_{i,j+1} + \mathbf{c}_{i+1,j+1})\,. \tag{7}$$

For a more detailed discussion of Bézier curves and surfaces we refer to Farin et al. [Far01] and Piegl et al. [Pie95].

3.1. Error bounds

According to Filip et al. [FMM86] the introduced error while linearly approximating a C^2-continuous surface is bound by

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

with

$$M_u = \sup_{u,v\in D}\left\|\frac{\partial^2 \mathbf{s}}{\partial u^2}\right\|, M_v = \sup_{u,v\in D}\left\|\frac{\partial^2 \mathbf{s}}{\partial v^2}\right\|, M_{uv} = \sup_{u,v\in D}\left\|\frac{\partial^2 \mathbf{s}}{\partial u \partial v}\right\| \tag{9}$$

and $\mathbf{l}(u,v)$ be the linear approximation using right triangles of edge lengths Δu and Δv. Furthermore, as noted by Guthe et al.[GBK05], we can separate the sampling densities resulting in an approximation error which is bound by

$$\varepsilon \le \frac{1}{8}\left(\Delta u^2 (M_u + M_{uv}) + \Delta v^2 (M_v + M_{uv})\right)\,. \tag{10}$$

Using equation (10) we can estimate the minimal edge length for Δu and Δv to guarantee an upper bound approximation error ε while not over-tessellating the corresponding patch. Because ε is an upper bound, the error for each direction has to be at most $\frac{\varepsilon}{2}$:

$$\Delta u \geq \sqrt{\frac{4\varepsilon}{M_u + M_{uv}}} \quad \text{and} \quad \Delta v \geq \sqrt{\frac{4\varepsilon}{M_v + M_{uv}}}. \tag{11}$$

3.2. Uniform Tessellation

Applying our analysis of section 3.1 to tessellate bi-cubic Bézier surface patches we can compute the edge lengths of the linear approximating triangles Δu and Δv. Because the edge length is inverse to the corresponding tessellation parameter, those can be defined for both directions as:

$$t_u \geq \sqrt{\frac{M_u + M_{uv}}{4\varepsilon}} \quad \text{and} \quad t_v \geq \sqrt{\frac{M_v + M_{uv}}{4\varepsilon}}. \tag{12}$$

The bounds on the absolute values of the derivatives used in eq. (12) can be computed using the convex hull property of Bézier patches:

$$M_u \leq \max_{i=0..1, j=0..3} \left\| \mathbf{c}_{i,j}^{u^2} \right\|, \tag{13}$$

$$M_v \leq \max_{i=0..3, j=0..1} \left\| \mathbf{c}_{i,j}^{v^2} \right\|, \tag{14}$$

$$M_{uv} \leq \max_{i=0..2, j=0..2} \left\| \mathbf{c}_{i,j}^{uv} \right\|. \tag{15}$$

Edge Tessellation Factor For the edges we can simplify the aforementioned equations, because those are not depending on the mixed partial derivative M_{uv}:

$$t_{u_e} \geq \sqrt{\frac{M_u}{4\varepsilon}} \quad \text{and} \quad t_{v_e} \geq \sqrt{\frac{M_v}{4\varepsilon}}. \tag{16}$$

While not considering the mixed derivative guarantees a hole free tessellation as long as the control points of adjacent patches are identical, it can lead to an increased error along the boundary triangles of the patch [Hje14]. This can be solved by simply narrowing the border triangles using an scaling and translation of the interior parameters. For details on this, we refer to Hjelmervik et al. [Hje14].

4. Our Method

One of the main disadvantages of the aforementioned method, as illustrated in Figure 2, is the over-tessellation along areas of low curvature to guarantee a maximum approximation error ε.

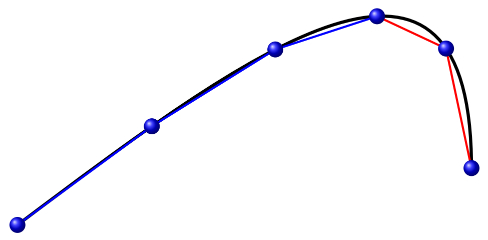

Figure 2: *Illustration of the approximation error for a constant tessellation factor as proposed in section 3.2. With increasing second derivative (right) the approximation error also increases. The two segments with the highest error are marked in red.*

To address this limitation, we re-position the samples on the surface to increase the density in high-curvature areas while decreasing it in areas of low curvature. As a first step, we compute local sampling densities t_u^l and t_v^l:

$$t_u^l(u,v) = \sqrt{\frac{\|\mathbf{s}_{u^2}(u,v)\| + \|\mathbf{s}_{uv}(u,v)\|}{4\varepsilon}}, \tag{17}$$

$$t_v^l(u,v) = \sqrt{\frac{\|\mathbf{s}_{v^2}(u,v)\| + \|\mathbf{s}_{uv}(u,v)\|}{4\varepsilon}}. \tag{18}$$

By integrating the local sampling density along the corresponding direction and taking the maximum over the other direction we compute the optimal sampling density for each direction, which leads to:

$$t_u \geq \sup_{v\in[0..1]} \int_0^1 \sqrt{\frac{\|\mathbf{s}_{u^2}(u,v)\| + \|\mathbf{s}_{uv}(u,v)\|}{4\varepsilon}} du \tag{19}$$

and

$$t_v \geq \sup_{u\in[0..1]} \int_0^1 \sqrt{\frac{\|\mathbf{s}_{v^2}(u,v)\| + \|\mathbf{s}_{uv}(u,v)\|}{4\varepsilon}} dv. \tag{20}$$

Additionally, we extended the bounds on derivatives of Filip. As we already mentioned above, we use the convex hull property of Bézier patches for an estimation of the upper bound. Because we limit ourself to bi-cubic surface patches, the second order partial derivatives along the u- and v- directions are linear depending on the corresponding parameter:

$$\|\mathbf{s}_{u^2}(u,v)\| \leq \max_{j\in[0..3]} \left((1-u)\left\|\mathbf{c}_{0,j}^{u^2}\right\| + u\left\|\mathbf{c}_{1,j}^{u^2}\right\| \right), \tag{21}$$

$$\|\mathbf{s}_{v^2}(u,v)\| \leq \max_{i\in[0..3]} \left((1-v)\left\|\mathbf{c}_{i,0}^{v^2}\right\| + u\left\|\mathbf{c}_{i,i}^{v^2}\right\| \right). \tag{22}$$

For the mixed partial derivative we stick to using a constant upper bound derived from the convex hull, because it is quadratic.

Using equations (19) to (22) we can determine ideal tessellation factors for each direction:

$$t_u \geq \sup_{j\in[0..3]} \int_{u=0}^1 \sqrt{\frac{(1-u)\left\|\mathbf{c}_{0,j}^{u^2}\right\| + u\left\|\mathbf{c}_{1,j}^{u^2}\right\| + M_{uv}}{4\varepsilon}} du, \tag{23}$$

$$t_v \geq \sup_{i\in[0..3]} \int_{v=0}^1 \sqrt{\frac{(1-v)\left\|\mathbf{c}_{i,0}^{v^2}\right\| + v\left\|\mathbf{c}_{i,1}^{v^2}\right\| + M_{uv}}{4\varepsilon}} dv. \tag{24}$$

Integrating over u and v respectively leads to:

$$t_u \geq \sup_{j\in[0..3]} \frac{\left(\left\|\mathbf{c}_{1,j}^{u^2}\right\| + M_{uv}\right)^{\frac{3}{2}} - \left(\left\|\mathbf{c}_{0,j}^{u^2}\right\| + M_{uv}\right)^{\frac{3}{2}}}{3\sqrt{\varepsilon}\left(\left\|\mathbf{c}_{1,j}^{u^2}\right\| - \left\|\mathbf{c}_{0,j}^{u^2}\right\|\right)} \tag{25}$$

and

$$t_v \geq \sup_{i\in[0..3]} \frac{\left(\left\|\mathbf{c}_{i,1}^{v^2}\right\| + M_{uv}\right)^{\frac{3}{2}} - \left(\left\|\mathbf{c}_{j,0}^{v^2}\right\| + M_{uv}\right)^{\frac{3}{2}}}{3\sqrt{\varepsilon}\left(\left\|\mathbf{c}_{j,1}^{v^2}\right\| - \left\|\mathbf{c}_{j,0}^{v^2}\right\|\right)}. \tag{26}$$

Interior Re-Parametrization We have now computed the total sampling density based on a local varying density. In the second step we therefore need to re-parameterize the uniform samples according to the local density. To reproduce the local sampling densities, the re-parameterization $\bar{u}(u,v)$ in u-direction has to fulfill the following condition:

$$\frac{\partial \bar{u}(u,v)}{\partial u} \propto \frac{1}{t_u^l(u,v)} . \tag{27}$$

Combining this with eq. (17) and the fact that for bi-cubic surface patches $\mathbf{s}_{u^2}(u,v)$ is linear in the u-direction, the new sample positions are then computed proportional from the old ones with:

$$\bar{u}(u,v) \propto \int_0^u \frac{1}{\sqrt{(1-u')\,\|\mathbf{s}_{u^2}(0,v)\| + u'\,\|\mathbf{s}_{u^2}(1,v)\| + M_{uv}}} du' , \tag{28}$$

where u' is the initial uniform sample position.

By solving eq. (28) for u and analogosly for v, we get

$$\bar{u}(u,v) = \frac{\sqrt{(1-u)M_{u^2}^0(v) + uM_{u^2}^1(v)} - \sqrt{M_{u^2}^0(v)}}{\sqrt{M_{u^2}^1(v)} - \sqrt{M_{u^2}^0(v)}} \tag{29}$$

and

$$\bar{v}(u,v) = \frac{\sqrt{(1-v)M_{v^2}^0(u) + vM_{v^2}^1(u)} - \sqrt{M_{v^2}^0(u)}}{\sqrt{M_{v^2}^1(u)} - \sqrt{M_{v^2}^0(u)}} , \tag{30}$$

with

$$\begin{aligned} M_{u^2}^0(v) &= \|\mathbf{s}_{u^2}(0,v)\| + M_{uv} , \\ M_{u^2}^1(v) &= \|\mathbf{s}_{u^2}(1,v)\| + M_{uv} , \\ M_{v^2}^0(u) &= \|\mathbf{s}_{v^2}(u,0)\| + M_{uv} , \\ M_{v^2}^1(u) &= \|\mathbf{s}_{v^2}(u,1)\| + M_{uv} . \end{aligned} \tag{31}$$

Given u and v for the current vertex, we then evaluate the surface at $(\bar{u}(u,v), \bar{v}(u,v))$.

Edge Tessellation Factor and Re-Parametrization Similar to the uniform tessellation without re-parametrization we can omit the mixed partial derivative M_{uv} for the edges and can compute the tessellation factor along the u-direction via

$$t_{u_e} \geq \frac{\left\|\mathbf{c}_{1,j}^{u^2}\right\|^{\frac{3}{2}} - \left\|\mathbf{c}_{0,j}^{u^2}\right\|^{\frac{3}{2}}}{3\sqrt{\varepsilon}\left(\left\|\mathbf{c}_{1,j}^{u^2}\right\| - \left\|\mathbf{c}_{0,j}^{u^2}\right\|\right)} , \tag{32}$$

where $j \in 0,3$ and the corresponding re-parameterized sample position:

$$\bar{u}_e(u,v) = \frac{\sqrt{(1-u)M_{u^2}^0(v) + uM_{u^2}^1(v))} - \sqrt{M_{u^2}^0(v)}}{\sqrt{M_{u^2}^1(v)} - \sqrt{M_{u^2}^0(v)}} , \tag{33}$$

with

$$M_{u^2}^0(v) = \|\mathbf{s}_{u^2}(0,v)\| \quad \text{and} \quad M_{u^2}^1(v) = \|\mathbf{s}_{u^2}(1,v)\| . \tag{34}$$

And for the v-direction the tessellation factor is defined as:

$$t_{v_e} \geq \frac{\left\|\mathbf{c}_{i,1}^{v^2}\right\|^{\frac{3}{2}} - \left\|\mathbf{c}_{i,0}^{v^2}\right\|^{\frac{3}{2}}}{3\sqrt{\varepsilon}\left(\left\|\mathbf{c}_{i,1}^{v^2}\right\| - \left\|\mathbf{c}_{i,0}^{v^2}\right\|\right)} , \tag{35}$$

where $i \in 0,3$ and the appropriated re-parametrization:

$$\bar{v}_e(u,v) = \frac{\sqrt{(1-v)M_{v^2}^0(u) + vM_{v^2}^1(u)} - \sqrt{M_{v^2}^0(u)}}{\sqrt{M_{v^2}^1(u)} - \sqrt{M_{v^2}^0(u)}} , \tag{36}$$

with

$$M_{v^2}^0(u) = \|\mathbf{s}_{u^2}(0,v)\| \quad \text{and} \quad M_{v^2}^1(u) = \|\mathbf{s}_{u^2}(1,v)\| . \tag{37}$$

Due to disregarding the mixed derivative for the edges and applying the same sampling density and re-parametrization on adjacent patches, the tessellation is guaranteed to be hole-free, as long as the control points of adjacent patches match. Similar to the uniform case, we solve the problem of increased error along boundary triangles by using the approach of Hjelmervik et al. [Hje14].

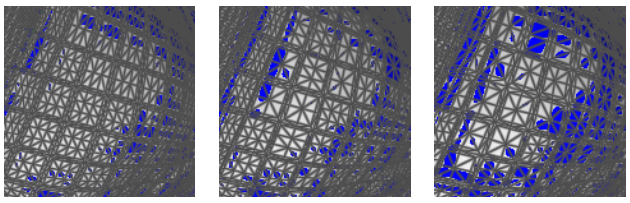
(a) *Naive Tessellation* **(b)** *Uni. Tessellation* **(c)** *Our Method*

Figure 3: *Comparison of the generated geometry of the aforementioned three different tessellation strategies.*

Figure 3 illustrates the difference in the resulting triangulation for a naive distance based tessellation (left), the uniform tessellation (middle) and ours (right). As can be seen reduces our method the number of generated triangles.

5. Implementation

Based on our preceding analyses we can guarantee an upper approximation error ε without over- or under-tessellating Bézier surface patches. To evaluate this we implemented the algorithm using OpenGL with its rendering pipeline including the tessellation stages [SA19]. This stage, with its *Tessellation Control Unit (TCS)* for computing the tessellation factors and the *Tessellation Evaluation Unit (TES)* for re-parametrizing and evaluating the patches fits ideally for our proposed algorithm.

Within the TCS we compute the tessellation factors as outlined in equations (19) and (20) for the inner part as well as equation (32) and eq. (35) for all four edges per patch. Furthermore, as the mixed partial derivative M_{uv} is constant over the patch, we can compute it in the TCS and pass it on to all following shaders. Additionally to that, we compute the control points of the partial derivatives $\mathbf{c}_{i,j}^{u^2}$ and $\mathbf{c}_{i,j}^{v^2}$ to reduce the number of computations in each TES invocation. In the next stage of the pipeline, the fixed-function *Tessellation*

Primitive Generator uses our computed factors and distributes the newly generated primitives uniformly over the patch.

After that the TES is invoked for each generated primitive for evaluating the patch at the corresponding (u,v)-parameter pairs. Before that, we re-parameterize the input parameter values with respect to the curvature based on our equations (29) and (30) for the inner part and eq. (33) and (36) for the outer ones. After this step, we compute new vertices by evaluating the patch at those new parameter values. Schäfer et al [SNK*14] outlined a fast and easy to implement algorithm for that.

5.1. View-dependent error bound

The approximation error ε introduced in section 3 is used as a constant factor for an upper bound while generating approximating geometry. Especially for visualization applications this can lead to an over-tessellation of the patches. Due to the perspective projection and in case the object is far off the viewing point this results in triangles of sub-pixel sizes include memory and processing overhead, as well as possible visual artifacts.

To overcome this limitation, we extend ε to take the distance to the camera into account. As an estimation of the distance between the viewing point and the patch, we use the distance between camera and bounding box of the patch control points. To produce a hole-free tessellation, the distance to the bounding box of the edge control points is used for the edges instead. In addition to the distance we introduce another user-defined parameter to specify the upper *pixel error*. The final approximation error ε is then computed from the pixel error, the screen resolution, camera field of view and distance to camera.

5.2. Tighter M_{uv} bound

Another improvement is using a tighter upper bound for the mixed partial derivative M_{uv}. As we mentioned in section 3, an upper bound of M_{uv} is the maximum norm of the control points of the second-order mixed partial derivative. A simply way to improve the error bound is to perform one midpoint subdivision of this bi-quadratic function. Computing a midpoint subdivision of a bi-quadratic patch is relatively cheap and we only need to compute 16 out of the 25 control points of the four sub-patches, since all control points adjacent to two or four sub-patches are convex combinations of adjacent control points. The remaining control points are the corners of the original patch and the new control points adjacent to them (see Figure 4).

5.3. Pre-compuation

We increase the throughput performance of our algorithm by reducing the computational operations per frame. This is achieved, similar to Yeo et al. [YBP12], by moving computations which are not depending on variable parameters such as the viewing point to a compute shader. For details on this we refer to Yeo et al. [YBP12].

6. Results and Discussions

We compared our proposed algorithm to a naive distance based level-of-detail implementation and a more advanced uniform tessellation strategy based on Filip et al. as described in section 3.2. We evaluate all three algorithms in terms of generated geometry, performance and visual errors. All algorithms were tested using models of different complexity and number of patches (cf. Figure 5). As a pre-processing step we converted all input models into a pure Bézier representation. For all our measurements we used an elliptic camera path of 500 frames around the models with varying distance to the object.

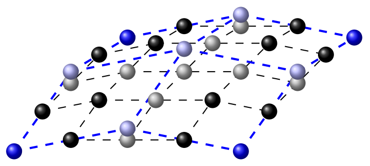
Figure 4: *Control points of the original mixed derivative (bi-quadratic Bézier patch) in blue and control points of four sub-patches after midpoint subdivision in black/grey. Only the dark control points are required to compute the convex hull, since the others are convex combinations of their neighbors.*

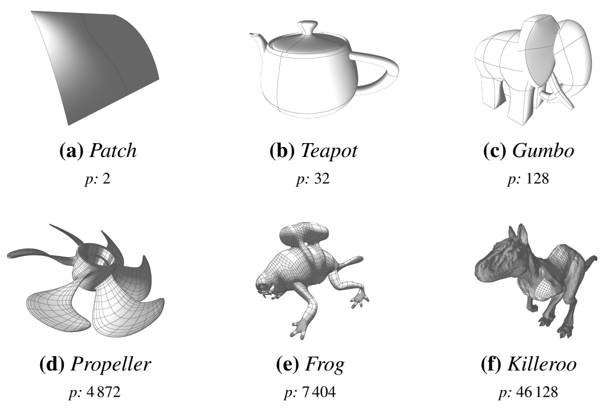

Figure 5: *Overview of the used test models. We converted all models into bi-cubic Bézier surface representations. Listed below each model are its patch count.*

We performed all measurements on an AMD Ryzen™ 7 2700X CPU, 32GB RAM and an NVIDIA GeForce™ RTX 2080 Ti running Linux 5.12.12, NVIDIA Driver Version 465.31 and Xorg 1.20.11 for display server.

6.1. Generated Geometry

The number of generated primitives are visualized in Figure 6 over the entire camera animation. As illustrated, our algorithm always outperforms uniform tessellation, while the Naive approach generates less geometry for distant camera positions. This is due to the fact that this implementation only considers the distance and no error bound constraint. A comparison of all algorithms for all models is shown in Table 1. In all our tests our method outperforms the other approaches in terms of the maximum number of generated primitives as well as the average one across the entire animation. For the Killeroo model our algorithm reduced the maximum number of generated primitives to 5.20% compared to the Naive implementation and 80.97% to uniform tessellation at close-up. On average, the Naive version still generates 2.27 and uniform tessellation 1.17 as many primitives as ours.

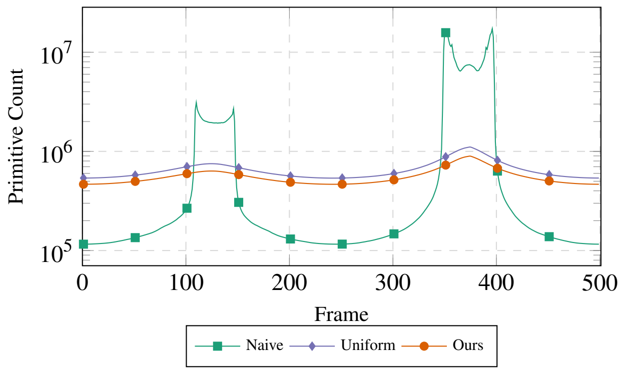

Figure 6: *Geometry generation of all three algorithms using the Killeroo model. For our tests we animate a camera path around the object on an elliptic path to guarantee different level-of-details.*

Improvement of bound on M_{uv} In section 5.2, we introduced a tighter bound for M_{uv}. The difference of the number of generated primitives compared to the upper bound by Filip et al. is illustrated in Figure 7. While never increasing the number of triangles, our bound reduces the number of generated primitives by 10% at close-ups.

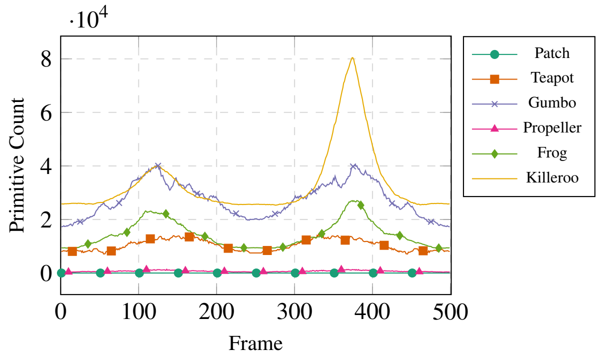

Figure 7: *Difference in the number of the generated primitives for our improved bound on M_{uv} and the upper bound by Filip et al. [FMM86].*

6.2. Performance

If we compare the performance of our approach to the others, we notice that despite the reduced geometry, there is unfortunately no significant advantages in terms of the average rendering time (cf. Table 2).

A closer look at uniform tessellation and our approach reveals that we are despite the reduced number of generated primitives only on par with respect to rendering time. This effect is a result of the more complex evaluations within the rendering pipeline. As the upper bound of Filip et al. is defined directly by the convex hull property, our tighter bound requires an additional (partial) midpoint subdivision as described in section 5.2. In addition, for each newly generated pair of (u,v)-coordiantes the equations (29) and (30) for interior and equations (33) and (36) for the border vertices have to be evaluated.

Using our approach, but also using uniform tessellation, we measured high frame rates in all our tests even for close-ups. Using the Killeroo model a minimal frame rate of 717.22 is achieved using our method. In contrast to the naive distanced based level-of-detail (LoD) approach, which drops to 78.57 we achieve a nearly ten times higher frame rate. Even if we increase the number of input patches, we still have acceptable frame rates. We rendered 5×5 Killeroo models (1 153 200 patches) without instancing at a frame rate of 182.77 FPS for a close-up while the LoD method drops below 30 FPS. The reason for this drop is the number of generated primitives as the rasterizers become the bottleneck [NVI18].

Figure 8 illustrates the frame rates for the Killeroo model across all three algorithms over the entire camera animation. If we compare the graphs for the number of generated primitives (Figure 6) with the frame rates, we can see a direct correlation between the frame rate drop and the number of generated triangles.

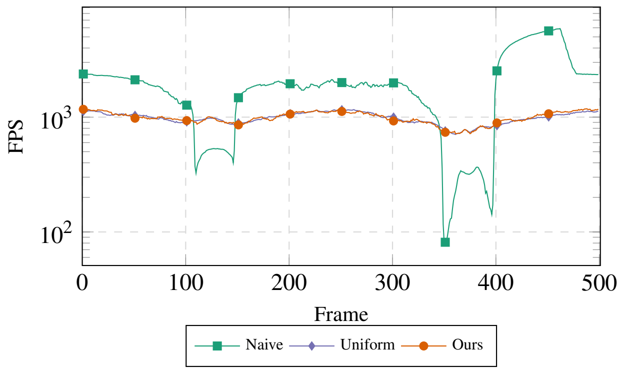

Figure 8: *Comparison of the performance for all algorithms using the Killeroo model. The frame rate drops for the Naive variant is due to the number of generated primitives (cf. Figure 6).*

TCS vs. Compute Shader As already mentioned, the evaluation of our upper bound on M_{uv} is more complex, as it requires computing additional control points. Furthermore, if we evaluate this within the TCS it has to be computed in each frame for each patch, even if the control points haven't changed (e.g. by object animation). This is a superfluous computation which can be precomputed. By moving this computation to a separate compute shader we experienced an increase of the average frame rate of 24.04% for the Killeroo model in our tests. A detailed comparison over the camera animation is shown in Figure 9.

6.3. Visual Errors

Using our approach we achieved an averaged visual error of 0.51 pixels for the Killeroo model at high frame rates. Figure 10 shows

Model	Number of generated Primitives									
	Ours		Naive				Uniform			
	Max	Avg	Max		Avg		Max		Avg	
Patch	**16 062**	**16 059**	16 384	(×1.02)	16 349	(×1.02)	**16 062**	(×1.00)	**16 059**	(×1.00)
Teapot	**132 331**	**76 972**	202 496	(×1.53)	121 663	(×1.58)	152 631	(×1.15)	92 319	(×1.20)
Gumbo	**428 776**	**270 443**	803 337	(×1.87)	465 083	(×1.72)	487 926	(×1.14)	317 453	(×1.17)
Propeller	**242 892**	**183 090**	4 028 703	(×16.59)	1 183 851	(×6.47)	275 267	(×1.13)	205 975	(×1.12)
Frog	**382 787**	**224 624**	9 730 247	(×25.42)	1 521 384	(×6.77)	449 061	(×1.17)	261 955	(×1.17)
Killeroo	**895 653**	**557 054**	17 232 927	(×19.24)	1 264 945	(×2.27)	1 106 172	(×1.24)	654 078	(×1.17)
Herd	**8 588 282**	**6 487 955**	183 989 842	(×21.42)	12 418 953	(×1.91)	10 042 864	(×1.17)	7 568 086	(×1.17)

Table 1: *Maximum and average number of primitives generated for each model and algorithm with reduction of geometry compared to ours.*

Model	Framerate in [FPS]									
	Ours		Naive				Uniform			
	Min	Avg	Min		Avg		Min		Avg	
Patch	4 648.42	8 124.34	6 026.09	(×1.30)	11 826.11	(×1.46)	5 627.08	(×1.21)	10 430.03	(×1.28)
Teapot	1 449.78	3 552.38	1 076.67	(×0.74)	3 123.68	(×0.88)	1 724.65	(×1.19)	3 903.81	(×1.10)
Gumbo	995.78	1 920.63	815.71	(×0.82)	1 564.51	(×0.81)	957.37	(×0.96)	1 838.31	(×0.96)
Propeller	1 323.31	2 610.17	310.09	(×0.23)	1 732.85	(×0.66)	1 474.85	(×1.11)	2 696.94	(×1.03)
Frog	969.02	2 274.24	118.30	(×0.12)	4 368.94	(×1.92)	971.74	(×1.00)	2 518.03	(×1.11)
Killeroo	717.22	989.69	78.57	(×0.11)	2 029.39	(×2.05)	706.58	(×0.99)	984.31	(×0.99)
Herd	182.77	200.61	18.30	(×0.10)	251.80	(×1.26)	186.93	(×1.02)	216.07	(×1.08)

Table 2: *Minimum and average frame rates for all algorithms across our used models. Despite the reduced number of primitives, our algorithm only outperforms uniform tessellation for two out of six test models.*

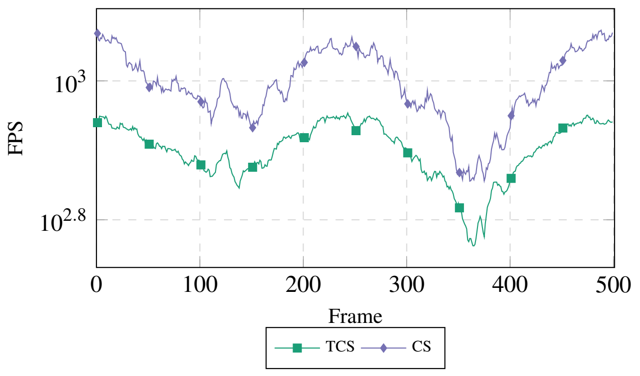

Figure 9: *Performance comparison of our upper bound on M_{uv} computed in the compute shader (CS) and the tessellation control shader (TCS). As the upper bound of mixed derivates is depending only on the control points, this can be pre-computed as long the geometry isn't changing.*

the screen space error in pixels for all algorithms. The error stays below one pixel for our approach except for two frames during an extreme close-up. This occurs due to the fact the slanted triangles at the border of the patches violate the triangle shape assumption of Hjelmervik et al. [Hje14]. A zoomed detail view of the region of interest is visualized in Figure 11.

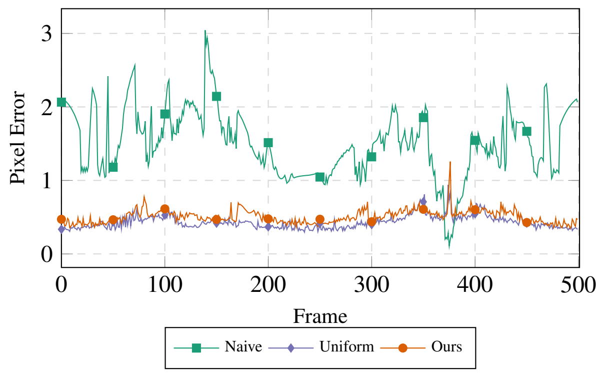

Figure 10: *Temporal progression of the screen space error in pixels for the Killeroo model across all three algorithms. In general uniform tessellation as well as our approach guarantee an error below one pixel. This error bound is only exceeded in case of highly slanted border triangles (cf. Figure 11).*

7. Conclusions

We have proposed an approach to tessellate and render bi-cubic Bézier patches using a re-parameterization of the sampling pattern to account for local curvature and to reduce over-tessellation in flat regions. Furthermore, we improved the upper bound on the mixed derivative introduced by Filip et al. [FMM86] to further reduce the number of generated primitives. Based on our analysis we reduced the number of triangles by approximately 15% compared to uniform tessellation using the error bounds from Filip et al. without exceeding the given upper bound error.

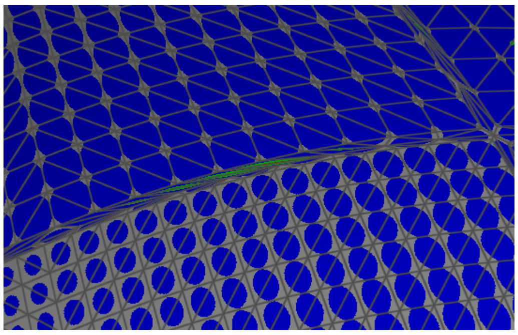

Figure 11: *Zoom into detail of the Killeroo model where the error bound of one pixel is exceeded (red pixels). This occurs at the outer strip of the tessellation at slanted triangles, which violate the triangle shape assumption of Hjelmervik et al. [Hje14].*

In our evaluations we observed a frame rate comparable to uniform tessellation. Despite the reduced number of primitives, our method is only on par because of the more complex upper bound computation and the following re-parameterization in the evaluation shader. If additional computations have to be performed per vertex, the reduced number of triangles would however still be an advantage. In addition, we achieve a slightly lower error at maximum tessellation level.

Acknowledgements

We gratefully acknowledge Bay Raitt's "Monster Frog". The Killeroo model is courtesy of Headus Pty Ltd (https://www.headus.com.au/).

References

[BS05] Boubekeur, Tamy and Schlick, Christophe. "Generic Mesh Refinement on GPU". *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. HWWS '05. Los Angeles, California: Association for Computing Machinery, 2005, 99–104. isbn: 1595930868. doi: 10.1145/1071866.1071882. url: https://doi.org/10.1145/1071866.1071882 2.

[EML09] Eisenacher, Christian, Meyer, Quirin, and Loop, Charles. "Real-time view-dependent rendering of parametric surfaces". *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D '09*. ACM Press, 2009. doi: 10.1145/1507149.1507172 2.

[Far01] Farin, Gerald. *Curves and Surfaces for Cagd: A Practical Guide*. MORGAN KAUFMANN PUBL INC, Oct. 2001. 520 pp. isbn: 1558607374 2.

[FMM86] Filip, Daniel, Magedson, Robert, and Markot, Robert. "Surface algorithms using bounds on derivatives". *Computer Aided Geometric Design* 3.4 (Dec. 1986), 295–311. doi: 10.1016/0167-8396(86)90005-1 1, 2, 6, 7.

[GBK05] Guthe, Michael, Balázs, Aákos, and Klein, Reinhard. "GPU-based trimming and tessellation of NURBS and T-Spline surfaces". *ACM Transactions on Graphics* 24.3 (July 2005), 1016–1023. doi: 10.1145/1073204.1073305 2.

[Hje14] Hjelmervik, Jon. "Direct Pixel-Accurate Rendering of Smooth Surfaces". *Mathematical Methods for Curves and Surfaces*. Springer Berlin Heidelberg, 2014, 238–247. doi: 10.1007/978-3-642-54382-1_14 2–4, 7, 8.

[Lut00] Lutterkort, D. *Envelopes of Nonlinear Geometry*. PhD thesis, Purdue University. 2000 2.

[Mic09] Microsoft Corporation. *Direct3D 11 Features*. http://msdn.microsoft.com/en-us/library/ff476342(VS.85).aspx. 2009 2.

[NVI18] NVIDIA Corporation. *NVIDIA Turing Architecture In-Depth*. https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth/. 2018 6.

[Pie95] Piegl, Les. *The NURBS Book*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. isbn: 9783642973857 2.

[SA19] Segal, Mark and Akeley, Kurt. *The OpenGL Graphics System: A Specification(Version 4.6 (Core Profile) - October 22, 2019)*. Tech. rep. The Khronos Group Inc, Oct. 2019. url: https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf 4.

[SNK*14] Schäfer, H., Niessner, M., Keinert, B., et al. *State of the Art Report on Real-time Rendering with Hardware Tessellation*. en. 2014. doi: 10.2312/EGST.20141037 5.

[YBP12] Yeo, Young In, Bin, Lihan, and Peters, Jörg. "Efficient pixel-accurate rendering of curved surfaces". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '12*. ACM Press, 2012. doi: 10.1145/2159616.2159644 2, 5.

[YBP14] Yeo, Young In, Bhandare, Sagar, and Peters, Jörg. "Efficient Pixel-accurate Rendering of Animated Curved Surfaces". *Mathematical Methods for Curves and Surfaces*. Springer Berlin Heidelberg, 2014, 491–509. doi: 10.1007/978-3-642-54382-1_28 2.

[ZS00] Zheng, Jianmin and Sederberg, Thomas W. "Estimating Tessellation Parameter Intervals for Rational Curves and Surfaces". *ACM Trans. Graph.* 19.1 (Jan. 2000), 56–77. issn: 0730-0301. doi: 10.1145/343002.343034. url: https://doi.org/10.1145/343002.343034 2.