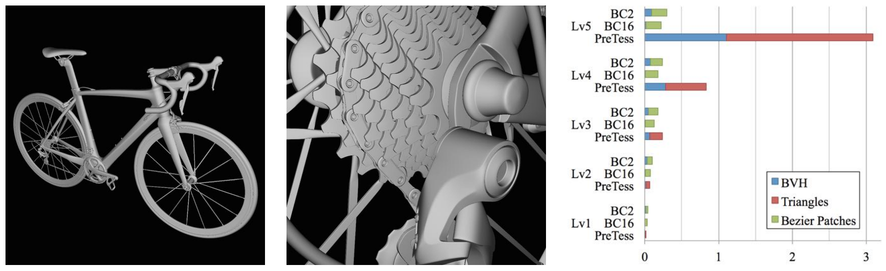
Direct Ray Tracing of Full-Featured Subdivision Surfaces with Bézier Clipping

Takahito Tejima
Pixar Animation Studios

Masahiro Fujita
Light Transport Entertainment

Toru Matsuoka
DeNA Inc.



(a) Direct ray trace rendering (b) Close view of (a) (c) Memory usage (GB)

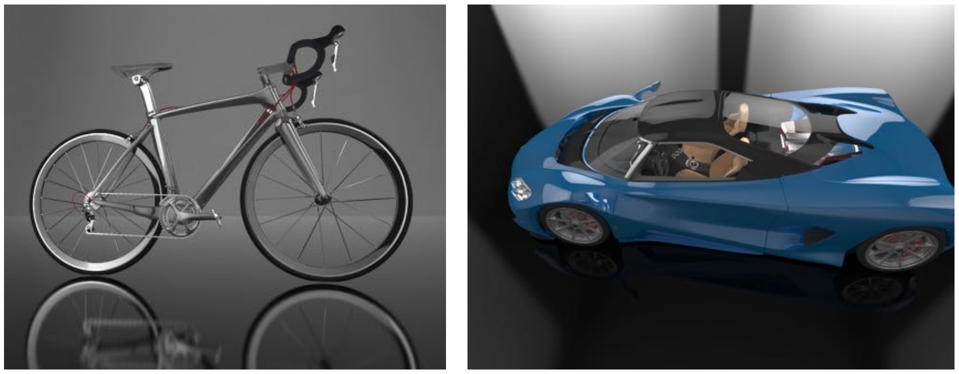
Figure 1. (a) A result of ray casting using our algorithm. 1.1M patches at 1K x 1K resolution, rendered in 0.218s with 305.2MB of memory. (b) Close view of the same model in 1.226 s. (c) Memory usage of PreTess:pre-tessellation, BC2:Bézier clipping method with 2 patches in leaf nodes and BC16:Bézier clipping method with 16 patches in leaf nodes.

Abstract

We present a novel, direct ray-tracing method for rendering Catmull-Clark subdivision surfaces. Our method resolves an intersection with Bézier patches without performing of polygon tessellation as done in most traditional renderers. It supports the full set of RenderMan and OpenSubdiv subdivision surface features, including hierarchical edits, creases, semi-sharp creases, corners, the chaikin rule, and holes.

Our method is fast (compared to traditional CPU ray tracers), robust and crack-free, has very low memory requirements, and minimizes BVH construction and traversal overhead for high subdivision levels. It computes ray-surface intersection using Bézier clipping on patches obtained from the full-featured subdivision surfaces.

In this paper, we explain our implementation and evaluate its performance and memory characteristics for primary ray casting. We further show examples of indirect illumination using secondary ray casting and note that because the method does not require view-dependent pre-tessellation, may offer further advantages in that case.



(a) Rendering example of the bike model (b) Rendering example of a car model. Isolation level 5, 2.2M patches, 597MB

Figure 2. Global illumination renderings by tracing secondary rays using our method.

1. Introduction

Subdivision surfaces are smooth surfaces obtained by recursively subdividing a control mesh. Modern subdivision surfaces contain many *features* necessary for authoring complex shapes. It is essential that any production renderer support the full set of such features, as defined by the RenderMan [Pixar Animation Studios 2005] specification and its open equivalent, OpenSubdiv [OpenSubdiv]. These features include the semi-sharp crease introduced by Derose et al. [1998] and the Forsey-Bartels [1988] hierarchical representation. A full list of these features is found in the OpenSubdiv online manual Subdivision Surfaces page, `http://graphics.pixar.com/opensubdiv/docs/subdivision_surfaces.html`.

We propose an efficient direct ray tracing method for feature adaptive Catmull-Clark subdivision surfaces using fast and robust Bézier patch intersection. Our method gives the intersection on the limit position of the subdivision surface, without having densely tessellated quads or triangles. It significantly reduces the memory footprint for both BVH nodes and geometry. Also, we address the crack issues on the patch boundaries which arise from non-uniform patch conformation and floating point computation accuracy.

An intersection algorithm is not only for rendering. It can also be used for global illumination (shadow and secondary ray) effect, collision detection, cloth simulation, texture painting tools, etc.

The Catmull-Clark subdivision method is the most common one employed in modeling and rendering software packages, so we focus on it in this paper.

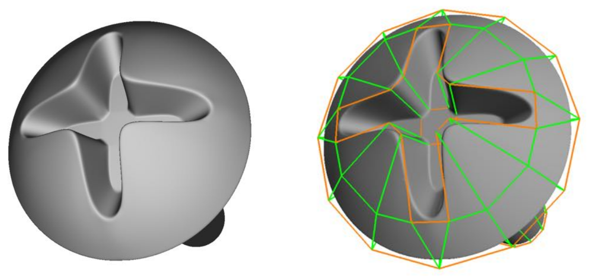


Figure 3. (a) A screw modeled by 84 faces with creases. (b) The control cage of the screw and GL rendering by OpenSubdiv [OpenSubdiv].

2. Related Work

We classify ray tracing methods for subdivision surfaces into two categories, a) Pre tessellation and b) Direct intersection, following Müller et al. [2003]. Pre-tessellation techniques are usually preferred, because they are simple and fast. Christensen et al. [2003] proposed using ray differentials to construct multiresolution caches for tessellated faces.

Kobbelt et al.'s [1998] direct intersection method introduced a bounding envelope technique for intersecting a ray with a subdivision surface. Benthin et al. [2007] proposed the efficient direct rendering of subdivision surfaces using packet ray tracing. Their method recursively subdivides a patch for a ray packet and the final intersection is computed by approximating patches with polygons.

More recently, full-featured subdivision surface rendering using GPU has been proposed [Nießner et al. 2012a; Nießner et al. 2012b]. They introduced feature adaptive analysis, which selectively refines around creases, corners, hierarchical edits and extraordinary vertices, and then represents a subdivision surface as a set of bicubic B-spline patches. We notice that this analysis is also useful for ray-tracing, since B-spline patch can be easily converted into Bézier patch, and direct Bézier patch intersection methods for ray tracing have already been studied, using the Newton method [Martin et al. 2000] and Bézier clipping [Efremov et al. 2005].

3. Our Approach

Our algorithm is composed of the following four stages: 1) Generate B-spline patches by feature adaptive subdivision. 2) Convert B-spline patches to Bézier patches. 3) Construct BVH using the bound of generated Bézier patch. 4) Compute Bézier - ray patch intersection using robust and crack-free Bézier clipping.

Algorithm 1 Our Bézier - ray intersection algorithm.

```
1: function BEZIERPATCHCLIP(patch, level, clipDir)
2:     rotPatch ← XYROTATE(patch, clipDir)
3:     if MAXEXTENT($rotPatch$) $\leq \epsilon$ or $level \geq kMaxLevel$ then
4:         return BILINEARPATCHINTERSECT(patch)
5:     else
6:         $(r_{min}, r_{max})$ ← BEZIERCLIPRANGE(rotPatch, clipDir)
7:         $nextClipDir$ ← clipDir == U ? V : U
8:         $nextLv$ ← clipDir == V ? level + 1 : level
9:         if $(r_{max} - r_{min}) \geq 0.5$ then
10:            $patch0$ ←BEZIERPATCHCROP(patch, clipDir, 0, $\frac{1}{2}$)
11:            $patch1$ ←BEZIERPATCHCROP(patch, clipDir, $\frac{1}{2}$, 1)
12:            $hit0$ ←BEZIERPATCHCLIP(patch0, nextLv, nextClipDir)
13:            $hit1$ ←BEZIERPATCHCLIP(patch1, nextLv, nextClipDir)
14:            return NEAREST(hit0, hit1)
15:        else
16:            $patch$ ← BEZIERPATCHCROP(patch, clipDir, $r_{min}, r_{max}$)
17:            return BEZIERPATCHCLIP(patch, nextLv, nextClipDir)
18:
19: function BEZIERISECT(patch, ray)
20:     $patch$ ← ZALIGN(patch, ray)
21:     return BEZIERPATCHCLIP(patch, 0, U)
```

3.1. Generate B-spline Patches by Feature-Adaptive Subdivision

We follow Nießner et al. [2012a] for feature adaptive tessellation of Catmull-Clark subdivision surfaces. Firstly, the method builds adjacency structure and a feature adaptive mesh representation using half-edge algorithm. A face is classified as an irregular patch if it contains extraordinary vertices with degree other than four, if its edges or vertices have creases, or if it contains hierarchical edits. Irregular patches are further subdivided until a user-specified subdivision level has been reached or they become regular (Fig. 5 left). We also ensure that patches contain a single crease. We do this by applying direct semi-smooth crease evaluation on the patch instead of iteratively subdividing to its sharpness level [Nießner et al. 2012b]. Single-crease patches suppress the total number of patches to be generated (Fig. 4). All of this work is done during a CPU preprocessing phase. It does not require re-computation unless the underlying topology changes.

As a result of the feature adaptive analysis, B-spline patches are generated for regular patches, and Gregory patches are generated for the remaining irregular patches. We translated them into Bézier patches for intersection calculations. Single-crease patches can be split into two or three Bézier patches depending on whether the sharpness is integer or fractional, as described in Nießner et al. [2012b]. We average inner control points of Gregory patches to replace with Bézier patches, assuming irregular patches are sufficiently isolated. Their surface derivatives can be evaluated with Gregory basis, once a surface parameter of the intersection has been determined.

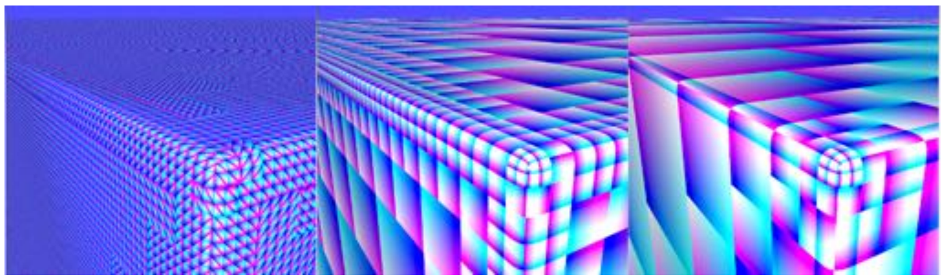


Figure 4. Comparison of the patch conformation around the edge with sharpness 8.0. Left: Dense quads from uniform subdivision at level 8. Center: Patches from feature adaptive subdivision. Right: Patches from feature adaptive subdivision with single crease patch enabled.

3.2. Convert B-spline Patches to Bézier Patches

When converting B-spline control points into Bézier control points, we use bireversal-invariant evaluation in a similar way as shown in Nießner et al. [2012a]. It produces bitwise identical results regardless of the uv direction of the base patches, which we confirmed by an exhaustive search of all representable floating point values on $[0, 1)$. Nießner et al. [2012a] also showed that for the patches generated by different subdivision levels that also share a boundary, the parent control points should be used to ensure watertight evaluation.

We set a flag on the patch generated to classify it as a *transition patch* (a patch that shares a boundary with patches of different subdivision level [Nießner et al. 2012a]). This flag is used in **conservative intersection algorithm** to prevent cracks (holes), which will be described in detail in Section 3.4.3.

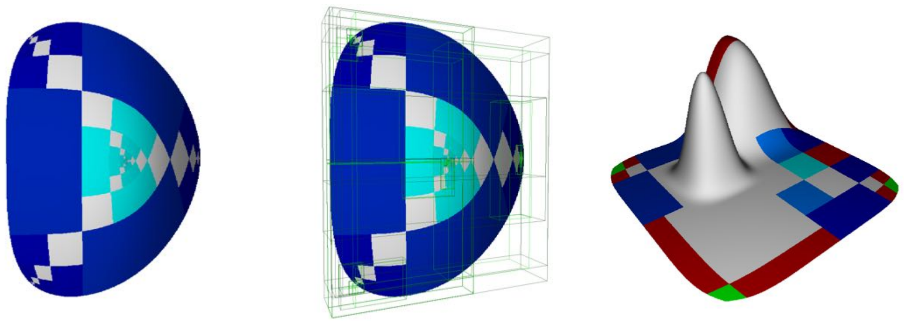


Figure 5. Left: Feature adaptive subdivision. Each patch has different colors. Middle: Its corresponding BVH visualization. Right: Hierarchical edits are also supported (From an OpenSubdiv example).

3.3. Construct the BVH from Generated Bézier Patch Bounds

Next, we construct an axis-aligned BVH (Bounding Volume Hierarchy) [Rubin and Whitted 1980] for the Bézier patches generated in Section 3.1. Thanks to the convex property of a Bézier surface, we only need to compute the bounding box enclosing the control points of Bézier patches. See Fig. 5 middle.

3.4. Compute Bézier Patch Intersection

Because we are intersecting rays against bicubic Bézier patches, many new challenges arise that are not typically addressed in traditional bilinear triangle intersection methods.

We use Bézier clipping, since Newton iteration has an initialization problem known as the *initial guess*. Firstly, we transform the patch to align the local Z-axis to the ray direction in order to gain the numerical efficiency and stability. Then we rotate the patch on the XY-plane at each clipping iteration to determine the clipping range more efficiently [Efremov et al. 2005]. Note that the rotated patch is used only for range determination. Bézier clipping is always applied to the ray aligned patch to avoid floating point errors. Once the clipping iteration converged, our method approximates the remaining small patch with a bilinear patch. The intersection of the bilinear patch can be numerically found in a robust way [Ramsey et al. 2004]. See Algorithm 1 for details.

We further added the following extension to ensure crack-free intersections across patch boundaries.

3.4.1. *Bézier Cropping*

Our method crops a patch from domain $([0, 1], [0, 1])$ to $([u0, u1], [v0, v1])$ for each iteration of Bézier clipping. Since feature adaptive analysis of subdivision surfaces produces arbitrarily oriented Bézier patches, and IEEE 754 floating point addition and multiplication are neither associative nor distributive, we also need to modify the cropping operation for reversal-invariance (Fig. 6).

Although exhaustive stress-testing of this code is impossible due to the combinatorial explosion, we still confirmed that it computes bitwise identical control points between adjacent patches for some of our test data and OpenSubdiv examples.

3.4.2. *Optimal Epsilon Selection*

Our method terminates clipping iterations when the maximum extent of a cropped patch becomes less than a threshold ϵ, or when the clipping iteration reaches $kMaxLevel$ iterations. We then find an intersection with a ray by approximating the surface with a bilinear patch. The optimal size of the bilinear patch is ideally smaller than 1 pixel of the screen for the eye ray.

To compute an optimal ϵ for the bilinear patch, we use ray differentials [Igehy

```
// p0 p1 p2 p3
// 0----s--------t---1
// 1----T--------S---0
BezierCrop(vec r[], vec p[], int stride, float s, float t) {
  vec p0 = p[0*stride],
      p1 = p[1*stride];
      p2 = p[2*stride];
      p3 = p[3*stride];
  float T = 1-s, S = 1-t;
  s = 1-T; t = 1-S;

  r[0*stride] =
    (p0*T*T*T + p3*s*s*s) + (p1*3*s*T*T  + p2*3*s*s*T);
  r[1*stride] =
    (p0*T*T*S + p3*s*s*t) + (p1*T*(2*S*s + T*t) + p2*s*(2*t*T + s*S));
  r[2*stride] =
    (p3*t*t*s + p0*S*S*T) + (p2*t*(2*s*S + t*T) + p1*S*(2*T*t + S*s));
  r[3*stride] =
    (p3*t*t*t + p0*S*S*S) + (p2*3*S*t*t  + p1*3*S*S*t);
}

BezierPatchCrop(vec cp[16], dir clipDir,
                float u0, float u1, float v0, float v1) {
  if (clipDir == U) {
    for (int i = 0; i < 4; ++i) BezierCrop(cp[i*4], cp[i*4], 1, u0, u1);
  } else {
    for (int i = 0; i < 4; ++i) BezierCrop(cp[i], cp[i], 4, v0, v1);
  }
}
```

Figure 6. Reversal-invariant Bézier cropping.

1999]. We approximate the distance from the ray origin to the patch by the hit distance to the axis-aligned bounding box of the patch, and conservatively estimate optimal epsilon. For the eye ray, the optimal epsilon ϵ is computed as follows,

$$\epsilon = \frac{1}{2} \cdot \min\left(\left| \frac{\partial P}{\partial x} \right|, \left| \frac{\partial P}{\partial y} \right| \right) \qquad (1)$$

where $\frac{\partial P}{\partial x}$ and $\frac{\partial P}{\partial y}$ are the partial derivatives of an approximate hit position P, which is estimated according to the pixel plane (see Fig. 7 right).

The optimal epsilon for secondary ray or others are computed similarly by tracing ray differentials.

3.4.3. *Conservative Intersection Algorithm*

The refinement of the surface to bicubic patches may result in adjacent patches that meet at the split between two patches at the next level of subdivision. These patches are identified as transition patches during feature adaptive analysis. Each of these transition patches has one to four transitioning edges which split at the midpoint of the edge. For GPU rasterizer-based rendering, a transition patch will be split into several sub-patches to avoid cracks where the tessellation interval mismatches [Nießner et al. 2012a]. Though there is no tessellation in our method, cracks may still appear as a result of the floating point computation if each side of a transitioning boundary edge is computed from control points at different levels of refinement.

We introduce a *conservative intersection algorithm* to address this problem. When the ray fails to intersect a transitioning edge, the method further subdivides the patch into four sub-Bézier patches at the middle of the parameter domain, $(u, v) = (\frac{1}{2}, \frac{1}{2})$. We then search for an intersection within these sub patches (Fig. 7 left and middle). See Fig. 8 for results. This *conservative intersection algorithm* is only needed when the expected intersection point is along one of the transitioning boundary edges of a transition patch. To guarantee the watertightness between these four sub-patches and adjacent patches, we override the border control points of the adjacent patches with control points from the subdivided four sub-patches during B-spline-to-Bézier conversion process. This guarantees that they are bitwise identical.

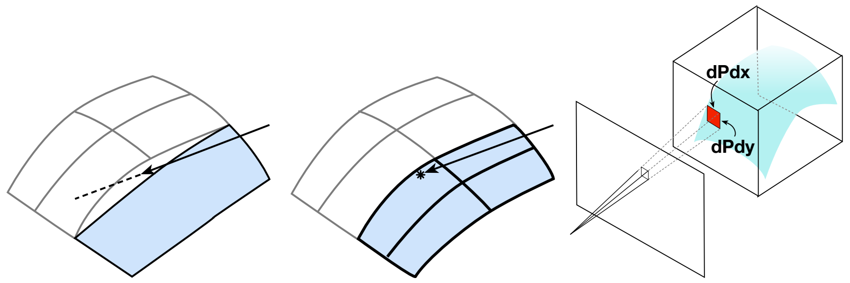


Figure 7. Left: Ray misses the surface because of a crack. Middle: Our conservative intersection algorithm generates four sub-patches on demand to fill a crack. Right: Ray differentials for the optimal ϵ selection.

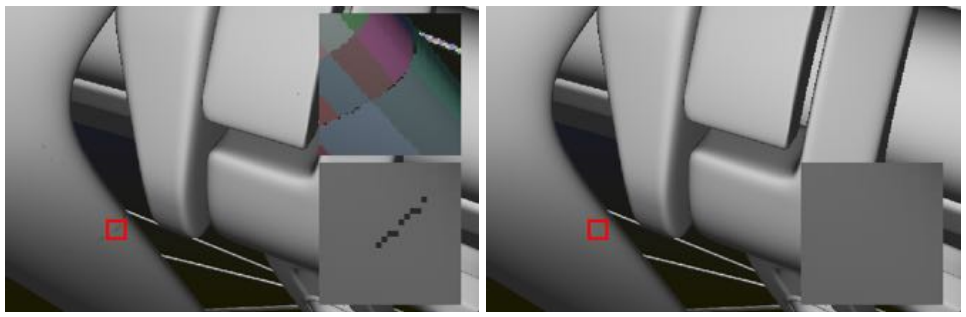


Figure 8. A rendering from the Bike model. Left: A crack appears without the conservative intersection algorithm near transition patch. Right: Crack-free rendering with the conservative intersection algorithm.

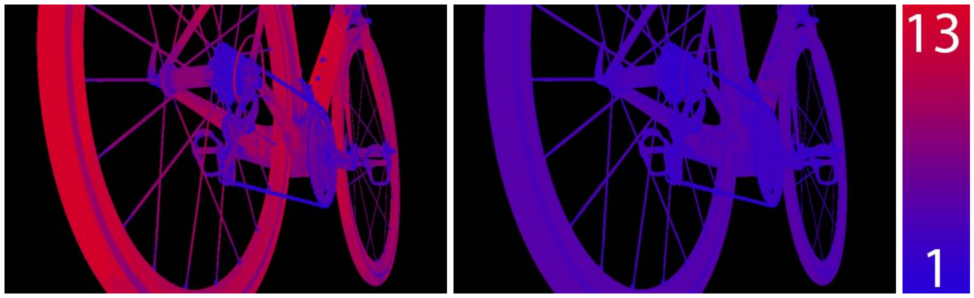


Figure 9. Visualization of the recursion level for Bézier intersection. Left: Bézier clipping disabled (Naíve midpoint subdivision). Right: Bézier clipping enabled.

4. Implementation and Results

Our implementation uses the OpenSubdiv [OpenSubdiv] library and custom C++ ray tracer. The input mesh is first processed by OpenSubdiv in order to build a feature adaptive hierarchical bicubic patch representation.

For the BVH, we have implemented a BVH builder with binned SAH partition [Shevtsov et al. 2007]. We compared two configurations for the maximum number of primitives (Bézier patches) in the leaf node, which are 2 and 16. Multi-threading is implemented using Intel Thread Building Blocks [Intel Threading Building Blocks].

4.1. Bézier Clipping Efficiency

Fig. 9 shows the efficiency that our clipping algorithm gains by exploiting the convex hull property of Bézier patches. Most pixels converge in less than five levels of recursion (Fig. 9 right). By comparison, they would need up to 13 levels of recursion to converge if we always split the patch at the midpoint (naíve intersection method for Bézier patch. Fig. 9 left).

4.2. Memory and Performance

We measured the performance of our renderer for the Bike model against the following methods: Pre-tessellation (PreTess), our Bézier clipping method (BC) and our method with conservative intersection algorithm (BCC). The maximum number of Bézier clipping iteration ($kMaxLevel$) is set to 16, which is enough high for most view configurations in our experiment. All methods use single precision.

The Bike model is composed of 51k faces of initial control cage, with 30k semi-sharp creased edges. Feature adaptive isolation produces 1.1M patches at isolation level 5. Applying uniform pre-tessellation up to the same level of subdivision generates 104M triangles. Fig. 1c and Fig. 10 show the memory footprint of both PreTess and BC/BCC methods. While the number of triangles of the uniform tessellation increases by 4x for each level, the number of Bézier patches stays moderate. Note that

Method	**# of Leaf Primitives**	**Lvl 1** (in MB)	**Lvl 2**	**Lvl 3**	**Lvl 4**	**Lvl 5**
PreTess	16	18.3	67.7	241.6	854.5	3164.0
BC16/BCC16	16	32.2	78.6	135.5	181.5	225.9
BC2/BCC2	2	43.2	106.1	182.8	245.3	305.2
# of triangles		0.4M	1.6M	6.5M	26M	104M
# of Bézier patches		164K	402K	694K	931K	1.1M

Figure 10. Memory footprints (geometry + BVH) of PreTess and BC/BCC in different levels. Triangle vertices are indexed.

the level has different meaning in PreTess and BCC/BC. PreTess level is a subdivision level, whereas BCC/BC level is a feature isolation level. In PreTess, the surface smoothness depends on the level. On the other hand, the surface is always smooth in BCC/BC since there is no tessellation, and the level controls the accuracy around extraordinary vertices and creases instead.

Fig. 11 and Fig. 12 show the performance in two view configurations. The rendering time of BCC16 is between Lvl 4 and Lvl 5 of PreTess. BCC2 requires more memory for BVH but it is much faster, since Bézier intersection is more expensive than BVH traversal. BCC crack-free rendering is similar to BC, which shows the efficiency of conservative intersection algorithm.

5. Conclusion and Future Work

We present an efficient and practical method for ray tracing of Catmull-Clark subdivision surfaces with semi-sharp creases and hierarchical edits. Our method uses Bézier patches for the efficient and precise computation of the intersection.

We propose robust, crack-free and automatic precision selection independent of geometric scale for Bézier-ray intersection. These properties are attractive for production rendering, as well as other applications including collision detection.

Our method is competitive with pre-tessellation methods in speed, while significantly reducing memory requirements. In this paper, we mainly focused on memory reduction and we evaluate results against uniformly tessellated triangles. We wish to pursue a more comprehensive performance comparison against other optimized renderers as future work.

Although some renderers use adaptive tessellation for efficiency, they often generate comparable numbers of triangles to those used in our tests. We also see many additional benefits from using a higher-order representation, including faster BVH updating for real-time animation and less bandwidth usage.

Our technique is computationally intensive and works on a large data set with

View	**Method**	**Level**	**Traverse (sec)**	**Intersect (sec)**	**Total time (sec)**
Distant (Fig. 1a)	PreTess	3	0.053	0.051	0.138
	PreTess	4	0.072	0.081	0.192
	PreTess	5	0.124	0.181	0.368
	BCC16	3	0.037	0.186	0.252
	BCC16	4	0.040	0.191	0.260
	BCC16	5	0.042	0.198	0.271
	BCC2	3	0.047	0.136	0.212
	BCC2	4	0.050	0.142	0.217
	BCC2	5	0.053	0.137	0.218
	BC2	5	0.055	0.120	0.205
Close (Fig. 1b)	PreTess	3	0.447	0.689	1.391
	PreTess	4	0.770	1.286	2.512
	PreTess	5	1.517	2.593	4.961
	BCC16	3	0.265	2.233	2.689
	BCC16	4	0.284	2.520	3.022
	BCC16	5	0.325	2.940	3.519
	BCC2	3	0.362	0.584	1.033
	BCC2	4	0.398	0.610	1.106
	BCC2	5	0.456	0.673	1.226
	BC2	5	0.452	0.558	1.116

BCC16 : Bézier Clipping & conservative intersection, leaf primitives = 16
BCC2 : Bézier Clipping & conservative intersection, leaf primitives = 2
BC2 : Bézier Clipping, leaf primitives = 2

Figure 11. Rendering time comparison at 1k x 1k resolution. Measured on Intel Xeon E5-2680 2.7 GHz (8 cores, 16 threads). Traverse + Intersect does not equal to Total time because of shading and threading management cost, etc. Intersect of BC and BCC includes the normal vector evaluation.

a small working-set memory footprint. It is thus a good candidate for seeking real-time rendering and deformation of subdivision surfaces through a massively parallel implementation on a GPU.

We also plan to add support for displacement mapping using a Ptex-like data structure [Lee et al. 2000; Nießner and Loop 2013] for detailed geometric representation, which will make our method more practical for use in production rendering.

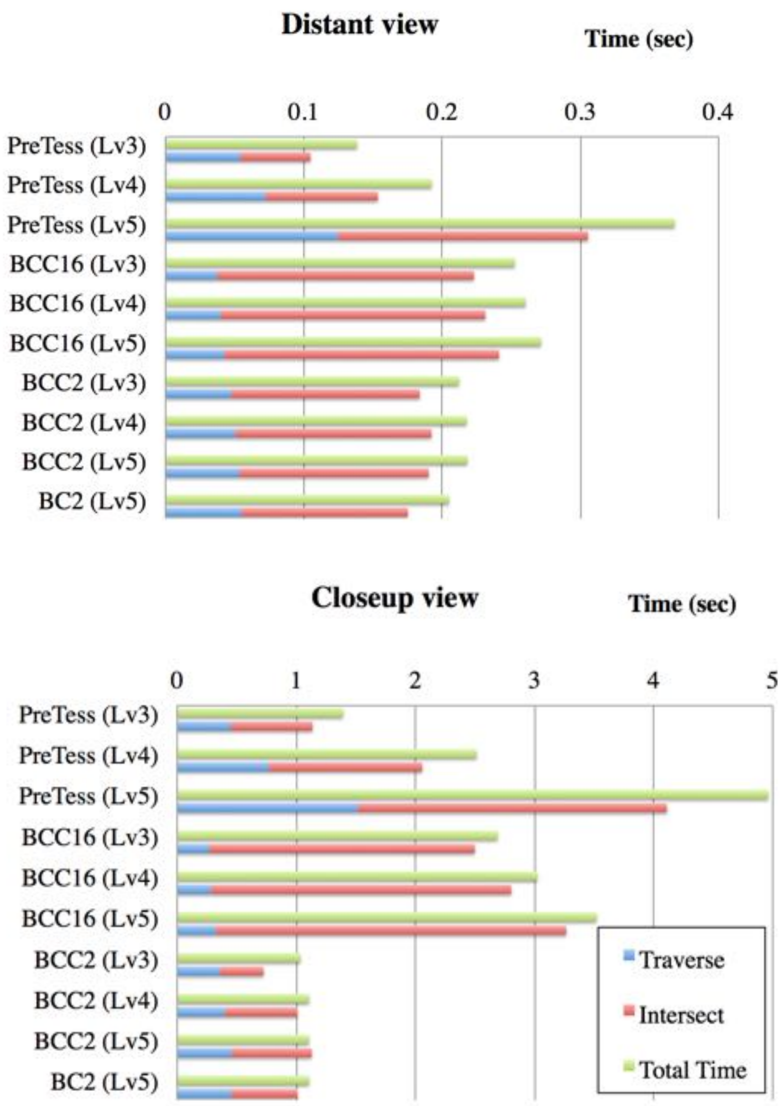


Figure 12. Plot of the Bike model rendering time.

Acknowledgements

We thank Yasutoshi Mori for the courtesy of the Bike model and the car model[1]. We also thank Manuel Kraemer, Mark Meyer, Tony DeRose, David G. Yu and the reviewers for their insightful comments and reviews and Amanda Turner of Williams College for proofreading

[1]https://github.com/MirageYM/3DModels

References

BENTHIN, C., BOULOS, S., LACEWELL, D., AND WALD, I. 2007. Packet-based ray tracing of catmull-clark subdivision surfaces. SCI Institute Technical Report UUSCI-2007-011, University of Utah. URL: http://www.sci.utah.edu/publications/SCITechReports/UUSCI-2007-011.pdf. 71

CHRISTENSEN, P. H., LAUR, D. M., FONG, J., WOOTEN, W. L., AND BATALI, D. 2003. Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes. In *Computer Graphics Forum*, vol. 22, Wiley Online Library, 543–552. URL: http://dx.doi.org/10.1111/1467-8659.t01-1-00702. 71

DEROSE, T., KASS, M., AND TRUONG, T. 1998. Subdivision surfaces in character animation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, ACM, 85–94. URL: http://doi.acm.org/10.1145/280814.280826. 70

EFREMOV, A., HAVRAN, V., AND SEIDEL, H.-P. 2005. Robust and numerically stable bézier clipping method for ray tracing nurbs surfaces. In *Proceedings of the 21st spring conference on Computer graphics*, ACM, 127–135. URL: http://dl.acm.org/citation.cfm?id=1090144. 71, 74

FORSEY, D. R., AND BARTELS, R. H. 1988. Hierarchical b-spline refinement. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, ACM, 205–212. URL: http://doi.acm.org/10.1145/54852.378512. 70

IGEHY, H. 1999. Tracing ray differentials. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 179–186. URL: http://dx.doi.org/10.1145/311535.311555. 75

INTEL THREADING BUILDING BLOCKS. URL: https://www.threadingbuildingblocks.org/. 77

KOBBELT, L. P., DAUBERT, K., AND SEIDEL, H.-P. 1998. Ray tracing of subdivision surfaces. In *Rendering Techniques 98*. Springer, 69–80. URL: http://dx.doi.org/10.1007/978-3-7091-6453-2_7. 71

LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 85–94. URL: http://dx.doi.org/10.1145/344779.344829. 79

MARTIN, W., COHEN, E., FISH, R., AND SHIRLEY, P. 2000. Practical ray tracing of trimmed nurbs surfaces. *J. Graph. Tools 5*, 1 (Jan.), 27–52. URL: http://dx.doi.org/10.1080/10867651.2000.10487519. 71

MÜLLER, K., TECHMANN, T., AND FELLNER, D. 2003. Adaptive ray tracing of subdivision surfaces. In *Computer Graphics Forum*, vol. 22, Wiley Online Library, 553–562. URL: http://dx.doi.org/10.1111/1467-8659.t01-2-00703. 71

NIESSNER, M., AND LOOP, C. 2013. Analytic displacement mapping using hardware tessellation. *ACM Trans. Graph. 32*, 3 (July), 26:1–26:9. URL: `http://doi.acm.org/10.1145/2487228.2487234`. 79

NIESSNER, M., LOOP, C., MEYER, M., AND DEROSE, T. 2012. Feature-adaptive gpu rendering of catmull-clark subdivision surfaces. *ACM Trans. Graph. 31*, 1 (Feb.), 6:1–6:11. URL: `http://doi.acm.org/10.1145/2077341.2077347`. 71, 72, 73, 76

NIESSNER, M., LOOP, C. T., AND GREINER, G. 2012. Efficient evaluation of semi-smooth creases in catmull-clark subdivision surfaces. In *Eurographics (Short Papers)*, 41–44. URL: `http://dx.doi.org/10.2312/conf/EG2012/short/041-044`. 71, 72

OPENSUBDIV. OpenSubdiv 3.0.0 alpha. URL: `http://graphics.pixar.com/opensubdiv/`. 70, 71, 77

PIXAR ANIMATION STUDIOS, 2005. The RenderMan interface version 3.2.1. URL: `http://renderman.pixar.com/view/rispec/`. 70

RAMSEY, S. D., POTTER, K., AND HANSEN, C. 2004. Ray bilinear patch intersections. *Journal of Graphics Tools 9*, 3, 41–47. URL: `http://dx.doi.org/10.1080/10867651.2004.10504896`. 74

RUBIN, S. M., AND WHITTED, T. 1980. A 3-dimensional representation for fast rendering of complex scenes. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '80, ACM, 110–116. URL: `http://doi.acm.org/10.1145/800250.807479`. 74

SHEVTSOV, M., SOUPIKOV, A., AND KAPUSTIN, A. 2007. Highly Parallel Fast KD-tree Construction for Interactive Ray Tracing of Dynamic Scenes. *Computer Graphics Forum 26*, 3, 395–404. URL: `http://dx.doi.org/10.1111/1467-8659.t01-2-00703`. 77

Author Contact Information

Takahito Tejima
Pixar Animation Studios
1200 Park Ave
Emeryville, CA 94608
takahito@pixar.com

Masahiro Fujita
Light Transport Entertainment
syoyo@lighttransport.com

Toru Matsuoka
DeNA Inc.
tx.matsuoka@gmail.com

Takahito Tejima, Masahiro Fujita, Toru Matsuoka, Direct Ray Tracing of Full-Featured Subdivision Surfaces with Bézier Clipping, *Journal of Computer Graphics Techniques (JCGT)*,

vol. 4, no. 1, 69–83, 2015
http://jcgt.org/published/0004/01/04/

Received:	2014-11-04		
Recommended:	2015-03-16	Corresponding Editor:	Morgan McGuire
Published:	2015-03-31	Editor-in-Chief:	Morgan McGuire