Huang YC, Feng JQ, Nießner M *et al.* Feature-adaptive rendering of loop subdivision surfaces on modern GPUs. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 29(6): 1014–1025 Nov. 2014. DOI 10.1007/s11390-014-1486-x

Feature-Adaptive Rendering of Loop Subdivision Surfaces on Modern GPUs

Yun-Cen Huang[1] (黄韵岑), *Member*, *CCF*, Jie-Qing Feng[1,*] (冯结青), *Senior Member*, *CCF*
Matthias Nießner[2], *Member*, *ACM*, Yuan-Min Cui[1] (崔元敏), *Member*, *CCF*, and Baoguang Yang[3] (杨宝光)

[1]*State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou 310058, China*

[2]*Computer Graphics Laboratory, Stanford University, Stanford, CA 94305, U.S.A.*

[3]*Qualcomm Technologies Incorporation, San Diego, CA 92121, U.S.A.*

E-mail: huangyuncen@zjucadcg.cn; jqfeng@cad.zju.edu.cn; niessner@cs.stanford.edu; cuiyuanmin@cad.zju.edu.cn
yang.nick.baoguang@gmail.com

Received December 17, 2013; revised May 14, 2014.

Abstract We present a novel approach for real-time rendering Loop subdivision surfaces on modern graphics hardware. Our algorithm evaluates both positions and normals accurately, thus providing the true Loop subdivision surface. The core idea is to recursively refine irregular patches using a GPU compute kernel. All generated regular patches are then directly evaluated and rendered using the hardware tessellation unit. Our approach handles triangular control meshes of arbitrary topologies and incorporates common subdivision surface features such as semi-sharp creases and hierarchical edits. While surface rendering is accurate up to machine precision, we also enforce a consistent bitwise evaluation of positions and normals at patch boundaries. This is particularly useful in the context of displacement mapping which strictly requires matching surface normals. Furthermore, we incorporate efficient level-of-detail rendering where subdivision depth and tessellation density can be adjusted on-the-fly. Overall, our algorithm provides high-quality results at real-time frame rates, thus being ideally suited to interactive rendering applications such as video games or authoring tools.

Keywords real-time rendering, Loop subdivision surface, hardware tessellation

1 Introduction

Catmull and Clark[1] and Doo and Sabin[2] introduced a new era of smooth surface modeling by enabling arbitrary connectivity on parametric surfaces, i.e., subdivision surfaces. Loop's method[3] stands out as a representative among different subdivision schemes[4] and is heavily used in industry applications such as video games or finite element simulations. It is specifically designed for triangular control meshes of arbitrary topology, and in contrast with Catmull-Clark subdivision, Loop subdivision maintains a triangular structure during the subdivision process.

Traditionally, subdivision surfaces are evaluated by recursively refining the underlying control mesh according to the corresponding subdivision rules. Though this can be easily implemented on today's GPUs, the exponential growth of memory storage and data transfer severely affects rendering performance. With the advent of GPU hardware tessellation in DirectX 11①, tessellated geometry can be generated and directly processed (i.e., rendered) on the GPU streaming multiprocessors[5]. This avoids costly memory I/O and allows for high-quality surface rendering involving fine-scale geometric detail. However, surface rendering using hardware tessellation requires direct per patch evaluation. While direct evaluation of regular patches is straightforward, it is non-trivial for irregularities such as extraordinary vertices, semi-sharp creases, and hierarchical edits. The milestone work by Stam[6-7] was the first one that proposed such a direct evaluation scheme for irregular patches at arbitrary parameter locations. However, significant computational overhead, e.g., vertex transformations to *eigen-space*, limits real-time performance. Most recently, Nießner *et al.*[8] proposed a combination of recursive refinement at features and direct evaluation of regular regions. This turns out to be faster than Stam's direct evaluation approach and also

Regular Paper
This work was supported by the National Natural Science Foundation of China under Grant No. 61170138 and the Program for New Century Excellent Talents in University of China under Grant No. NCET-10-0728.

*Corresponding Author

①http://msdn.microsoft.com/en-us/library/ff476342(VS.85).aspx, Sept. 2014.

handles subdivision surfaces features (e.g., creases). While this allows for fast and efficient surface rendering on modern GPUs, it focuses on Catmull-Clark subdivision surfaces. In this work, we adopt the pipeline of feature adaptive subdivision by fully exploiting the parallelism of modern GPUs and the hardware tessellation architecture. However, in contrast to the original feature adaptive approach, we focus on Loop subdivision surfaces. Therefore, we carefully extend the watertight evaluation method from bicubic patches to quartic box-spline patches. Our method also supports the rendering of semi-sharp creases and hierarchical editing. Further, we propose a level-of-detail scheme specifically crafted for triangular Loop meshes by adaptively adjusting the subdivision depth and tessellation density.

2 Related Work and Preliminary

2.1 Related Work

Traditional evaluation of subdivision surfaces is performed by iterative global refinement following the Loop subdivision rules. Therefore, Pulli and Segal[9] proposed packing two triangles into one quadrilateral to minimize the memory overhead. Shiue *et al.*[10] partitioned the control mesh into a set of *frag-meshes* and refined them respectively based on a pre-computed valence-related lookup table. However, due to the valence-related lookup table, it is not directly applicable to the control meshes of arbitrary topology. In addition, pre-refinement step is required. Kim and Peters[11] extended Shiue's method to Loop subdivision surfaces. Boubekeur and Schlick[12] decreased the computational complexity by adaptively refining the control mesh according to pre-computed adaptive patch refinement patterns. While these methods achieve real-time rates, the exponential growth in memory severely affects render performance due to the bandwidth limitations on modern GPUs.

Stam evaluation[6-7] overcomes this limitation by performing direct evaluation of the subdivision surface. Therefore, the respective Catmull-Clark or Loop subdivision surface can be analytically evaluated via eigen analysis of the corresponding subdivision matrices. Although Stam's algorithm can be implemented on the GPU, computational overhead and code branching impede high-performance GPU evaluation. Further, it is practically infeasible to extend the algorithm to handle semi-sharp creases and hierarchical editing due to combinatoric issues. Rather than evaluating polynomials for regular patches, Bischoff *et al.*[13] accelerated Loop subdivision surface evaluation on GPU via forward differences.

In order to speed up rendering, researchers developed several approximate techniques, however, at the cost of surface quality and accuracy. The basic idea of these approaches is to find an approximate closed-form solution in order to efficiently evaluate irregular patches. For example, a curved PN-triangle decouples the geometry and its normal information, i.e., a cubic triangular Bézier surface equipped with a quadratic normal field[14]. Boubekeur and Schlick[15] proposed QAS to render Loop subdivision surfaces, where each patch is approximated by two quadratic triangular Bézier patches, for geometry and normal field, respectively. Amresh *et al.*[16] adopted Gregory patches[17] to approximate Loop subdivision surfaces. Due to the fact that the Loop subdivision surface is derived from the definition of quartic box spline, Li *et al.*[18] adopted quartic triangular Bézier patches to approximate the geometry of Loop subdivision surfaces, where each patch is obtained by interpolating 15 uniformly sampled points. To remedy the artifacts along the boundary of irregular patches, the normal field of the surface is also approximated via two quartic triangular Bézier patches.

Nießner *et al.*[8,19] proposed a fast and exact patching algorithm for rendering Catmull-Clark subdivision surfaces, which was recently extended to deal more effectively with semi-sharp creases[20]. First, irregular patches are refined adaptively by using a table-driven subdivision approach employing GPU compute kernels. Then all patches are evaluated and rendered using the GPU hardware tessellation unit. Furthermore, they presented a bitwise exact evaluation variant that is particularly useful for incorporating high-quality displacement mapping[21-22].

Table 1 shows an overview of properties of different Loop subdivision methods, including Shiue's approach[10], (uniform) global refinement (based on subdivision tables[8]) (Global), Stam's direct evaluation scheme[7], Li's approximate patching algorithm[18], and our method.

Table 1. Comparison of Different Loop Subdivision Rendering Techniques

	Shiue's	Global	Stam's	Li's	Ours
Speed	+	++	++	+++	+++
Memory	++	+	++	+++	+++
Accurate	Yes	Yes	Yes	No	Yes
PreSubd	Yes	No	Yes	No	No
Features	Yes	Yes	No	No	Yes

Note: the comparison includes speed, memory efficiency (Memory), accuracy (Accurate), requirement for an initial pre-subdivision step (PreSubd), and handling of features such as semi-sharp creases or hierarchical edits (Features).

2.2 Loop Subdivision Surfaces

A Loop subdivision surface takes a coarse triangular control mesh of arbitrary connectivity as input, as shown in Fig.1(a). In each level of refinement, two types of vertices are generated: vertex and edge points. Along with corresponding edges, they represent the refined mesh as shown in Fig.1(b). Both edge and vertex points are defined by linear combinations of vertices of the previous subdivision level. These linear combinations are referred to as the subdivision rules, or subdivision masks (each subdivision scheme has a different set of rules). The rules for Loop subdivision are depicted in Figs. 2(a) (vertex rule) and 2(b) (edge rule). Continuing the refinement process will result in a smooth limit surface that is C^2 everywhere except at extraordinary points where it is C^1. Vertex positions and normals on the limit surface can be directly obtained by applying the limit stencils as shown by Halstead *et al.*[23]. To further improve modeling flexibility, additional rules have been introduced: sharp creases[24], semi-sharp creases[25], and hierarchical edits[26].

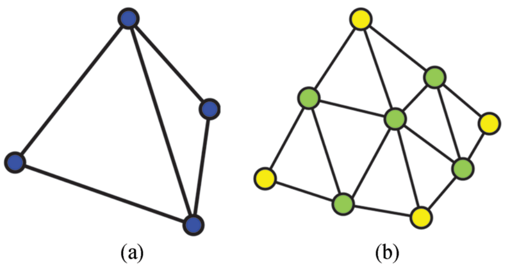


Fig.1. Loop subdivision of (a) a control mesh, resulting in (b) a refined mesh with vertex (yellow) and edge (blue) points being highlighted.

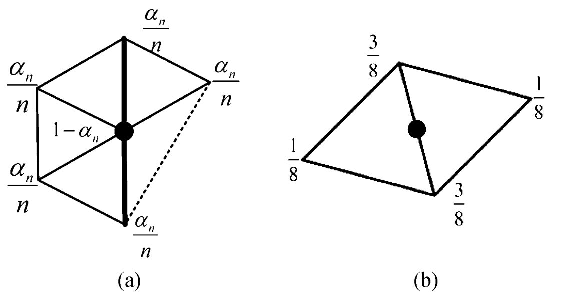


Fig.2. Loop subdivision masks. (a) Vertex point mask of an n-valence vertex, where $\alpha_n = \frac{5}{8} - (\frac{3}{8} + \frac{2}{8}\cos\frac{2\pi}{n})^2$. (b) Edge point mask.

3 Feature-Adaptive Rendering of Loop Subdivision Surfaces on GPU

Employing the Loop subdivision rules (cf. Subsection 2.2), a coarse control mesh can be refined as a series of fine triangular meshes recursively. As mentioned, the limit surface is composed of quartic box spline patches and globally C^2-continuous, except for the extraordinary points surrounded by an infinite number of quartic box spline patches, as shown in Fig.3(a). Semi-sharp creases behave similarly (see Fig.3(b)). This suggests to apply subdivision only where needed, i.e., in irregular regions which cannot be directly evaluated (regular patches can be evaluated according to the box spline definition). Features indicate a type of vertex which 1) is extraordinary, 2) belongs to a semi-sharp crease, or 3) is defined by a hierarchical edit. A patch is regular if its three vertices are regular (i.e., valence of 6). Otherwise, the patch is considered to be irregular.

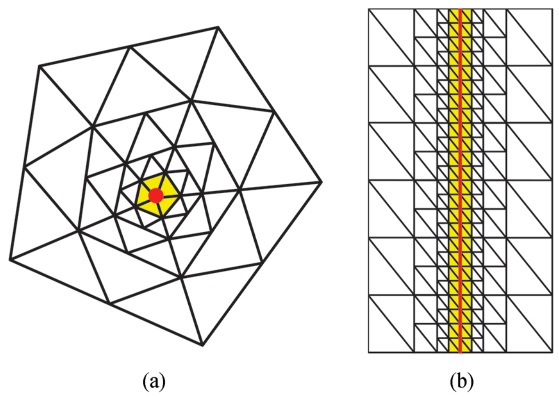


Fig.3. Arrangements of quartic box spline patches around (a) an extraordinary vertex and near (b) a semi-sharp crease. Irregular vertices and patches are colored in red and yellow, respectively.

An overview of our algorithm is shown in Fig.4. First, we generate subdivision tables for each level, corresponding to the configuration of the control net (cf. Fig.3). The tables are used to compute new vertices using a GPU compute shader running in parallel as described in Subsection 3.1. Then patches are constructed in accordance with the configuration of neighbor patches to eliminate T-junctions (see Subsection 3.2). Finally, the surface is rendered using hardware tessellation as shown in Subsection 3.3. In Subsection 3.4, we introduce a variant of patch tessellation evaluation to guarantee watertightness between adjacent patches. Finally, we propose a view-dependent LOD scheme in order to optimize for rendering quality and runtime performance (see Subsection 3.5).

3.1 Subdivision Table Generation and Parallel Subdivision

In order to apply the subdivision rules, we precompute subdivision tables, similar to Nießner *et al.*[8] The tables encode topological information which is re-

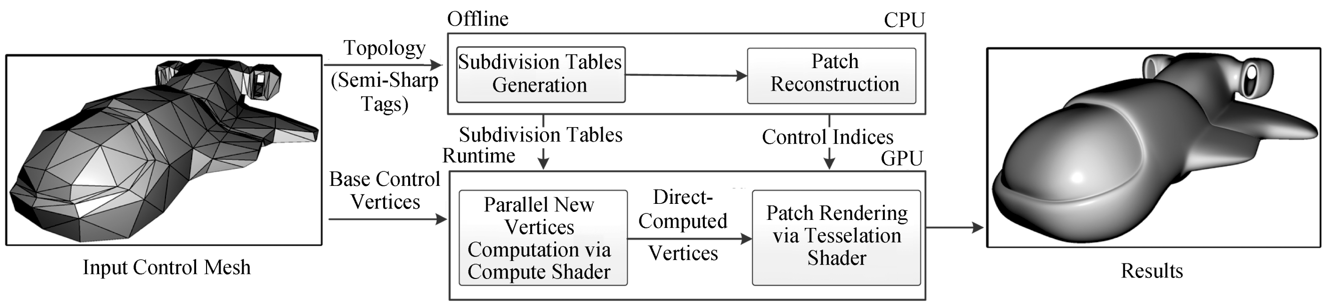


Fig.4. Framework overview of our proposed feature-adaptive rendering algorithm.

quired to evaluate edge and vertex points efficiently using a GPU compute kernel. In contrast to global refinement approaches (e.g., Shiue's algorithm[10]), we only subdivide adaptively around features. Hence, the amount of triangles generated near extraordinary vertices is only linear with respect to the subdivision level (rather than the exponential). This reduces memory storage and bandwidth significantly, thus enabling a faster surface evaluation. Since the topology of the control mesh is typically static during animation, modeling and rendering, the subdivision tables can be efficiently pre-computed on the CPU.

Once tables are generated, the GPU kernels employ the encoded subdivision rules in order to efficiently compute edge and vertex points of the next corresponding subdivision level. There are two types of kernels, which are executed for each subdivision level: edge point kernels and vertex point kernels. The edge point kernel requires an index buffer containing four indices per edge, referring to vertices of the two adjacent triangles. The vertex point kernel takes an index buffer with indices referring to all 1-ring neighbors for each vertex, as well as a vertex valence buffer. Note that we assign a single thread for each output vertex per kernel with all threads running in parallel. An example of subdivision tables for a pyramid control mesh is shown in Fig.5. If a patch contains sharp edges, we additionally store corresponding sharpness values. The subdivision tables also encode hierarchical surface edits.

3.2 Patch Reconstruction

After adaptive subdivision around irregularities, we end up with a nested set of regular patches. We render these patches with the GPU hardware tessellation unit where each patch is defined by 12 control points[7] (see Fig.6(a)). While the evaluation of regular patches is straightforward due to their box spline definition, we must eliminate potential T-junctions between adjacent patches of different subdivision levels. In contrast to the quad-based patching structure of Catmull-Clark surfaces, our approach must handle triangular Loop patches. To this end, two types of patches are defined: plain patches and non-plain patches. Our adaptive subdivision procedure ensures that two adjacent patches belong to either the same or the successive subdivision levels.

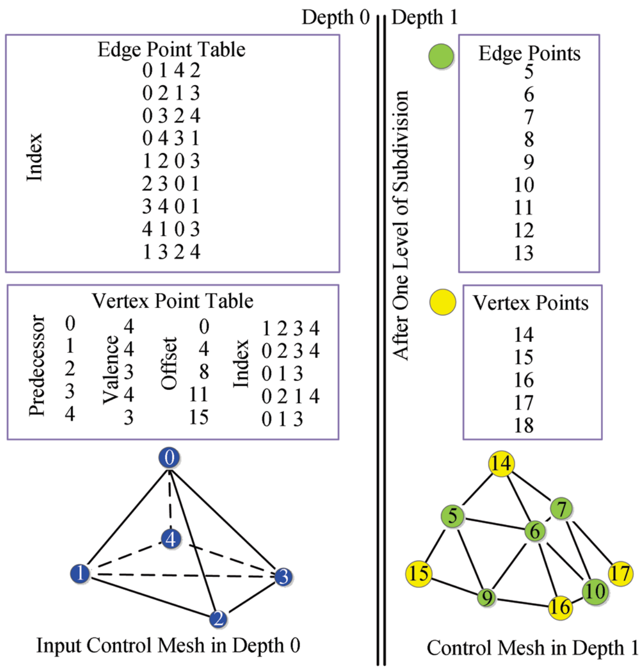
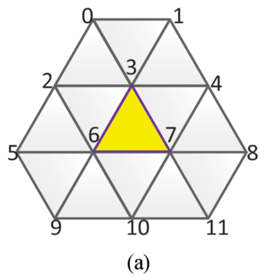


Fig.5. Example of partial subdivision tables for a pyramid-shaped control mesh. Vertex points are marked in green and edge points in yellow.

3.2.1 *Plain Patches*

A plain patch has only neighbors of the same or the next coarser subdivision level (but no neighbors of the finer level). We further classify plain patches into regular plain patches (RPPs) and irregular plain patches (IPPs), as shown in Fig.7. RPPs in each level will be directly rendered with the GPU tessellation unit, while IPPs will be refined in parallel until the maximum subdivision level. Only IPPs of the maximum level will be rendered. These are directly rendered as triangles without further tessellation, i.e., the tess factor (short for tessellation factor) is set to 1.0. For further details, see Subsection 3.3.



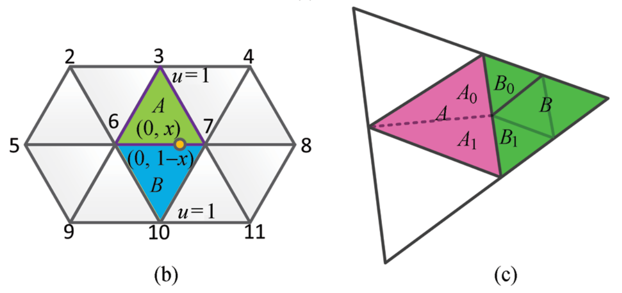


Fig.6. (a) A regular patch (yellow) with its 12 control points. (b) 10 control points involved in evaluating the shared edge of adjacent patches A and B. (c) An illustration of watertightness between two adjacent patches in different levels of subdivision.

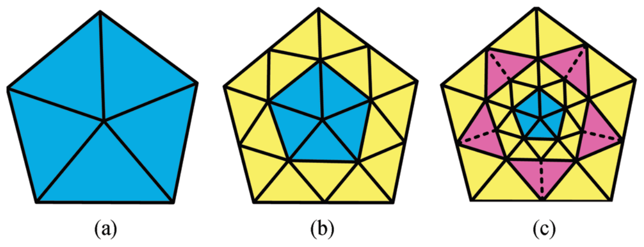


Fig.7. Patching for different subdivision levels: regular plain patches (RPPs) are marked in yellow, irregular plain patches (IPPs) in blue, and non-plain patches (NPPs) in pink. (a) Depth = 0. (b) Depth = 1. (c) Depth = 2.

3.2.2 *Non-Plain Patches*

Non-plain patches (NPPs) have at least one neighbor of the next finer subdivision level. By definition, all NPPs are regular (see above). The goal is to eliminate T-junctions between NPPs and neighboring patches of finer levels. A simple solution would be to adopt power-of-two tessellation factors in order to align tessellation points between different subdivision levels. However, that would severely limit level-of-detail possibilities. In order to allow for arbitrary edge tessellation factors, we split NPPs into sub-patches according to the configuration of its adjacent patches, as shown in Fig.8. Each sub-patch will share the same control net with its parent NPP. However, parametric sub-domains are different. This splits edges on patch boundaries of different subdivision levels and allows for the assignment of matching tessellation factors to shared edges. Thus, we are able to eliminate all T-junctions while not restricting edge tessellation factors.

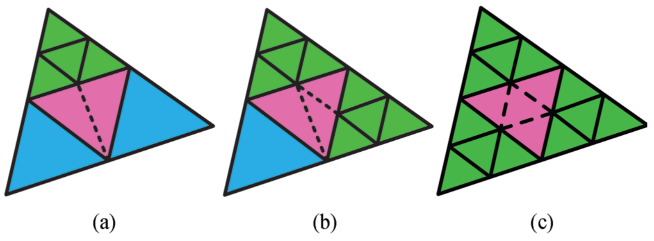


Fig.8. Three possible patterns to split an NPP into sub-patches: NPPs are marked in pink, patches of the current level in blue, and patches of the finer level in green. Note that we can apply different tessellation strategies for the sub-domains based on the triangulation patterns of the hardware tessellator (e.g., integer, fraction.).

3.3 Patch Rendering via Hardware Tessellation

The sampling density of the final rendered mesh is controlled by two tessellation parameters, i.e., a global tessellation factor *g_tessfactor* and a maximum adaptive subdivision depth *g_depth*. Since IPPs cannot be evaluated directly, *g_depth* sets the tessellation density at features such as extraordinary vertices. In order to obtain an even mesh tessellation, *g_depth* must correspond to *g_tessfactor*, i.e., $g\_depth = \lceil \log_2 g\_tessfactor \rceil$, where $\lceil x \rceil$ is the smallest integer not smaller than x. Let $tessfactor_i$ be the tessellation factor on subdivision level i, with $tessfactor_{i+1} = 0.5 \times tessfactor_i$ and $tessfactor_0 = g\_tessfactor$. For a given $g\_tessfactor$, we then perform $g\_depth = \lceil \log_2 g\_tessfactor \rceil$ refinement steps, thus enforcing $tessfactor_{g\_depth} = 1.0$. Hence, IPPs at the maximum subdivision level are only evaluated at corner points, allowing for accurate evaluation by applying the subdivision limit stencils[24]. Resulting IPP limit points are then simply rendered as standard triangles.

In order to evaluate regular patches (RPPs and NPPs), we adopt Stam's formulae[7]. That is, we use the implicit quartic box spline representation which allows for direct patch evaluation at arbitrary domain locations. Thus, regular patches can be efficiently processed by the GPU hardware tessellation unit, consequently, achieving high frame rates for surface rendering. We would also like to point out alternative evaluation strategies for regular patches[18,27].

3.4 Watertightness

In Subsection 3.2, we introduced a patching algorithm that enforces sampling points on shared edges to match on corresponding parameter locations. Thus, T-junctions between different subdivision levels are eliminated. In theory, that prevents cracks due to the underlying mathematical surface definition. However, in practice, floating point imprecisions may cause inconsistencies along patch boundaries. In particular, when adding surface displacements, mismatching surface normals cause problematic rendering artifacts. We now extend our algorithm, in order to overcome floating point limitations, and provide for bitwise matching vertex and normal evaluations along shared edges. We refer to this as watertight surface evaluation.

Following Nießner *et al.*[8], the key idea is to reformulate surface evaluations as reversal-invariant. That is, evaluations along a shared edge with domains u and $1-u$, respectively, must be identical. Therefore, floating point operations need to be commutative, which can be ensured by enabling IEEE floating point strictness when compiling shader programs. In the following, we first present a watertight evaluation procedure for patches that share the same subdivision level. We then introduce a method that handles patches with shared edges belonging to different subdivision levels.

3.4.1 Same Subdivision Level

In order to ensure watertight patch evaluation, we must guarantee bitwise-identical vertex positions, partial derivatives, and surface normal along shared boundaries. As described earlier, we evaluate regular patches employing the quartic box spline definition:

$$\begin{aligned}&\boldsymbol{S}(u,v)=\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}(u,v),(u,v)\in\Omega,\\&\frac{\partial\boldsymbol{S}}{\partial u}(u,v)=\boldsymbol{P}^{\mathrm{T}}\frac{d\boldsymbol{S}}{du}(u,v),\\&\frac{\partial\boldsymbol{S}}{\partial v}(u,v)=\boldsymbol{P}^{\mathrm{T}}\frac{d\boldsymbol{S}}{dv}(u,v),\end{aligned}$$

where $\boldsymbol{P}$ are the 12 control points of the quartic box spline patch $\boldsymbol{S}(u,v)$ as shown in Fig.6(a), and $\boldsymbol{S}(u,v)$ denotes 12 basis functions. The parameter domain is a "unit triangle" defined as $\Omega=\{(u,v)|u\in[0,1],v\in[0,1-u]\}$.

Cracks may occur along the shared boundary curve between two adjacent patches or at the shared corner, where the adjacent patches are evaluated independently. We differentiate between the boundary case and the corner case.

Boundary Case. Let A and B be the two adjacent patches, as shown in Fig.6(b). Without loss of generality, the shared boundary curves are the parameter line $u=0$ for both A and B. However, their parametric directions v are opposite. To resolve this crack problem, a bitwise-identical evaluation approach is designed by employing the symmetry of basis functions $\boldsymbol{B}(u,v)$[7] and the commutativity of floating point addition and multiplication. Thus, the shared boundaries of A and B are evaluated independently as follows:

$$\begin{aligned}\boldsymbol{S}_A(0,v)&=\sum_{i=2}^{11}\boldsymbol{p}_i^A\times b_i(0,v),\\\boldsymbol{S}_B(0,v)&=\sum_{i=2}^{11}\boldsymbol{p}_i^B\times b_i(0,v),\end{aligned}$$

where $\{\boldsymbol{p}_i^A\}$ and $\{\boldsymbol{p}_i^B\}$ are the control points of the patches A and B, respectively, and $b_i(u,v)$ is the i-th basis function of the quartic box spline. Note that $b_0(0,v)$ and $b_1(0,v)$ are both zero along the boundary curve. This leaves us with

$$\begin{aligned}\boldsymbol{S}_X(0,v)=&\boldsymbol{p}_2^Xb_2(0,v)+\boldsymbol{p}_3^Xb_3(0,v)+\boldsymbol{p}_4^Xb_4(0,v)+\\&\boldsymbol{p}_5^Xb_5(0,v)+\boldsymbol{p}_6^Xb_6(0,v)+\boldsymbol{p}_7^Xb_7(0,v)+\\&\boldsymbol{p}_8^Xb_8(0,v)+\boldsymbol{p}_9^Xb_9(0,v)+\boldsymbol{p}_{10}^Xb_{10}(0,v)+\\&\boldsymbol{p}_{11}^Xb_{11}(0,v),\\&X=A,B.\end{aligned}$$

Due to the symmetry of the basis functions, we obtain $b_i(0,v)=b_{13-i}(0,1-v)$, and $\boldsymbol{p}_i^B=\boldsymbol{p}_{13-i}^A$, $i=2,3,\ldots,11$. In order to enforce bitwise-identical results, we reformulate the evaluation order:

$$\begin{aligned}\boldsymbol{S}(0,v)=&[[\boldsymbol{p}_2b_2(0,v)+\boldsymbol{p}_{11}b_{11}(0,v)]+\\&[\boldsymbol{p}_3b_3(0,v)+\boldsymbol{p}_{10}b_{10}(0,v)]]+[[\boldsymbol{p}_4b_4(0,v)+\\&\boldsymbol{p}_9b_9(0,v)]+[\boldsymbol{p}_5b_5(0,v)+\boldsymbol{p}_8b_8(0,v)]]+\\&[\boldsymbol{p}_6b_6(0,v)+\boldsymbol{p}_7b_7(0,v)].\end{aligned}$$

Consequently, $\boldsymbol{S}_A(0,v)=\boldsymbol{S}_B(0,1-v)$, irrespective of the evaluation directly. Partial derivatives are processed similarly.

Bitwise-consistent normal evaluation along shared edges extends to three different edge configurations:

$$\boldsymbol{N}(u,v)=\begin{cases}\dfrac{\partial\boldsymbol{S}}{\partial u}(u,v)\times\dfrac{\partial\boldsymbol{S}}{\partial v}(u,v), & \text{if } u=0,\\\dfrac{\partial\boldsymbol{S}}{\partial v}(u,v)\times\dfrac{\partial\boldsymbol{S}}{\partial w}(u,v), & \text{if } v=0,\\\dfrac{\partial\boldsymbol{S}}{\partial w}(u,v)\times\dfrac{\partial\boldsymbol{S}}{\partial u}(u,v), & \text{if } 1-u-v=0.\end{cases}$$

Corner Case. A corner vertex ($u=1$, or $v=1$, or $u=v=0$) is evaluated according to its 1-ring neighbor vertices[24]. In order to achieve bitwise-identical corner evaluations, we enforce a consistent evaluation order. For each extraordinary vertex, we pick the neighbor with the smallest vertex ID. We then sum contributing terms of the linear combination of the limit stencils[23] by starting with term corresponding to the chosen vertex. Since this information is shared among all adjacent patches, we obtain the same evaluation order and thus bitwise-exact matches.

3.4.2 Different Subdivision Levels

Special attention must be paid to handle shared edges whose adjacent patches belong to different subdivision levels. As shown in Fig.6(c), B_0 and B_1 are child patches of B of the next finer level. A_0 and A_1 are two sub-patches of A (same subdivision level), as described in Subsections 3.2. The sample points on the shared edge of B_0 and A may not be bitwise-identical (the input control points are different), therefore we evaluate these patches with the (exact same) control points of A and B_0. Therefore, when evaluating B_0, we use the 12 control points of B_0 for the interior and the 10 control points of A for boundary curve of B_0 (these belong to the coarser subdivision level). Note that B is irregular; thus, we cannot use its own control points. As a result, the boundary curves on B_0 and A are both evaluated with the same control points of A. When following the same evaluation rules as introduced in Subsection 3.4.1, the rendered surface is watertight.

While watertight surface evaluation provides high-quality tessellation results, it introduces code branching and requires additional adjacency information. This leads to an increase of render time as discussed in Subsection 4.6.

3.5 View-Dependent Level-of-Detail Rendering

As introduced in Subsection 3.3, *g_tessfactor* is set in order to specify the tessellation density. The maximum subdivision level *g_depth* follows accordingly: $\lceil \log_2 g\_tessfactor \rceil$. Similar to the view-dependent rendering by Nießner *et al.*[8], we introduce a level-of-detail scheme for Loop subdivision surfaces.

Adaptive Tessellation Factors. We determine tess factors on-the-fly by employing different metrics for adaptive surface tessellations[28]. One way is to assign a global *g_tessfactor* for the entire surface. As the implemented result in Fig.9 shows, we simply use the model's centroid and set *g_tessfactor* according to the distance to the camera. Alternatively, we compute local tess factors, based on screen space edge length measures. That is, we assign separate tess factors to each patch edge, thus allowing for optimal tessellations. Note that in this way, shared edges receive matching tess factors, which is a necessity for crack-free surface rendering. While this provides good results in practice, more elaborate LOD metrics are feasible at the cost of additional computations[29].

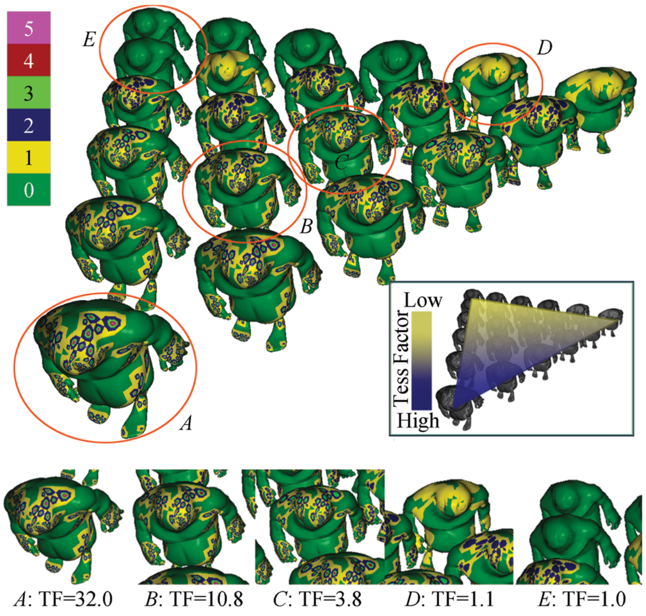


Fig.9. Level-of-detail rendering: colors identify different subdivision levels as stated in the top left corner.

Adaptive Maximum Subdivision Depth. By default, the maximum subdivision depth is set to $\lceil \log_2 g\_tessfactor \rceil$. However, when enabling adaptive tessellation, we also dynamically determine the subdivision depth. In order to prevent irregular patches at the finest level from having tess factors greater than 1.0, we enforce *g_tessfactor* to be smaller than or equal to 2^{g_depth}. Since we can easily compute a good estimate for *g_depth*, this imposes no limitations.

4 Results and Applications

We have implemented our approach using DirectCompute for the GPU subdivision kernels and Direct3D 11[②] to access the hardware tessellator. All GPU code is written in HLSL. For our experiments, we use a PC with a 2.80 GHz Intel Core i5-2300 CPU and an NVIDIA GeForce GTX 570 GPU running Windows 7.

In this section we show examples and results of our approach of handling features of Loop subdivision surfaces, such as semi-sharp creases, displacement mapping, and hierarchical editing. Extraordinary vertices are also considered to be features since adjacent patches cannot be directly evaluated. Rendering results generated by our algorithm are shown in Fig.10. Corresponding performance measurements are listed in Table 2. All timings are provided in milliseconds and account for all runtime overhead, including GPU subdivisions with compute shader and final rendering with the hardware tessellator. Memory requirements for our test models are shown in Fig.11(a).

②http://msdn.microsoft.com/en-us/library/ff476342(VS.85).aspx, Sept. 2014.

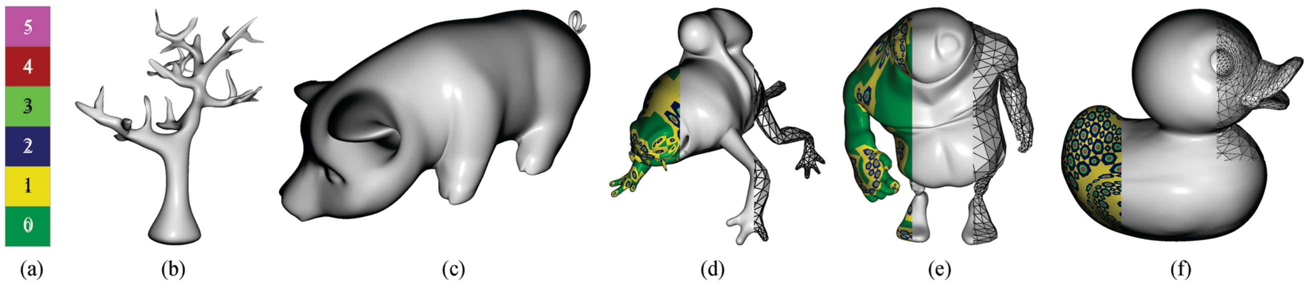


Fig.10. Rendering the true loop subdivision surface with our method. (a) Color table indicating the different subdivision levels. (b) Tree. (c) Pig. (d) Monster Frog. (e) Big Guy. (f) Duck. In these examples, we use a global tess factor of 32.

Table 2. Rendering Performance for Our Test Models Using Different Tessellation Densities

Model	Number of Triangles (k)	Tessellation Factor	Subdivision Depth	Performance (ms)
Tree	0.5	1	0	0.01
		2	1	0.12
		4	2	0.22
		8	3	0.40
		16	4	1.00
		32	5	3.59
Pig	1.2	1	0	0.03
		2	1	0.12
		4	2	0.24
		8	3	0.45
		16	4	1.08
		32	5	3.45
MF	1.2	1	0	0.03
		2	1	0.16
		4	2	0.33
		8	3	0.71
		16	4	2.13
		32	5	8.19
BG	2.9	1	0	0.04
		2	1	0.17
		4	2	0.33
		8	3	0.73
		16	4	2.11
		32	5	8.15
Duck	4.2	1	0	0.05
		2	1	0.22
		4	2	0.66
		8	3	1.39
		16	4	3.61
		32	5	12.66

Note: The adaptive subdivision depth corresponds to the respective tessellation factors (i.e., $g\_depth = \lceil \log_2 g\_tessfactor \rceil$). MF: Monster Frog, BG: Big Guy.

We also list the number for IPPs, RPPs, and NPPs in Table 3. The amount of sub-patches can be deduced from the number of irregular vertices and the subdivision depth. Note that since we only adaptively subdivide, patch growth is linear rather than exponential.

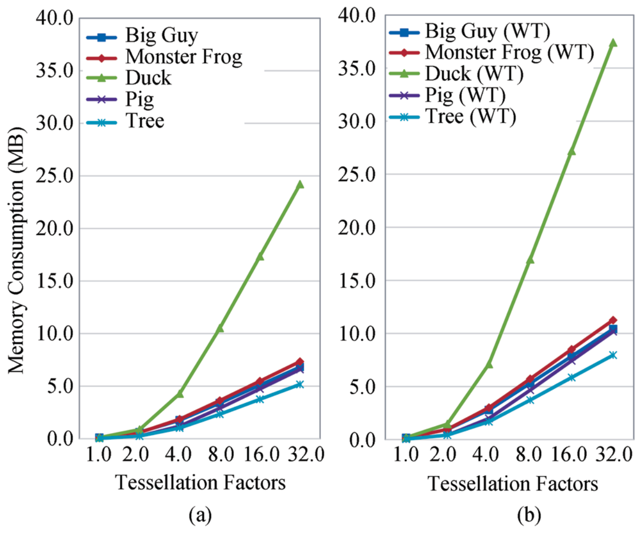


Fig.11. (a) Memory consumption without watertight rendering. (b) Memory consumption with watertight rendering.

4.1 Semi-Sharp Creases

Our algorithm efficiently handles semi-sharp creases, which are widely used in character modeling applications[25]. Therefore, the subdivision tables encode additional sharpness attributes. These are used by the subdivision kernels to apply the sharp subdivision rules[24] if necessary. Examples of models with semi-sharp creases of varying sharpness parameters are shown in Fig.12.

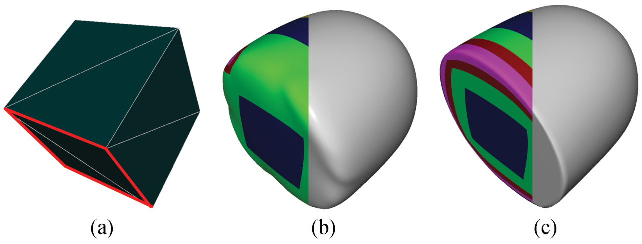


Fig.12. (a) Input control mesh with semi-sharp creases marked in red. (b) Semi-sharp edges are assigned a sharpness of 3.5. (c) Semi-sharp edges are assigned a sharpness of 10.

Table 3. Patch Count by Sub-Patch Types for Our Test Models Big Guy (BG) and Monster Frog (MF) at Each Subdivision Level

Model	Sub-Path Type	Tessellation Factor	Subdivision Depth	Patch Count (k)
BG	IPP	1	0	1.2
		2	1	1.8
		4	2	1.8
		8	3	1.8
		16	4	1.8
		32	5	1.8
	RPP	1	0	1.1
		2	1	3.0
		4	2	6.7
		8	3	10.0
		16	4	14.0
		32	5	17.0
	NPP	1	0	0.5
		2	1	1.7
		4	2	3.6
		8	3	5.4
		16	4	7.3
		32	5	9.2
MF	IPP	1	0	1.4
		2	1	2.0
		4	2	2.0
		8	3	2.0
		16	4	2.0
		32	5	2.0
	RPP	1	0	0.6
		2	1	2.9
		4	2	6.9
		8	3	11.0
		16	4	15.0
		32	5	19.0
	NPP	1	0	0.4
		2	1	1.9
		4	2	3.9
		8	3	5.9
		16	4	7.9
		32	5	9.9

4.2 Displacement Mapping

Displacement mapping[21-22] is an efficient technique to add fine-scale geometric detail to a coarse base surface. In particular, scalar-valued displacements are widely used in order to minimize storage and memory bandwidth. We apply displacements to a Loop subdivision surface, which can be efficiently rendered using our feature-adaptive approach. Fig.13 shows an example of the displaced Monster Frog model.

4.3 Hierarchical Editing

Hierarchical surface edits[26] are used to add local surface detail by modifying vertex positions on specific subdivision levels. Our approach can efficiently handle hierarchical edits by adaptive subdivision. That is, we tag vertices with hierarchical edits as irregular and encode the edits in the subdivision tables. An example with hierarchical surface edits is shown in Fig.14. Note that once the locations of edits are specified, we can dynamically change their values at runtime.

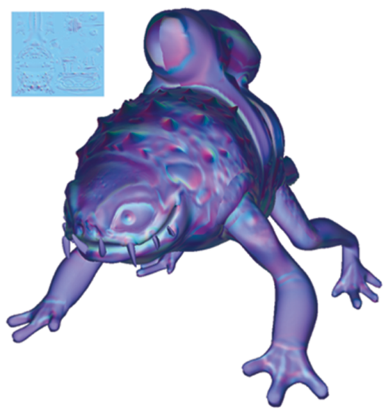


Fig.13. Applying displacement mapping using our algorithm on the Monster Frog model.

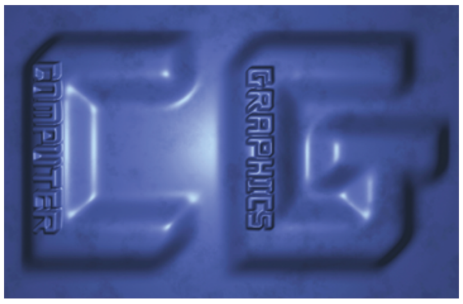


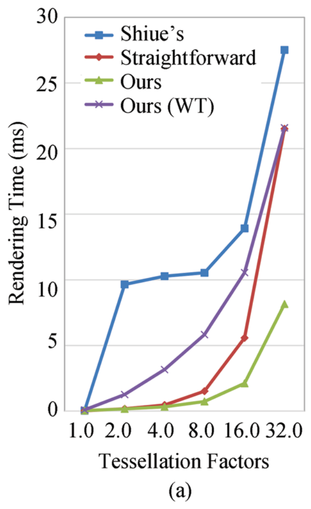
Fig.14. Test model with hierarchical surface edits rendered with our approach.

4.4 Comparison with Global Refinement Approaches

Global refinement methods have been proposed before the introduction of the GPU hardware tessellator. These approaches require mesh vertices to be streamed to and from multiprocessors every subdivision step, thus causing a large amount of memory I/O. Our method avoids this limitation, by only subdividing a minimal amount of patches and evaluating the vast majority of patches with the hardware tessellation on-chip.

We compare our approach (w/ and w/o watertightness) to two global refinement approaches. One is a DirectCompute version of Shiue's method[10]; the other is a straightforward global refinement method based on subdivision tables (see Subsection 3.1). Note that

Shiue's method requires one level of pre-refinement. As shown in Figs. 15 and 16, our method (without watertightness) is over 3 times faster and consumes 80% less memory than Shiue's algorithm, and is 1.3 times faster and consumes 50% less memory than the straightforward approach.



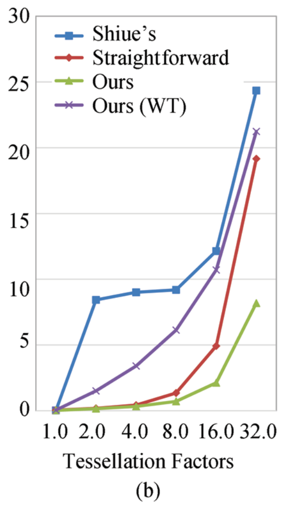
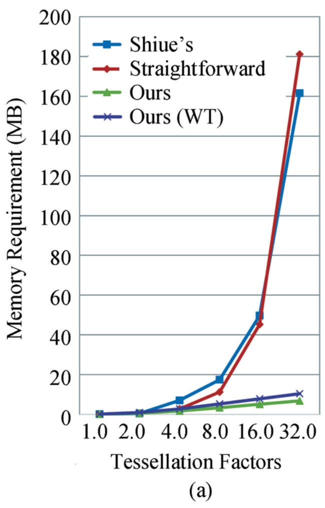


Fig.15. Rendering time for Shiue's method, the straightforward method, and ours without and with watertightness (WT). (a) Big Guy model. (b) Monster Frog model.



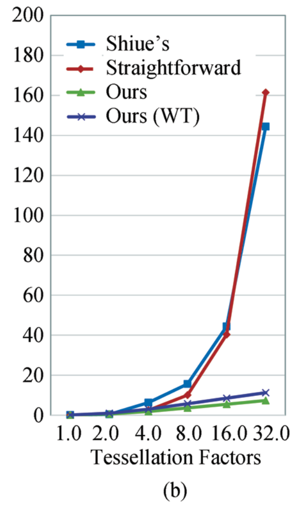


Fig.16. Comparisons of GPU memory requirement between Shiue's method, the straightforward global refinement method, and ours without and with watertightness (WT). (a) Big Guy model. (b) Monster Frog model.

4.5 Comparisons with Approximate Patching and Stam Evaluation

As mentioned in Section 2, approximate patching algorithms can be used to adopt hardware tessellation in order to render subdivision surfaces. They achieve high render performance, however, at the cost of rendering quality. First, we compare our method against Li's, as shown in Fig.17. While our method renders the globally smooth Loop limit surface, the approximation introduces surface discontinuities, which cause clearly visible rendering artifacts.

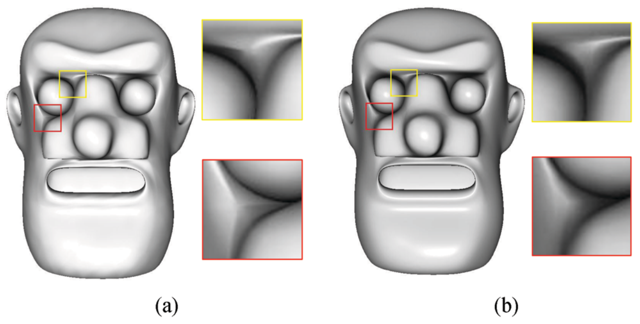
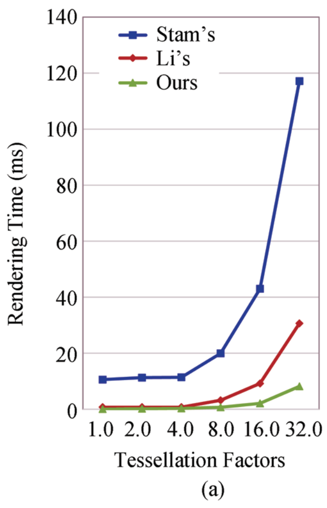


Fig.17. Qualitative comparison between (a) Li's approximate patching algorithm and (b) ours on the Head model.

Since Stam's algorithm requires extraordinary vertices to be isolated, one level of pre-refinements must be applied. Note that our approach does not have this limitation. Further, a GPU implementation of Stam's algorithm involves a significant amount of floating point operations and code branching, which affects the overall render performance. Fig.18 shows a comparison among our method, Li's and Stam evaluation. Our algorithm is the most efficient. It is even faster than the approximate method by Li. We attribute this to additional patch setup costs of Li's algorithm that particularly slow down hull shader execution.



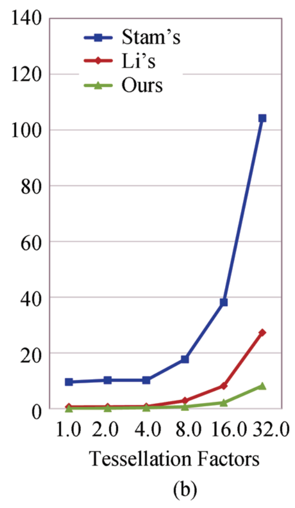


Fig.18. Rendering comparisons among Stam evaluation, Li's method, and ours. (a) Big Guy model. (b) Monster Frog model.

4.6 Watertight Rendering

In this subsection, we provide a comparison between the watertight (see Subsection 3.4) and the default variant of our method.

In general, watertight evaluations consume more memory, since adjacency information is required. Evaluations are also more expensive since additional code branching is introduced. In order to confirm bitwise watertightness, we stream the tessellated geometry to the CPU where we verify the results.

Rendering time and memory requirements of both variants are shown in Fig.15 and Fig.11. Overall, the watertight version of our algorithm is approximately 80% slower and consumes 50% more memory than the native approach.

5 Conclusions

In this paper, we presented a novel approach to accurately render Loop subdivision surfaces. Our algorithm design fully exploits the power of modern GPUs, thus achieving real-time frame rates. Since we only refine patches at irregularities, memory consumption and bandwidth is kept at a minimum. In contrast to many other approaches, we do not require any pre-refinement, meaning that our method can directly process triangular meshes of arbitrary connectivity. In addition, we presented a (bitwise) watertight evaluation approach, allowing for crack-free rendering even with displacements. We also efficiently handle Loop subdivision features, e.g., semi-sharp creases and hierarchical edits. To conclude, results show that our algorithm facilitates both time- and memory-efficient rendering. This makes our method ideally suited to real-time applications such as video games and interactive authoring tools.

References

[1] Catmull E, Clark J. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 1978, 10(6): 350-355.

[2] Doo D, Sabin M. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 1978, 10(6): 356-360.

[3] Loop C. Smooth subdivision surfaces based on triangles [Master Thesis]. Department of Mathematics, University of Utah, 1987.

[4] Zorin D, Schröder P, DeRose T, Stam J, Kobbelt L. Subdivision for modeling and animation. In *Proc. ACM SIGGRAPH 99 Course Notes*, August 1999.

[5] Schäfer H, Nießner M, Keinert B *et al.* State of the art report on real-time rendering with hardware tessellation. In *Eurographics 2014 — State of the Art Reports*, April 2014, pp.93-117.

[6] Stam J. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Proc. the 25th Annual Conference on Computer Graphics and Interactive Techniques*, July 1998, pp.395-404.

[7] Stam J. Evaluation of Loop subdivision surfaces. In *Proc. ACM SIGGRAPH 99 Course Notes*, August 1999.

[8] Nießner M, Loop C, Meyer M *et al.* Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces. *ACM Transactions on Graphics*, 2012, 31(1): Article No. 6.

[9] Pulli K, Segal M. Fast rendering of subdivision surfaces. In *Proc. the 7th Eurographics Workshop on Rendering Techniques*, 1996, pp.61-70.

[10] Shiue L, Jones I, Peters J. A realtime GPU subdivision kernel. *ACM Transactions on Graphics*, 2005, 24(3): 1010-1015.

[11] Kim M, Peters J. Realtime Loop subdivision on the GPU. In *Proc. ACM SIGGRAPH 2005*, July 2005, Article No. 123.

[12] Boubekeur T, Schlick C. A flexible kernel for adaptive mesh refinement on GPU. *Computer Graphics Forum*, 2008, 27(1): 102-114.

[13] Bischoff S, Kobbelt L P, Seidel H P. Towards hardware implementation of loop subdivision. In *Proc. the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, Aug. 2000, pp.41-50.

[14] Vlachos A, Peters J, Boyd C *et al.* Curved PN triangles. In *Proc. the 2001 Symposium on Interactive 3D Graphics*, Mar. 2001, pp.159-166.

[15] Boubekeur T, Schlick C. QAS: Real-time quadratic approximation of subdivision surfaces. In *Proc. the 15th Pacific Conference on Computer Graphics and Applications*, Oct. 29-Nov. 2, 2007, pp.453-456.

[16] Amresh A, Femiani J, Fünfzig C. Methods for approximating Loop subdivision using tessellation enabled GPUs. *Advances in Visual Computing*, 2012, 7431: 115-125.

[17] Loop C, Schaefer S, Ni T *et al.* Approximating subdivision surfaces with Gregory patches for tessellation hardware. *ACM Transactions on Graphics*, 2009, 28(5): Article No. 151.

[18] Li G, Ren C, Zhang J *et al.* Approximation of Loop subdivision surfaces for fast rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2011, 17(4): 500-514.

[19] Nießner M. Rendering subdivision surfaces using hardware tessellation [Ph.D. Thesis]. Department of Computer Science, University of Erlangen-Nuremberg, 2013.

[20] Nießner M, Loop C, Greiner G. Efficient evaluation of semismooth creases in Catmull-Clark subdivision surfaces. In *Proc. Eurographics (Short Papers)*, May 2012, pp.41-44.

[21] Szirmay-Kalos L, Umenhoffer T. Displacement mapping on the GPU-State of the art. *Computer Graphics Forum*, 2008, 27(6): 1567-1592.

[22] Nießner M, Loop C. Analytic displacement mapping using hardware tessellation. *ACM Transactions on Graphics*, 2013, 32(3): Article No. 26.

[23] Halstead M, Kass M, DeRose T. Efficient, fair interpolation using Catmull-Clark surfaces. In *Proc. the 20th Annual Conference on Computer Graphics and Interactive Techniques*, Sept. 1993, pp.35-44.

[24] Hoppe H, DeRose T, Duchamp T *et al.* Piecewise smooth surface reconstruction. In *Proc. the 21st Annual Conference on Computer Graphics and Interactive Techniques*, July 1994, pp.295-302.

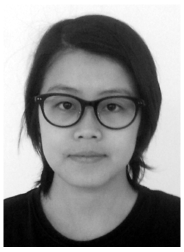
[25] DeRose T, Kass M, Truong T. Subdivision surfaces in character animation. In *Proc. the 25th Annual Conference on Computer Graphics and Interactive Techniques*, July 1998, pp.85-94.

[26] Forsey D R, Bartels R H. Hierarchical B-spline refinement. In *Proc. the 15th Annual Conference on Computer Graphics and Interactive Techniques*, Aug. 1988, pp.205-212.

[27] Peters J. Smooth patching of refined triangulations. *ACM Transactions on Graphics*, 2001, 20(1): 1-9.

[28] Zheng J, Sederberg T W. Estimating tessellation parameter intervals for rational curves and surfaces. *ACM Transactions on Graphics*, 2000, 19(1): 56-77.

[29] Yeo Y I, Bin L, Peters J. Efficient pixel-accurate rendering of curved surfaces. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Mar. 2012, pp.165-174.

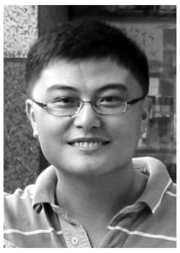


Yun-Cen Huang obtained her B.S. degree in computer science and technology from Fuzhou University in 2010. She is currently a Ph.D. candidate in the State Key Lab of CAD & CG, Zhejiang University, Hangzhou. Her current research interests include displacement mapping, GPU rendering, and geometry modeling.

Jie-Qing Feng is a professor in the State Key Lab of CAD & CG, Zhejiang University, Hangzhou. He received his B.S. degree in applied mathematics from the National University of Defense Technology in 1992 and his Ph.D. degree in computer graphics from Zhejiang University in 1997. His research interests include mannequin modeling, real-time rendering and deformations in animation.

Matthias Nießner is a visiting assistant professor in Pat Hanrahan's group at Stanford University. In 2013, he received his Ph.D. degree in computer science from the University of Erlangen-Nuremberg, Germany, under the supervision of Günther Greiner. His research is focused on different fields of computer graphics and computer vision, including real-time rendering, reconstruction of 3D scene environments, and semantic scene understanding.

Yuan-Min Cui obtained his B.S. degree in computer science from Zhejiang University in 2007. He is currently a Ph.D. candidate in the State Key Lab of CAD & CG, Zhejiang University, Hangzhou. His current research interests include free-form deformation and GPU parallel computing.



Baoguang Yang obtained his M.S. degree in computer science and technology from Zhejiang University in 2006. His current research interests include rendering.