Interactive Pixel-Accurate Rendering of LR-Splines and T-Splines

Jon M. Hjelmervik[1] and Franz G. Fuchs[1]

[1]SINTEF ICT, Forskningsveien 1, 0314 Oslo, Norway

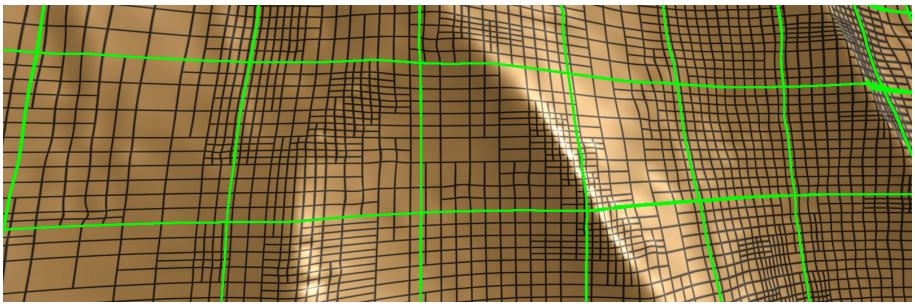


Figure 1: *The rendering algorithm decouples the patch geometry (black lines) from the tesselator geometry (green lines).*

Abstract

Flexible surface types on irregular grids, such as T-splines and LR-splines, are gaining popularity in science and industry due to the possibility for local grid refinement. We present a novel rendering algorithm for those surface types that guarantees pixel-accurate geometry and water-tight tessellation (no drop-outs). Before rendering, we extract the Bézier coefficients. The resulting irregular grids of Bézier patches are then rendered using a multi-stage algorithm, that decouples the tesselator and the patch geometry. The implementation using OpenGL utilizes compute shaders and hardware tessellation functionality. We showcase interactive rendering achieved by our approach on three representative use cases.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms

1. Introduction and related work

Smooth surfaces are popular both in engineering as well as entertainment industries. A commonly used representation is tensor product NURBS surfaces, which consist of a set of polynomial patches laid out in a regular grid. Recent works have demonstrated rendering methods providing reliable, and visually pleasing results, including associated data such as textures or scientific data. The drawback of this representation is that local refinement is not possible, as the grid must be refined along a line in one of the parameter directions. We are therefore seeing a shift towards more flexible surface types, such as T-Splines [SZBN03] and LR B-Splines [DLP13].

The hardware tessellation functionality available in GPUs is based on dividing the surface into *tessellation patches* which can be tessellated individually. See [SNK*14] for a thorough description of hardware tessellation and state-of-the-art algorithms. For NURBS surfaces it is common to let each polynomial (Bézier) patch be a tessellation patch. A benefit with this choice is that two neighboring patches share one common edge, which is required for a tessellation

DOI: 10.2312/egsh.20151016

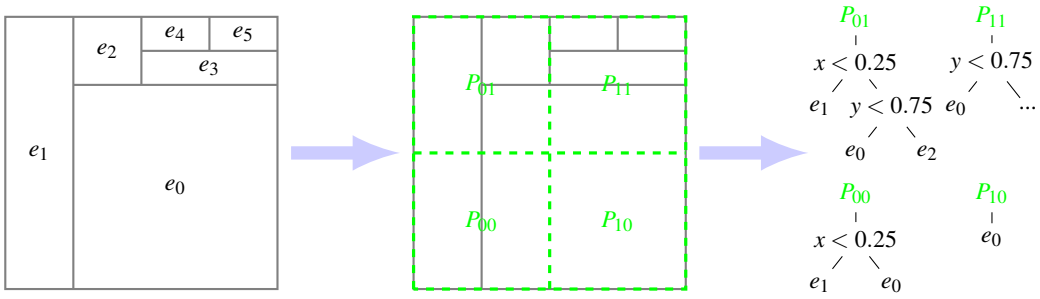


Figure 2: *Left: LR elements e_i. Middle: Tessellation patches $P_{j,k}$. Right: Search forest.*

without visible cracks. In the case of splines on unstructured grids this is typically not the case.

Catmull-Clark subdivision surfaces, where the limit surface consist of a set of bicubic B-spline surfaces, bear strong connections to B-spline surfaces. Nießner et. al. [NLMD12] presented a rendering algorithm for Catmull-Clark subdivision surfaces, where the control net is refined near extraordinary surfaces to simplify the tessellation and evaluation implementation and improve its performance. A similar approach in our case would be to insert tessellation patches such that each patch consists of one Bézier patch, and that neighboring patches share a common edge. However, our data sets have a large number of extraordinary vertices. This strategy would therefore require a very large number of tessellation patches leading to low rendering performance. Instead, we chose the opposite approach: The proposed method uses uniformly distributed tessellation patches independent of the structure of the underlying Bézier patches. Those Bézier patches are subsequently referred to as "elements".

Yeo et al. [YBP12] presented a method for computing a sufficient tessellation level to obtain pixel-accurate rendering. Their implementation is a two-pass algorithm that first computes the tessellation level based on SLEVEs [Lut00]. In the second pass, the tessellation control shader reads the tessellation level for the patch. In order to ensure that the curve between two neighboring patches is tessellated with the same level, also the neighboring tessellation level is read. Hjelmervik [Hje14] presented a single pass algorithm for the same problem, where the error estimate is computed based on upper bounds on the second order derivatives of the surface. Both methods produce similar results.

In this article, we will build on [Hje14, YBP12] and extend the approach to a larger class of surface types. We present a multi-pass algorithm that separates the computation of tessellation level from the rendering pass from the SLEVEs approach. However, we build upon the error estimate from [Hje14], because it only requires that each tessellation patch is C^2 continuous.

2. The proposed method

Appropriate methods for interactive rendering of spline surfaces should be both pixel-accurate and water-tight. There are two main challenges when developing methods using the hardware tessellator, namely to compute an appropriate tessellation level and to efficiently evaluate the surface at the newly generated vertices. We will start by describing the error estimate resulting in pixel-accurate surface evaluation.

2.1. Pixel-accurate tessellation

The hardware tessellator provides a valid triangulation for the patches of a surface $S(u,v)$. In OpenGL a tessellation control shader determines the tessellation level. Ideally, one wants as few triangles as possible, while ensuring that the error between the original surface and its tessellated counterpart is less than half a pixel, i.e.,

$$\Delta P := 2\left\|\pi_s(S(u,v)) - \begin{bmatrix}x\\y\end{bmatrix}\right\|_\infty \leq 1 \tag{1}$$

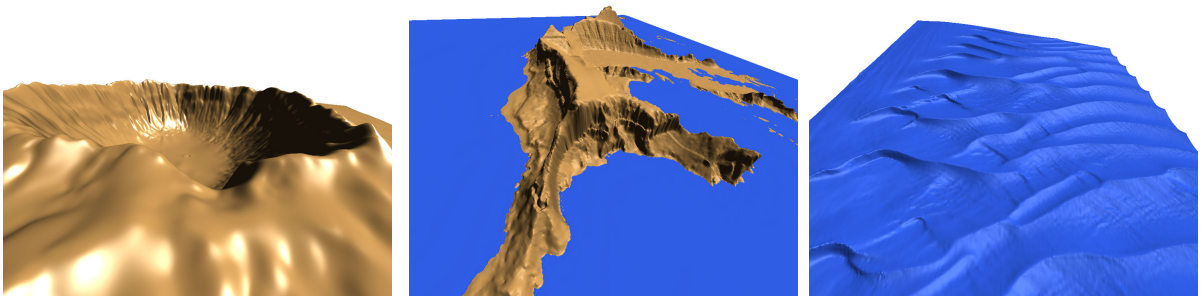
where $\begin{bmatrix}x\\y\end{bmatrix}$ is the pixel's center and $\pi_s : \mathbb{R}^3 \to \mathbb{R}^2$ is the projection to the screen. The linear interpolation error (to a C^2 function) can be used to determine the sampling distance of a curve. As presented in [Hje14], this leads to to a view dependent error bound, that depends on upper bounds on the second order derivatives and the distance in eye space (estimated by the control points of S), i.e.,

$$M(\partial_u\partial_u S),\ M(\partial_u\partial_v S),\ M(\partial_v\partial_v S),\ MS \tag{2}$$

where M is the model view matrix. This error bound is the foundation of the following algorithm for pixel-accurate rendering of Bézier patches on unstructured grids.

2.2. Overall algorithm

Our method guarantees a geometrically pixel-accurate rendering based on [Hje14]. Before rendering the Bézier coefficients of each patch are extracted, making the rendering applicable to both LR- and T-spline surfaces. Each frame the algorithm consists of the following stages (see Figure 2 for notation):



(a) Barringer Crater, Arizona, USA.

(b) Terrain of Værøy, Norway. Courtesy of kartverket.no.

(c) Bathymetry of Banc du Four. Courtesy of Laboratoire Domaines Océaniques.

Figure 3: *Visualization examples with Blinn-Phong shading of data represented by LR B-splines.*

Method 1 (Pixel-Accurate Method)

1. *Per Element Compute Shader (**PEC-shader**):* In the first stage we compute bounds for the expressions in Equation (2) for each LR-spline element e_i. The resulting 12 float values per element are stored in an image buffer.
2. *Per Patch Compute Shader (**PPC-shader**):* The second stage loops over all elements e_i that belong to a patch $P_{j,k}$ computing upper and lower bounds of the expressions in Equation (2). The result in [Hje14] is then used to calculate the inner tessellation level for each tessellation patch. To optimize the performance, elements outside the view-frustum are culled.
3. *Rendering Stage:* The previous two compute shaders provide the inner tessellation level for each tessellation patch $P_{j,k}$, which guarantee pixel-accurate rendering of the geometry. This is used in a normal render stage.

We also developed a **heuristic method** which, in all our examples tested, is roughly twice as fast, while leading to very similar results.

Method 2 (Heuristic Method) This method is a simplification of Method 1. The first stage (PEC-shader) is changed to compute and store the tessellation level *locally per element* according to the error bound for pixel-accurate tessellation given in [Hje14]. This results in one integer value per element, stored in an image buffer. The next stage (PPC-shader) then loops over all elements that belong to that patch and uses the maximum tessellation level encountered. The rendering stage is identical to the one in Method 1.

Both methods evaluate the surface in the rendering stage. We will continue by describing efficient means of evaluating surfaces.

2.3. Fast look-up tables

The proposed methods in Section 2.2 need a fast and reliable algorithm for evaluating spline surfaces on unstructured grids. Given a point in the parameter domain one needs to find which element e_i this point belongs to, see Figure 2. To improve the performance of the evaluation, we first convert the surfaces into Bézier segments. This can easily be done by computing the control points of the interpolating polynomial. This strategy has the added advantage that our implementation is independent of the surface representation.

The common strategy for rendering Bézier surfaces is to pass the control points through the tessellation shader stages, and use well known algorithms such as the De Casteljau's algorithm for evaluation. On unstructured grids, one needs an additional step to find which element the parameter value belongs to. Referring to Figure 2 for an illustration, the algorithm is as follows.

- Split the parameter domain into a regular grid (similar to the tessellation grid).
- For each grid cell, build a kd-tree for all active elements.

The head of each search tree is stored in a texture for rapid lookups. In the case of a single active element in a grid cell, the element index is stored instead. Please note that the rendered surface is crack free also across element boundaries, since the evaluation procedure always produces the same value for a given parameter value.

3. Results

We present visualization of LR B-spline surfaces in three use cases. The Barringer Crater, USA (Figure 3(a)) with 32193 LR elements, the terrain of Værøy, Norway (Figure 3(b)) with 429007 LR elements, and the bathymetry of Banc du Four, France (Figure 3(c)) with 59824 LR elements. All data sets have a large number of extraordinary vertices, cf. Figure 1.

In Table 1 we can see how the number of tessellation patches influences the total rendering time of the surface. The time for the PEC compute shader is independent of the zoom-level and the number of patches. For the examples in Figures 3(a)-(c) the PEC shader takes respectively

		Barringer Crater, see Figure 3(a)					Terrain of Værøy, see Figure 3(b)					Banc du Four, see Figure 3(c)				
	#patches	32^2	64^2	128^2	256^2	512^2	32^2	64^2	128^2	256^2	512^2	32^2	64^2	128^2	256^2	512^2
Method 1	#prim[$\times 10^3$]	823	590	460	468	702	2369	2744	2480	2017	1688	599	362	290	320	550
	max(ΔP)	1.75	1.00	0.78	0.97	0.97	3.69	2.47	1.82	0.68	0.62	1.31	0.67	0.67	0.56	0.51
	PPC[ms]	0.45	0.18	0.12	0.11	0.16	24.0	11.0	5.5	2.4	1.4	1.10	0.39	0.23	0.21	0.24
	render[ms]	0.45	0.40	0.42	0.56	1.30	2.4	2.9	3.1	3.1	4.1	0.40	0.34	0.33	0.49	1.30
	total [ms]	0.96	0.64	0.60	0.72	1.50	27.0	15.0	9.2	6.1	6.1	1.50	0.78	0.62	0.75	1.60
Method 2	#prim[$\times 10^3$]	684	504	426	450	691	2127	2439	2288	1893	1626	524	335	276	310	546
	max(ΔP)	2.41	1.04	1.00	0.97	0.97	3.69	1.92	1.46	0.93	0.62	1.16	0.80	0.67	0.67	0.51
	PPC[ms]	0.09	0.04	0.02	0.04	0.10	8.2	3.7	1.9	0.7	0.4	0.27	0.08	0.04	0.05	0.10
	render[ms]	0.43	0.42	0.43	0.57	1.30	2.3	2.7	2.9	3.0	4.0	0.37	0.32	0.32	0.48	1.20
	total [ms]	0.57	0.50	0.50	0.65	1.40	11.0	6.8	5.3	4.1	4.9	0.68	0.45	0.40	0.57	1.30

Table 1: *Comparison of performance of rendering with the proposed methods, with a screen resolution* 1024×768 *on an NVIDIA Titan GPU. Pixel-accurate results are higlighted with gray. A value of max(ΔP) slightly lower than 1 is ideal.*

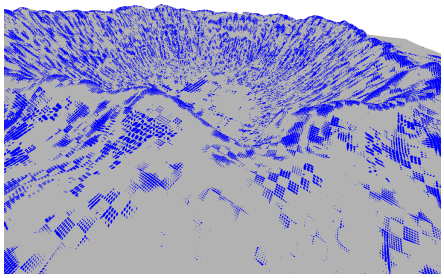


Figure 4: *Visualization of parametric accuracy of the Barringer Crater. The gray color indicates less than 10% error, and blue 10%-100%. The error never exceeds the given tolerance. The blue color indicates no over-tessellation.*

0.054/0.046 ms, 0.57/0.43 ms, and 0.059/0.042 ms for the pixel-accurate/heuristic method.

All examples in Table 1 show that too few tessellation patches can result in parametric inaccuracy. In those cases, there were patches that reached the maximum tesselation level of 64 triangles which was not enough for a pixel accurate result. The larger a patch, the more likely it contains an element with a large (absolute) second order derivative of the surface, dictating a high tessellation level for the whole patch. Therefore, more patches potentially allow a more localized adaptation. As Table 1 shows, the number of primitives decreases with increased number of patches, leading to faster rendering times. For the Barringer Crater and Banc du Four a minimum is reached at 128^2 patches. For more patches the number of primitives increases again since each patch generates at least 2 triangles. Figure 4 shows a visualization of the parametric accuracy.

The heuristic method resulted in a speedup factor of roughly 2, without detectable loss of visual quality in our test cases. In comparison, the method in [Hje14] generated roughly the same number of primitives and was slightly faster (but not watertight).

4. Conclusion and Future Work

The presented approach allows interactive rendering of splines on unstructured grids. The approach ensures pixel-accurate geometry and water-tight tessellation. We showcase the use on three different cases. In the future we will implement a simplified algorithm for 2.5 dimensional surfaces that is expected to drastically improve rendering times. A further improvement of the proposed method is to allow for a non-uniform tessellation patch structure, with good heuristics to ensure optimal performance.

References

[DLP13] DOKKEN T., LYCHE T., PETTERSEN K. F.: Polynomial splines over locally refined box-partitions. *Comput. Aided Geom. Des. 30*, 3 (Mar. 2013), 331–356. 1

[Hje14] HJELMERVIK J.: Direct pixel-accurate rendering of smooth surfaces. In *Mathematical Methods for Curves and Surfaces, 2012*, vol. 8177 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014, pp. 238–247. 2, 3, 4

[Lut00] LUTTERKORT D. C.: *Envelopes of nonlinear geometry*. PhD thesis, Purdue University, West Lafayette, IN, USA, 2000. AAI3017831. 2

[NLMD12] NIESSNER M., LOOP C., MEYER M., DEROSE T.: Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces. *ACM Trans. Graph. 31*, 1 (Feb. 2012), 6:1–6:11. 2

[SNK*14] SCHÄFER H., NIESSNER M., KEINERT B., STAMMINGER M., LOOP C.: State of the Art Report on Real-time Rendering with Hardware Tessellation. Lefebvre S., Spagnuolo M., (Eds.), Eurographics Association, pp. 93–117. 1

[SZBN03] SEDERBERG T. W., ZHENG J., BAKENOV A., NASRI A.: T-splines and t-nurccs. *ACM Trans. Graph. 22*, 3 (July 2003), 477–484. 1

[YBP12] YEO Y. I., BIN L., PETERS J.: Efficient pixel-accurate rendering of curved surfaces. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, ACM, pp. 165–174. 2