Pacific Graphics 2016
E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Volume 35 (2016), Number 7

TSS BVHs: Tetrahedron Swept Sphere BVHs for Ray Tracing Subdivision Surfaces

P. Du [1,2†] and Y. J. Kim[3] and S. E. Yoon[1‡]

[1]Korea Advanced Institute of Science and Technology, South Korea
[2]Hangzhou Dianzi University, China
[3]Siemens PLM Software, USA

Abstract

We present a novel, compact bounding volume hierarchy, TSS BVH, for ray tracing subdivision surfaces computed by the Catmull-Clark scheme. We use Tetrahedron Swept Sphere (TSS) as a bounding volume to tightly bound limit surfaces of such subdivision surfaces given a user tolerance. Geometric coordinates defining our TSS bounding volumes are implicitly computed from the subdivided mesh via a simple vertex ordering method, and each level of our TSS BVH is associated with a single distance bound, utilizing the Catmull-Clark scheme. These features result in a linear space complexity as a function of the tree depth, while many prior BVHs have exponential space complexity. We have tested our method against different benchmarks with path tracing and photon mapping. We found that our method achieves up to two orders of magnitude of memory reduction with a high culling ratio over the prior AABB BVH methods, when we represent models with two to four subdivision levels. Overall, our method achieves three times performance improvement thanks to these results. These results are acquired by our theorem that rigorously computes our TSS bounding volumes.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

Subdivision surfaces such as Catmull-Clark and Loop schemes have been widely used for representing various models in computer graphics. Especially, the Catmull-Clark scheme has been a default standard for the geometric modeling and has surprisingly close-to-optimal properties [Cas12], while it is one of earliest subdivision schemes.

Ray tracing and its derived physically based rendering techniques such as Monte Carlo path tracing have been extensively studied. These techniques can support realistic effects such as motion blur, soft shadows, and reflections. Unfortunately, most of these techniques are designed for polygonal meshes, but less on subdivision meshes [WMG*09], compared to its importance in the field.

Ray tracing subdivision surfaces commonly have two problems. The first issue is related to a high memory requirement of maintaining subdivided meshes and acceleration data structures on those meshes. The second issue is a low culling ratio of existing acceleration data structures such as bounding volume hierarchies based on Axis-Aligned Bounding Boxes (AABBs). For example, we found that ray tracing a virtual museum scene consisting of 7.9 M patches in its control mesh requires 36.82 GB for having 809 M subdivided

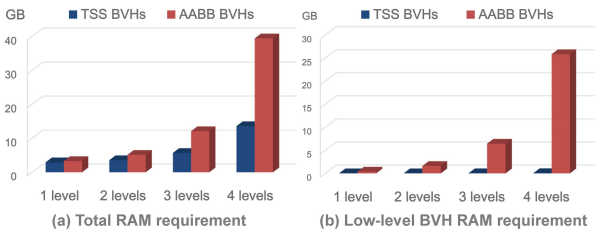

Figure 1: *Memory requirement of different methods. Different methods use the same, upper-level AABB BVHs, and subdivided meshes. Our methods using TSS BVHs in their low-level BVHs have the linear storage complexity, while prior methods such as AABB BVHs have exponential storage complexity as a function of the tree depth. (a) shows the total memory requirement of different methods, while (b) shows memory requirement of low-level BVHs. Our TSS BVHs take 12.6 MB, 25.2 MB, 37.8 MB, and 50.4 MB from the first to the fourth subdivision levels, respectively.*

† dupengtomas@gmail.com
‡ sungeui@gmail.com

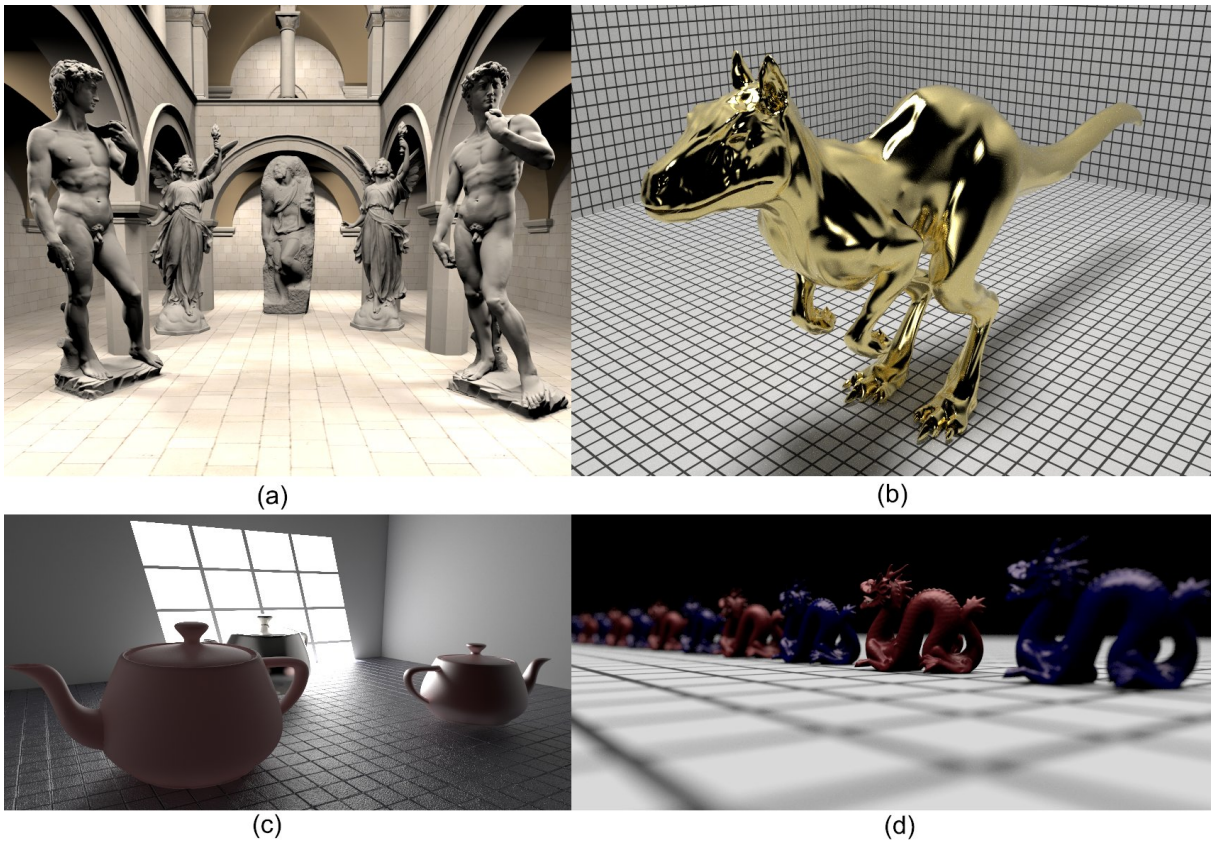

Figure 2: *We test our method with a variety of rendering effects generated by path tracing and photon mapping.*

quads with four subdivision level and their AABB BVHs: 25.86 GB for AABB BVHs and 10.96 GB RAM for subdivided quads.

To address them, prior techniques on ray tracing subdivision surfaces focused on adaptive tessellation methods [MTF03, CLF*03], packetization [BBLW07], and on-demand computations [BWN*15]. Nonetheless, ray tracing subdivision surfaces has been considered slow and required high memory footprints at runtime. Unfortunately, tighter BVHs tailored for subdivision surfaces with compact memory requirement have been less studied.

Main contributions. To address these problems, we propose a novel bounding volume hierarchy tailored for handling the Catmull-Clark subdivision scheme. Especially, we propose to use Tetrahedron Swept Sphere (TSS) to effectively bound each patch of Catmull-Clark scheme (Sec. 4.3). Our TSS based BVH does not encode any geometric information, but encodes a distance bound per each depth. This feature leads our TSS BVH to have a linear space complexity. We also explain how to compute an error-bounded subdivision within a user-specified error bound to the limit surface (Sec. 4.1), and describe parallel slab based culling (Sec. 4.4) to efficiently cull out our TSS bounding volumes.

To demonstrate benefits of our methods, we have tested four different benchmarks with path tracing and photon mapping (Fig. 2). We found that our new acceleration hierarchy achieves a high intersection test culling ratio (up to 97 %). Furthermore, our TSS BVHs have a linear space complexity as a function of the hierarchy depth, while prior methods have the exponential growth rate. As a result, we found that TSS BVHs achieve more than two orders of magnitude memory reduction over the prior AABB BVHs, when we have the subdivision level of three or four (Fig. 1). Overall, we get three times overall memory reduction considering all the subdivided meshes and BVHs, and similar running time improvements when considering four subdivision levels in our tested benchmarks, compared with the prior AABB method. We believe that our BVH representation takes one step closer toward handling subdivision surfaces more efficiently for ray tracing.

2. Related Works

In this section, we review subdivision schemes, mainly the Catmull-Clark scheme, and various ray tracing methods for the scheme.

2.1. Subdivision Schemes

Many subdivision techniques have been proposed, since they can provide unlimited resolutions. Influential techniques include Catmull-Clark [CC78], Doo-Sabin [DS78], and Loop [Loo78] subdivision schemes. Catmull-Clark and Doo-Sabin schemes are based on tensor product of B-Splines and thus operate on quadrilateral

meshes, while the Loop scheme is a popular subdivision scheme for triangle meshes. Mathematically, the limit subdivision surfaces constructed by Doo-Sabin, Catmull-Clark and Loop schemes correspond to bi-quadratic, bi-cubic and bi-quartic spline surfaces with singularities, respectively. See recent surveys for more detailed information [Cas12, ZS00].

Recently, several researchers have considered to use GPU techniques for efficiently tessellating the Catmull-Clark scheme. Different kinds of data structure are proposed to represent Catmull-Clark subdivision meshes such that the subdivision process can be executed on the GPU completely based on these data structures [NLMD12, PEO09]. To reduce memory requirement and computational overheads, Loop and Schaefer proposed a GPU tessellation algorithm for visually approximating Catmull-Clark subdivision surfaces using a collection of bi-cubic patches (one for each face of a quad-mesh), which is C^2 continuous everywhere except at extraordinary vertices [LS08]. These techniques are mainly designed for the rasterization scheme, not for ray tracing methods. For our work with ray tracing, we propose to use an error-bounded subdivision scheme, to provide certain guarantees on ray intersection tests with subdivision schemes.

2.2. Ray Tracing Subdivision Meshes

Ray tracing subdivision meshes has been developed for improving interactive performance and reducing the memory requirement. Müller et al. presented an adaptive subdivision scheme, where more refinement steps are adaptively applied to the surface, as the surface is more closely located to the ray [MTF03]. Christensen et al. [CLF*03] proposed an efficient multi-resolution measure based on the observation that coherent rays have small differentials, while incoherent rays have large ones.

Benthin et al. [BBLW07] proposed a ray tracing Catmull-Clark subdivision surfaces approach based on ray packet. To avoid memory overflow caused by large scale models, this approach does not retain any of the subdivided patches in memory once rays terminate. To efficiently evaluate the underlying surface representation, Benthin et al. also proposed to lazily tessellate necessary patches, while utilizing adaptive subdivision [BWN*15].

Recently, converting the base patches to Bezier patches with Newton or Bezier clipping methods [TFM15] has been shown to reduce the memory size and could be fast. Nonetheless, tessellating subdivision surfaces into triangles and performing intersection tests with them has been commonly adopted in practice. We therefore choose to study this approach in this work.

Kobbelt et al. [KDS98] proposed envelop meshes to bound the limit surface of subdivision surfaces, and was tested with the Loop scheme. The upper and lower envelope meshes are computed by moving the vertices of the limit surface in the corresponding normal directions, and are used for ray intersection tests. While this scheme can provide the same error reduction rate to our method, the envelope meshes are required to be separately stored or locally refined, while our approach encodes only one error bound for each depth.

While useful subdivision schemes and caching techniques have been proposed for ray tracing subdivision schemes, BVHs used for handling triangular meshes are commonly adopted even for ray tracing subdivision meshes. In this paper, we propose a novel BVH tailored for the Catmull-Clark subdivision scheme.

3. Overview

We give a brief overview on the Catmull-Clark subdivision scheme and its issues related to ray tracing, followed by presenting high-level ideas of our approach.

3.1. Catmull-Clark Subdivision Scheme

The Catmull-Clark scheme recursively refines the subdivision mesh from its initial mesh, the control mesh. It is an explicit iteration method, which uses the coordinates of the current level to compute the coordinates in the next level. Each iteration process can be summarized as follows:

1. Generate new face points, f^{new}, by computing the center of each patch, where d_j is j-th vertex coordinate among n_p different vertices of the patch: i.e. $f^{new} = \frac{1}{n_p} \times \sum_{j=1}^{n_p} d_j$.
2. Generate new edge points, e^{new}, by averaging the two vertices, v_j, of each edge and two face points, f_j, of its neighboring faces: $e^{new} = \frac{1}{4} \sum_{j=1}^{2} (f_j + v_j)$.
3. Refine vertices. For each vertex of the primitive, the new vertex, v^{new}, is the weighted average of the adjacent edge point mean E, face point mean F, and itself v, where n is the number of face points incident to the vertex: i.e. $v^{new} = \frac{F+2E+(n-3)v}{n}$.
4. Generate a new mesh by connecting each new face point to all of its surrounding new edge points.

To support the Catmull-Clark subdivision scheme for ray tracing, many different methods have been proposed, as discussed in Sec. 2. Nonetheless, it has remained as one of challenging problems for ray tracing the Catmull-Clark subdivision schemes due to its high memory requirement and slow ray-patch intersection tests.

3.2. Our Approach

To address the high memory requirement and slow ray-patch intersection tests for the Catmull-Clark subdivision scheme, we propose a novel bounding volume, TSS, for ray tracing the scheme.

As one of design goals for our method, we allow users to set a tolerance threshold between a subdivided mesh and the limit surface determined by the input control mesh. As a result, users can directly control the resolution and accuracy of computed subdivision meshes. Based on the tolerance, we also construct our acceleration data structure and perform various ray tracing operations.

We first compute a subdivision mesh from the input control mesh given the user-specified error tolerance. As our basic acceleration data structure for ray tracing the Catmull-Clark scheme, we build a two-level BVH from the input control mesh. The two-level BVH (Fig. 3) consists of two parts: an upper AABB BVH and lower TSS BVHs. We use an AABB BVH for the upper BVH computed from the input control mesh. The AABB BVH is pre-constructed in a bottom-up method to bound the limit surface defined by the input

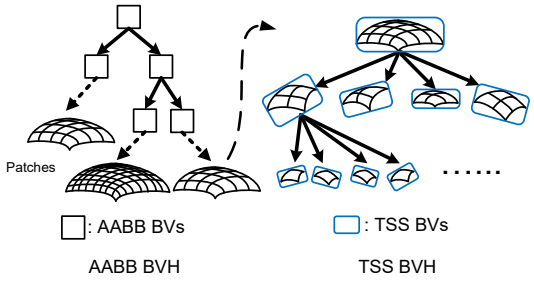

Figure 3: *Two-level BVHs: We build a two-level BVH for the Catmull-Clark subdivision mesh given a user-specified error tolerance. The upper part is a common AABB BVH constructed from patches of the control mesh, while each lower TSS BVH with TSS bounding volumes is constructed on demand in a top-down method. This figure shows 2D illustration of AABB and our TSS bounding volumes (BVs), while exact 3D TSS BVs are shown in Fig. 6.*

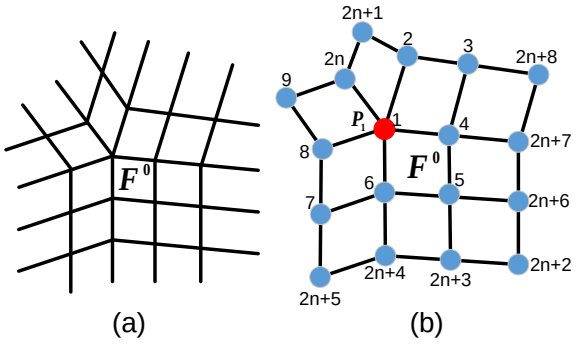

Figure 4: *Given a central control patch, F^0, in a control mesh, i.e., subdivision level zero (shown in the left), we show its vertex ordering of the one ring neighbors of the patch on the right. n is the valence of P_1, which is 5 in the example.*

control mesh. The leaf node of the upper AABB BVH contains a single patch of the control mesh, and its AABB bounds the quad and its limit surface. We then construct the lower TSS BVH from the quad patch stored in each leaf node of the upper AABB BVH. We build the TSS BVH on demand in a top-down manner to further bound the subdivided mesh of the patch given the error tolerance. We associate our TSS bounding volume with each node of the TSS BVH.

During the rendering process, we traverse the two-level BVH to effectively cull away rays that do not intersect with the subdivided mesh. While traversing the upper AABB BVH, we perform standard ray-AABB intersection tests. When we traverse nodes of TSS BVH, we perform parallel slab based culling to effectively cull out non-intersection pairs of the ray and TSS. If an intersection occurs, we decompose the intersected node into four sub-nodes and use the culling method recursively. As we reach the leaf node of the TSS BVH, we perform an intersection test between the ray and the single quad associated with the leaf node; specifically, we break the quad into two triangles and perform ray-triangle intersection tests.

4. Ray Tracing Subdivision Meshes

In this section, we explain main components of our method.

4.1. Error-Bounded Subdivision

We would like to bound computed subdivision meshes within the user specified error bound, ε, from the limit surface. To achieve our goal, we need to identify a conservative subdivision level from its control mesh. Fortunately, this has been studied earlier, and we utilize prior techniques.

Suppose that we have a central control patch, $F^k(u,v)$, with subdivision level k. Fig. 4(a) shows an example of the central control patch $F^k(u,v)$ at its control mesh, i.e., k is 0. We also arrange vertices of the one ring neighbors of the central control patch $F^k(u,v)$ based on the control vertex ordering suggested by Stam's method [Sta98]. Fig. 4(b) shows an example of such a vertex ordering from the control patch, where n represents the valence of a vertex, P_i $(1 \le i \le 2n+8)$.

The error between the subdivision mesh computed at subdivision level k and its limit surface, $S(u,v)$, is then bounded as the following [CCY06]:

$$||F^k(u,v) - S(u,v)|| \le \frac{1}{z}\left(\frac{1}{w}\right)^k M, \tag{1}$$

where M is the second order norm for the control mesh of $F^0(u,v)$, and defined as the following:

$$\begin{aligned} M = max\{&\{||P_{2i} - 2P_1 + P_{2(i+1)\%n+1}|| : 1 \le i \le n\} \cup \\ &\{||P_{2i+1} - 2P_{2(i\%n)+2} + P_{2(i\%n)+3}|| : 1 \le i \le n\} \cup \\ &\{||P_2 - 2P_3 + P_{2n+8}||, ||P_1 - 2P_4 + P_{2n+7}||, \\ &||P_6 - 2P_5 + P_{2n+6}||, ||P_4 - 2P_5 + P_{2n+3}||, \\ &||P_1 - 2P_6 + P_{2n+4}||, ||P_8 - 2P_7 + P_{2n+5}||, \\ &||P_{2n+6} - 2P_{2n+7} + P_{2n+8}||, ||P_{2n+2} - 2P_{2n+6} + P_{2n+7}||, \\ &||P_{2n+2} - 2P_{2n+3} + P_{2n+4}||, ||P_{2n+3} - 2P_{2n+4} + P_{2n+5}|| \\ &\}\}. \end{aligned}$$

w and z are then defined as follows, depending on the valence of the vertex P_i:

$$w = \begin{cases} \frac{3}{2} & n = 3 \\ 4 & n = 4 \\ \frac{25}{18} & n = 5 \\ \frac{4n^2}{3n^2+8n-46} & n > 5, \end{cases} \quad z = \begin{cases} 1 & n = 3 \\ 3 & n = 4 \\ \frac{25}{18} & 5 \le n \le 8 \\ \frac{4(n^2-8n+46)}{n^2} & n > 8. \end{cases}$$

Ineq. 1 requires the initial quadrilateral, control mesh to be subdivided at least twice [Sta98] to guarantee that each face of the subdivided mesh contains at most one extra-ordinary vertex, whose valence is not equal to four. Based on Ineq. 1, we can get the minimum subdivision level k under the given error bound ε as the following:

$$k = \lceil log_w \frac{M}{z\varepsilon} \rceil. \tag{2}$$

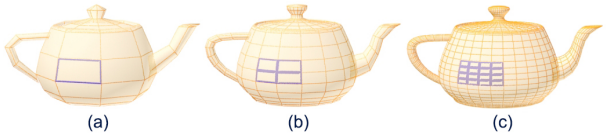

Figure 5: *The teapot shown in subdivision meshes computed with different error bounds. From the left, we use 2, 0.5, 0.1 for* ε*. In these cases, the teapot is approximated with 126, 504 and 2016 quads, respectively.*

Given the chosen subdivision level, the actual distance bound, ε_k, between the computed mesh with k subdivision level and its limit surface is computed as the following:

$$\varepsilon_k = \frac{1}{z}(\frac{1}{w})^k M, \tag{3}$$

where $\varepsilon_k \leq \varepsilon$. In practice, ε_k tends to become much smaller than ε. For example, ε_k becomes 0.002 when ε is set to be 0.1 for the virtual museum scene (Fig. 2(a)). Fig. 5 show results of approximating the teapot using subdivision meshes until the error bound ε_k becomes less than 0.1.

4.2. Two-Level BVH

We use a two-level BVH as our basic acceleration hierarchical structure. We use the AABB BVH for the upper-level culling structure in the granularity of patches used for the Catmull-Clark scheme. We choose simple AABBs for their efficient ray-AABB intersection tests. Two-level hierarchies are also commonly adopted in many other ray tracing approaches [LYTM08, HQL*10, KSY14].

To construct AABBs of the upper-level AABB BVH, we consider control points of each patch and then construct an AABB from those points. It has been well known that the AABB computed in this manner contains all the control points of the limit surface and all the subdivided meshes from the control mesh [Fün07]. For ray intersection tests, we simply perform ray-AABB tests for culling out rays that do not intersect during the hierarchical traversal. When we cull out such rays, we can guarantee that those rays are away from the user-specified error tolerance ε from the limit surface.

While the upper-level BVH provides a certain degree of culling, their BVs are rather loose, resulting in low culling ratio. To achieve additional culling ratio, we use a low-level BVH. Once we arrive at a leaf node of the upper-level BVH, we first check whether we constructed a low-level, TSS BVH for the leaf node [†]. If not, we first generate a subdivision mesh, whose error bound from the limit surface is less than the user-specified error bound ε (Sec. 4.1). We then construct the TSS BVH, which is significantly more compact than an AABB BVH.

[†] For this, we use a simple LRU based caching scheme that also works well in a parallel mode [KBK*10]

4.3. Tetrahedron Approximations

For efficient ray tracing of the Catmull-Clark scheme, we need a compact and effective BV representation for subdivided patches. For our goal, we utilize a tetrahedron to approximate the Catmull-Clark Subdivision Meshes (CCSM) within a conservative distance bound based on the following, tetrahedron-CCSM distance bound theorem.

Theorem 4.1 (Tetrahedron-CCSM Distance Bound) Suppose that we have $F^k(u,v)$, F^k in short, a CCSM computed with the subdivision level k. We define a tetrahedron, T^k, as a bounding volume to F^k, whose four vertices are four corner points of $F^k(u,v)$. We additionally use a bilinear surface defined from four corners of $F^k(u,v)$:

$$\begin{aligned} B_{F^k} &= (1-v)[(1-u)F^k(0,0)+uF^k(1,0)] \\ &\quad +v[(1-u)F^k(0,1)+uF^k(1,1)]. \end{aligned} \tag{4}$$

We then enlarge the tetrahedron into Tetrahedron Swept Sphere (TSS) by rolling a sphere with a radius of ε_t, the conservative bound on the difference between the CCSM and the bilinear surface, where

$$||F^k(u,v) - B_{F^k(u,v)}|| \leq max||f_{uv} - b_{uv}|| \equiv \varepsilon_t, \tag{5}$$

and f_{uv} are vertex coordinates for $F^k(u,v)$, and b_{uv} are their corresponding points on the bilinear surface $B_{F^k(u,v)}$. TSS then bounds the CCSM.

Proof To compute a conservative distance bound between the tetrahedron bounding volume T^k and the CCSM F^k, we need to compute the one sided Hausdorff distance from F^k to T^k. Consider a triangle, $\triangle_i^{F^k}$, in F^k, and let f_j $(j=1,2,3)$ be the vertices of $\triangle_i^{F^k}$. The one sided Hausdorff distance between two convex objects $\triangle_i^{F^k}$ and T^k always realizes at one of vertices f_j $(j=1,2,3)$ [BHEK10]. Based on this fact, we can compute the one sided Hausdorff distance, $h(\triangle_i^{F^k}, T^k)$, as follows:

$$h(\triangle_i^{F^k}, T^k) = \max_j d(f_j, T^k), \tag{6}$$

where $d(f_j, T^k)$ is the minimum distance between f_j and T^k. We can then easily compute $h(F^k, T^k)$ by using $h(\triangle_i^{F^k}, T^k)$ [TLK09]:

$$h(F^k, T^k) = \max_i h(\triangle_i^{F^k}, T^k). \tag{7}$$

Based on the above equations, we can derive the following:

$$\begin{aligned} h(F^k, T^k) &= \max_i h(\triangle_i^{F^k}, T^k) \\ &= \max_i \max_j d(f_j, T^k) \\ &\leq \max_i \max_j ||f_j - b_j|| = max||f_{uv} - b_{uv}|| = \varepsilon_t, \end{aligned} \tag{8}$$

where b_j is the corresponding vertex to f_j from the bilinear surface B_{F^k}. The inequality satisfies since we construct T^k from four corner points of $F^k(u,v)$. As a result, once we compute a TSS by rolling a sphere with a radius of ε_t to the tetrahedron T^k, the computed TSS bounds the CCSM F^k. □

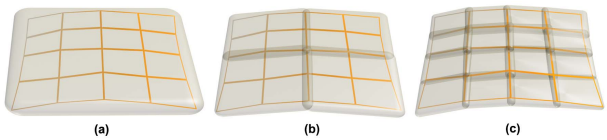

Figure 6: *This shows TSSs of our low-level BVH for a patch. From the left, it shows the first, second, and third levels of TSSs with a mesh subdivided two times.*

We build a low-level BVH, TSS BVH, for the patch in a top-down manner by utilizing Theorem 4.1. Starting from the root node of a TSS BVH containing a patch, we construct a TSS from the patch. In particular, we use four corners of the patch for constructing a tetrahedron and compute the distance bound between the patch and the bilinear surface defined by those four corners. TSS of the root node is then computed by rolling a sphere with the radius of the distance bound along the tetrahedron. We recursively apply the procedure by subdividing the patch into 2 by 2 sub-patches according to the Catmull-Clark scheme. We then perform the same procedure to each sub-patch for computing its TSS (Fig. 6). Note that we do not store four corners defining a TSS of a node. We identify those vertices from a vertex array storing vertices of a patch associated with a TSS BVH during the BVH traversal. For this, we use a simple, vertex ordering method (Sec. 4.5).

For each node of the low-level, TSS BVH, we need to store the distance bound associated with its TSS. A naive, but accurate method would store the distance bound for each node. Instead, we store a single distance bound for all the nodes in each depth of the TSS BVH, since those distance bounds are in a small range. For this, we consider all the nodes in the same depth, and store only the maximum distance bound for each depth of the low-level, TSS BVH. This enables us to store only h distance bounds, as the function of the depth, h, of the low-level BVH containing n different nodes; i.e., $h = \log n$.

To perform a TSS-ray intersection test, we can decompose the TSS into three parts: four spheres, six cylinders, and four triangular prisms. Unfortunately, this naive TSS-ray intersection test is very expensive; it can require about 290 float operations. Instead, we do not perform this naive approach, and propose to use the parallel slab based culling method as efficient TSS-ray intersection tests.

4.4. Parallel Slab based Culling for TSS BVs

We present an efficient culling technique for the TSS-ray intersection technique, inspired by prior slabs techniques [KK86]. This culling technique is based on the following, parallel slab culling theorem.

Theorem 4.2 (Parallel Slab Culling) Suppose that there are an i-th pair of two parallel planes, P_{2i} and P_{2i+1}, bounding a TSS, whose normal is $\vec{n}_i$. Suppose also that a ray is represented by its origin, o, and direction, $\vec{d}$. Suppose that intersection points between P_{2i} (and P_{2i+1}) and the ray are projected onto the normal vector $\vec{n}$, and are denoted as d_{2i} (and d_{2i+1}). The intersection time, t_{2i}, between the ray and the plane P_{2i}, along the normal vector $\vec{n}_i$ is defined as the

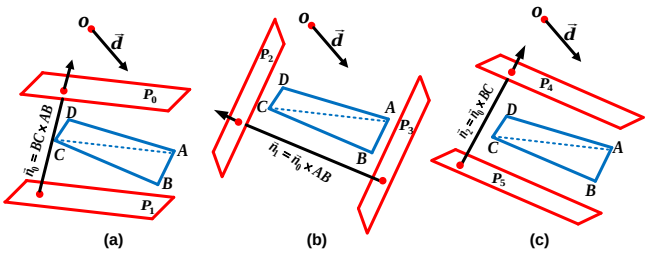

Figure 7: *This figure shows three pairs of parallel planes enclosing four vertices defining a TSS.*

Algorithm 1 Parallel slab culling that checks an intersection between a ray and a TSS.

1: Create the first pair of parallel slabs and calculate the contact times t_0 and t_1.
2: **if** $t_0 < 0 || t_1 < 0$ **then**
3: return false
4: Create the second pair of parallel slabs and calculate the contact times t_2 and t_3.
5: **if** $t_2 < 0 || t_3 < 0$ or $[t_0,t_1] \cap [t_2,t_3] = \emptyset$ **then**
6: return false
7: Create the third pair of parallel slabs and calculate the contact times t_4 and t_5.
8: **if** $t_4 < 0 || t_5 < 0$ or $[t_0,t_1] \cap [t_2,t_3] \cap [t_4,t_5] = \emptyset$ **then**
9: return false
10: return true

following:

$$t_{2i} = \frac{d_{2i} - o \cdot \vec{n}_i}{\vec{d} \cdot \vec{n}_i}. \tag{9}$$

Suppose that we have n pairs of such parallel planes. If $[t_0,t_1] \cap [t_2,t_3] \cap \cdots \cap [t_{2n},t_{2n+1}] = \emptyset$, where $[t_{2i},t_{2i+1}]$ is the intersection interval of the i-th pair of parallel planes, then there are no intersections between the ray and the TSS.

This theorem can be proved by contradiction, and, furthermore, has been used in prior ray tracing methods as slabs [KK86], which is also related to the Cyrus-Beck clipping algorithm [CB78]. In this paper, we focus on its benefits and how to use the theorem for our TSS BVs as an effective culling.

A straightforward culling on the intersection time t_i with the plane P_i is that when $t_i < 0$, it guarantees no intersection between the TSS and the ray. Note that this parallel slab culling is efficient, especially compared to the naive TSS-ray intersection test, since the culling method requires two dot products and one subtract/division float operation for computing each intersection time, resulting in 21 float operations on average.

To use the parallel slab culling method, we need to identify pairs of parallel planes, each of which encloses the TSS. Fortunately, our TSS is defined by four corner points, A,B,C, and D, of a quad that is contained in the TSS. We therefore pick three vertices A,B,C from the tetrahedron and define a plane, P_{2i}, from those three vertices. From those three vertices, we compute the normal vector $\vec{n}_i$

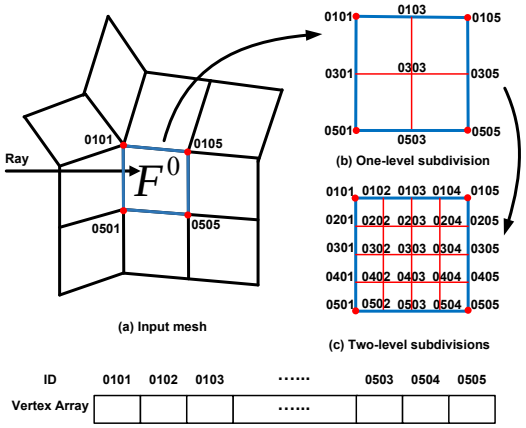

Figure 8: *Vertex ordering. The first and second two digits encode indices along v- and u- directions, respectively, and the combination of those two indices defines an ID of a vertex of a patch. We store vertices in the vertex array in the ascending order of IDs.*

of the plane. We then construct a separate, parallel plane, P_{2i+1} passing the left, fourth vertex D of the tetrahedron. The parallel slab culling theorem requires us to provide intersection points d_{2i} projected to the normal vector $\vec{n}_i$ of the plane P_{2i}. Since any points including such intersection points in the plane maps to the same position along the normal vector, we project one of vertices, e.g., A, defining the plane P_{2i}; we project D of the plane P_{2i+1} for computing d_{2i+1}. Based on this simple procedure, we can construct three pairs of parallel planes. Fig. 7(a) shows an example pair of such planes from four vertices defining a TSS.

When we construct pairs of parallel planes based only on their normals of the planes, we cannot get a high culling rate. This is mainly because the quad is nearly flat and these pairs of parallel slabs are similar. We therefore use only a single pair based on this approach, and use another set of pairs of parallel planes vertical to the cross product of the normal of a plane and an edge of the plane. We compute two pairs in this process. Fig. 7(b) and (c) show those two pairs of parallel planes. (b) is computed based on the cross product between the normal of the plane passing A,B,C vertices and an edge between A and B, and (c) is computed based on the cross product between the same normal of the plane and another edge between B and C. Overall, we consider only three pairs of parallel planes, since those three pairs can tightly bound the TSS with a small culling overhead.

The pseudo-code of this parallel culling method tailored to a TSS BV is shown in Alg. 1. To consider the distance bound ε_t associated with a TSS, we offset each of computed parallel planes as the amount of ε_t outwards from the center of the TSS.

4.5. Vertex Ordering

Storing primitives of subdivided meshes can take a high memory requirement. Fortunately, for our two-level BVHs, we do not need to explicitly store the topology of subdivision patches, resulting in a more compact representation.

Table 1: *Subdivision level and total rendering time given the error bound, 0.001. We also report the relative length of the bound to the diagonal length of the bounding box of each scene.*

Scene	Killeroo	Teapots	Dragons	Museum
Init. quads	0.6M	1.7M	7M	7.9M
Sub. quads	61M	10M	183M	51M
Sub. level	4	2	3	2
Rel. length	0.1%	5.1%	0.87%	3.6%
Avg. curvature	0.25	0.38	0.022	0.043
Render time	4336s	9636s	2001s	10407s

Given the maximum subdivision level within the user-specified error bound (Sec. 4.1), we can calculate the maximum number of vertices along u- and v-directions, i.e. n_{sub}, and we can then mark the four corner points of the original patch starting from one to n_{sub} along u- and v-directions, respectively. As a result, the combination of these indices defines unique IDs of those corner points (Fig. 8). We can then recursively apply this indexing scheme to encode vertices of subdivided patches from the original patch. Since each vertex has an unique ID, we can calculate its corresponding index in the vertex array, and store all those vertices into a vertex array with an ascending order of their IDs.

This vertex ordering scheme helps us to save memory by avoiding topology recording, and to efficiently get the IDs of the corner vertices given a subdivision level. Furthermore, we can compute IDs of those corner vertices used for TSS BVHs. As a result, we do not need to store IDs even in nodes of our TSS BVHs. Note that this scheme also works for irregular quad patches, since we do not need to consider one ring neighbors of each quad patch for ray-mesh intersection tests. Compared with a method storing topology explicitly, this simple vertex ordering method has 55% memory reduction and 8% to 11% performance improvement in our tested benchmarks.

5. Results and Comparisons

We have implemented and integrated our methods into pbrt [PH04]. We use an Intel Xeon-E5440 CPU machine with 16 GB RAM and four cores, each of which is 2.83 GHz. Since Windows 7 OS uses about 2.7 GB, our ray tracing system can use up to about 13.3 GB RAM as the maximum size of the memory pool.

5.1. Benchmarks and Tested Methods

To test behaviors of our method in a variety of rendering effects generated by path tracing and photon mapping, we have chosen the following well-known benchmarks:

- Virtual museum, Fig. 2(a). This model contains one Matthew, two Lucy, and two David models in the Sponza scene. We use path tracing with low-discrepancy sampling. All those scanned models are represented by 7.9 M patches, while the Sponza scene is represented by triangle meshes. To illuminate the scene, we use 12 area lights, generate 128 samples, i.e., 128 rays per pixel.
- Killeroo, Fig. 2(b). This model is represented by a highly glossy gold material and its image is generated by one light with path

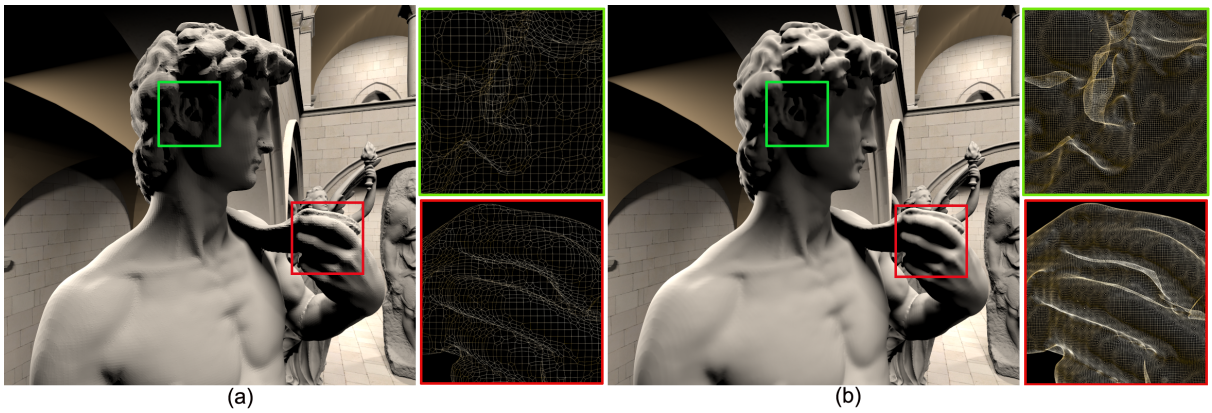

Figure 9: *Rendering results with different subdivision levels of the David model, whose control mesh has* 1*M quads. By having more subdivisions (two more subdivisions) for Fig. (b) we can achieve more smooth and visually pleasing results than Fig. (a).*

tracing. The Killeroo model consists of 0.6 M patches, while its background walls are represented with triangles.

- Teapots, Fig. 2(c). This scene contains three teapots with different materials, translucent, metal and plastic ones, in a room. Three teapots consist of 1.7 M patches. We use photon mapping, and generate 0.4 M indirect photons and use 128 rays per pixel and 75 gathering samples.
- Dragons, Fig. 2(d). The scene contains twelve instanced dragons consisting of 7 M patches with depth-of-field using path tracing.

To show benefits of our methods, we have tested the following techniques:

- Our two-level BVHs w/ parallel slab culling (TSS BVHs w/ PSC). This uses our two-level BVHs consisting of the upper AABB BVH and low-level TSS BVHs. It also uses Parallel Slab Culling (PSC).
- Our two-level BVHs w/o parallel slab culling (TSS BVHs). This method is same to the aforementioned one except using PSC. For intersection tests, we perform TSS-ray intersection tests that check collisions between the ray and three parts, i.e., four spheres, six cylinders, and four triangular prims.
- Naive two-level BVHs (AABB BVHs). This method uses the AABB BVH for the upper level BVH as well as for low-level BVHs.

We have compared the performance of these techniques by utilizing four working threads, unless mentioned otherwise. We also used the lazy evaluation with the tessellation cache for all the tested methods, as adopted for our method. Specifically, for subdivision levels 1 and 2, we use a lazy evaluation. For subdivision levels 3 and 4, data required for our method can be fit in the cache, but data for low-level AABBs are too huge to be stored in the cache. As a result, we simply re-compute necessary low-level AABBs without caching; we found that this approach is faster than using the cache and thus adopted this approach.

As mentioned earlier, some portions of our benchmarks are represented by regular triangles, not by the Catmull-Clark scheme. We report the performance of different methods in terms of culling ratio and the overall rendering performance. We measure the culling ratio only with quads represented by the Catmull-Clark scheme.

On the other hand, we report the overall performance improvements while handling triangles and quads. This is chosen mainly because it is rather difficult to measure runtime performance only with the Catmull-Clark scheme. Reporting the overall performance in this manner may be also useful, since many scenes in practice have a mix of triangles and quads. Note that the overall performance improvement of our method reported here will be lower than that measured only with the Catmull-Clark scheme, since we simply use the common approaches using AABBs for handling ray-triangle tests. For storing the subdivided meshes, we use our vertex ordering for all the tested methods; there is no difference in terms of the mesh representation.

5.2. Comparisons and Analysis

As the subdivision level increases, we can achieve more smooth and visually pleasing results (Fig. 9). Table 1 shows computed subdivision levels given a maximum error bound between CCSM and its limit surface, i.e. 0.001, based on Eq. 2. We get two to four subdivision levels for our tested benchmarks given the error bound. These subdivision levels result in 10 M to 183 M quads from their control meshes.

We measure how much performance improvement our method TSS BVHs w/ PSC achieves over TSS BVHs w/o PSC and AABB BVHs (Fig. 10). Our method achieves up to three times improvement over AABB BVHs, and its improvement goes higher as we use higher subdivision levels. This improvement is mainly thanks to higher culling ratios and lower memory requirement. Specifically, lower-level AABB BVHs take higher memory space over our TSS BVHs, and thus require more memory swaps than our representation. For example, our TSS BVHs takes 25 MB, when low-level AABB BVHs takes 1.6 GB for the museum scene.

Using parallel slab based culling, PSC, achieves runtime improvement up to 149% over our method that does not use PSC. On average, we get 25% and 60% overall performance improvement for the dragon and museum scenes, respectively. Note that

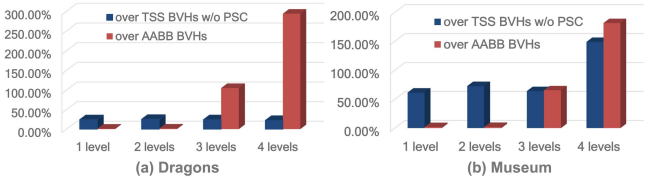

Figure 10: *Overall performance improvement of our method TSS BVHs w/ PSC over our method w/o PSC and the AABB BVHs, as the function of the subdivision level. The overall performance is measured both with quads generated by the Catmull Clark scheme, and input triangles representing the scene.*

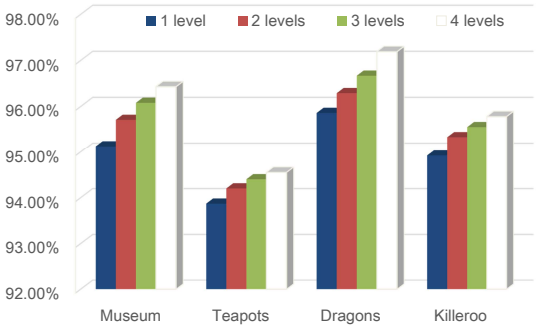

Figure 11: *Intersection tests culling ratios of our method TSS BVHs w/ PSC with different subdivision levels. Culling ratios are measured only with the Catmull-Clark scheme.*

these performance improvement variations are mainly due to varying numbers of triangles intersected with rays under the study with different benchmarks.

Memory requirement. We measure memory requirements of different methods with the virtual museum benchmark; other benchmarks show similar trends. Fig. 1 shows how much memory requirements different methods have as the subdivision level increases. Different methods share the same data structures such as the high-level BVH, vertex/edge/face tables of maintaining subdivided meshes, while different methods have different low-level BVHs. Our TSS BVHs use only one float variable for each level of low-level BVHs to reserve the distance bound, while AABB BVHs need six float variables to define an AABB of its BVH node.

Our method has linear storage complexity as a function of the tree depth, while prior methods have exponential ones. This difference in terms of storage complexity results in drastic savings for our method compared to AABB BVHs, as shown in the right of Fig. 1. For example, our low-level TSS BVHs require only 12.6 MB, 25.2 MB, 37.8 MB, and 50.4 M from the first and to fourth subdivision levels. This results in 30:1, 63:1, 170:1, and 513:1 memory reductions from the first to forth subdivision levels, roughly two times more reductions as we get one more subdivision level. When considering all the data structures including the subdivided meshes, our method using low-level TSS BVHs achieves about 3:1 memory reductions over the prior method using AABB BVHs.

Culling ratios. We use two types of culling ratio measures. The first one, ray culling ratio, is to measure how well a culling method culls out non-intersection rays given all the rays generated from a rendering method. This is a useful definition in terms of analyzing the behavior of ray tracing at a high level. The second measure that we use is called intersection test culling ratio, which measures how well a culling method culls out non-colliding intersection tests given all the intersection tests that we generate to perform a rendering method. This measure is a more direct metric to see the culling efficiency in terms of culling out intersection tests. The main reason why we have such two different, but similar measures of culling ratios is that processing a ray may require varying numbers of intersection tests between BVs and the ray for computing the first intersection point between the ray and the subdivision mesh.

When we use the naive AABB BVHs, the ray culling ratio of the high-level AABB BVHs is 47 % on average. The ray and intersection test culling ratios of the low-level AABB are 16.86 % and 93 % on average, respectively. Note that intersection test culling ratios are usually higher than their corresponding, ray culling ratios by definitions of these culling ratios. While the low-level AABB BVHs achieves high intersection test culling ratio, they require excessive amount of memory space, as mentioned before. On the other hand, by adopting our low-level TSS BVHs w/o PSC, we use much less memory space and, furthermore, achieve a higher intersection test culling ratio of 97.2%, given four subdivision level. The culling ratio goes down by 0.03% with PSC, but the overall performance improves a lot thanks to its low runtime overhead of using PSC, while it is more conservative, resulting in a lower culling ratio.

We also measure the intersection test culling ratio of our method, TSS BVHs w/ PSC, as the subdivision level goes higher (Fig. 11). Culling ratios increase, because subdivided quads become smaller, and thus computed TSSs bound them more tightly. We also calculate the average curvature [Tau95] for centers of the patches with four subdivision levels, as shown in Table 1. Together with results of Fig. 11, we can see that the culling ratio becomes lower as the curvature increases.

Comparisons with Bezier Clipping (BC). We have conducted additional comparisons with Bezier Clipping (BC) [TFM15]. At the subdivision level four, our TSS BVHs w/ PSC method has five times speedup compared with BC for the low-level culling, and 40%-50% improvement in terms of the overall rendering time.

Adaptive subdivision. Our method works with adaptive subdivision (Fig. 12). We follow many existing techniques with crack stitching methods [MTF03, CLF*03]. Nonetheless, our contribution on the adaptive subdivision is minor, and thus we did not discuss much along this aspect, and show results mainly with uniform subdivision for the sake of simplicity in the paper.

Parallel performance. We measure throughput, the number of ray-quad intersection tests of our method TSS BVHs w/ PSC per sec-

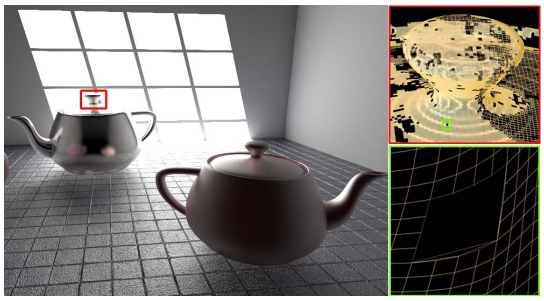

Figure 12: *This image is acquired with adaptive subdivision. We perform a crack stitching method to prevent any cracks. On the right images, we show wire-frame rendering of the computed adaptive subdivision mesh.*

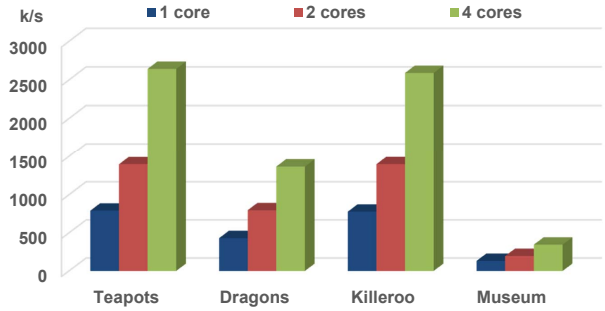

Figure 13: *Ray-quad intersection tests per second with different threads.*

ond with different numbers of threads (Fig. 13). We achieve the highest throughput, about 2.5 million ray-quad intersection tests per second, on the teapot scene, the simplest scene. As we have more complexity such as more quads and additional rendering effects, we achieve less throughput. Once users set the tolerance threshold, we precompute the subdivision level and construct various data structures before the ray-quad intersection tests. As a result, we can achieve a high parallel scalability on ray-quad intersection tests in the experiment.

Construction time. We perform subdivision and build our TSS BVHs for patches on demand, when they are necessary for performing intersection tests. On average with a single-thread version, our method spends 0.5 ms for computing a low-level TSS BVH consisting of four levels of subdivision for each patch. Since the total construction w/ subdivision takes less than 10% of the total rendering time for the virtual museum scene with path tracing in pbrt, we did not optimize our construction method. Nonetheless, it is very important to design an efficient construction method for real-time ray tracing methods (e.g., Embree [WWB*14] and OptiX [Par10]) and for dynamic models. We leave this issue for future work.

Limitations. Our current work has certain limitations. First of all, our current work was not tested with displacement mapping. Fortunately, supporting displacement mapping has been reasonably well known [BWN*15], and our hierarchy can be extended in a similar manner with a larger error bound containing the displaced geometry. Nonetheless, the naive extension could be very conservative and will require more elaborated study for computing tight bounds. As a result, this is left for future work. As mentioned before, designing an efficient construction method for handling dynamic models and real-time ray tracing kernels is an important topic.

6. Conclusion

We have presented a compact BVH representation, TSS BVHs, for efficient ray tracing of subdivision surfaces. Our TSS BVH tightly bounds the limit surfaces of the Catmull-Clark scheme, and keeps only the distance bound information for each depth of the TSS BVH, to reduce the memory consumption. To demonstrate benefits of our method, we have tested our method with different rendering methods and benchmarks. In the tested experiments, we have found that our method achieves high culling ratios up to 97.2% and drastic memory reduction by a factor of two orders of magnitude with three or four subdivision levels thanks to the linear space complexity of our method. These improvements have resulted in overall memory reduction and performance improvement up to a factor of three over the prior AABB BVH based method.

Our tetrahedron-CCSM distance bound theorem can be applied to triangles generated by the Loop subdivision, by considering a triangle to be a quad whose one edge is contracted into a vertex. We, however, found that this naive extension does not give much performance improvement. We would like to design an optimized technique tailored to the Loop subdivision. After using our TSS representations, the subdivided meshes become the major bottleneck in terms of memory requirement. We would like to design a compact mesh representation for representing subdivision schemes in a memory-efficient manner.

Acknowledgements

Prof. Du was supported by National Nature Science Foundation of China (No. 61502130, No. 61472111, No. 61672009), Zhejiang Provincial Natural Science Foundation of China (No. LQ15F020011). Prof. Yoon was supported in part by MSIP/IITP (R0126-16-1108) and MSIP/NRF (2013-067321).

References

[BBLW07] BENTHIN C., BOULOS S., LACEWELL D., WALD I.: *Packet-based Ray Tracing of Catmull-Clark Subdivision Surfaces*. Tech. report, University of Utah, 2007. 2, 3

[BHEK10] BARTOŇ M., HANNIEL I., ELBER G., KIM M.-S.: Precise hausdorff distance computation between polygonal meshes. *Computer Aided Geometric Design 27*, 8 (2010), 580–591. 5

[BWN*15] BENTHIN C., WOOP S., NIEßNER M., SELGARD K., WALD I.: Efficient ray tracing of subdivision surfaces using tessellation caching. In *High Performance Graphics* (2015), pp. 5–12. 2, 3, 10

[Cas12] CASHMAN T. J.: Beyond catmull-clark? a survey of advances in subdivision surface methods. *Comput. Graph. Forum 31*, 1 (2012), 42–61. 1, 3

[CB78] CYRUS M., BECK J.: Generalized two- and three-dimensional clipping. *Computers and Graphics 3*, 1 (1978), 23 – 28. 6

[CC78] CATMULL E., CLARK J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350–356. 2

[CCY06] CHENG F., CHEN G., YONG J.: Subdivision depth computation for extra-ordinary catmull-clark subdivision surface patches. *Advances in Computer Graphics 4035* (2006), 404–416. 4

[CLF*03] CHRISTENSEN P. H., LAUR D. M., FONG J., WOOTEN W. L., BATALI D.: Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes. *Computer Graphics Forum 22*, 3 (Sept. 2003), 543–552. 2, 3, 9

[DS78] DOO D., SABIN M.: Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design 10*, 6 (1978), 356–360. 2

[Fün07] FÜNFZIG V.: *Spherical Techniques and their Applications in a Scene Graph System: Collision Detection and Occlusion Culling*. Cuvillier Verlag, 2007. 5

[HQL*10] HOU Q., QIN H., LI W., GUO B., ZHOU K.: Micropolygon ray tracing with defocus and motion blur. *ACM Transactions on Graphics 29*, 4 (2010), 64:1–64:10. 5

[KBK*10] KIM T.-J., BYUN Y., KIM Y., MOON B., LEE S., YOON S.-E.: HCCMeshes: Hierarchical-culling oriented compact meshes. *Computer Graphics Forum (Eurographics) 29*, 2 (2010), 299–308. 5

[KDS98] KOBBELT L., DAUBERT K., SEIDEL H.-P.: Efficient ray tracing of subdivision surfaces using tessellation caching. In *Proceedings of the Eurographics Workshop* (1998), pp. 69–80. 3

[KK86] KAY T. L., KAJIYA J. T.: Ray tracing complex scenes. In *SIGGRAPH '86* (1986), pp. 269–278. 6

[KSY14] KIM T.-J., SUN X., YOON S.-E.: T-ReX: Interactive global illumination of massive models on heterogeneous computing resources. *IEEE Transactions on Visualization and Computer Graphics 20*, 3 (2014), 481–494. 5

[Loo78] LOOP C.: *Smooth subdivision for surfaces based on triangles.* Master's thesis, University of Utah, 1978. 2

[LS08] LOOP C., SCHAEFER S.: Approximating catmull-clark subdivision surfaces with bicubic patches. *ACM Transactions on Graphics 27*, 1 (2008), 8:1–8:11. 3

[LYTM08] LAUTERBACH C., YOON S.-E., TANG M., MANOCHA D.: ReduceM: Interactive and memory efficient ray tracing of large models. *Comput. Graph. Forum (EGSR) 27*, 4 (2008), 1313–1321. 5

[MTF03] MÜLLER K., TECHMANN T., FELLNER D.: Adaptive ray tracing of subdivision surfaces. *Computer Graphics Forum 22*, 3 (2003), 553–562. 2, 3, 9

[NLMD12] NIEβNER M., LOOP C., MEYER M., DEROSE T.: Feature-adaptive gpu rendering of catmull-clark subdivision surfaces. *ACM Trans. Graph. 31*, 1 (2012), 6:1–6:11. 3

[Par10] PARKER S. G.: Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics 29*, 4 (2010), 66:1– 66:13. 10

[PEO09] PARTNEY A., EBEIDA M. S., OWENS J. D.: Parallel view-dependent tessellation of catmull-clark subdivision surfaces. In *High Performance Graphics* (2009), pp. 99–108. 3

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*. Morgan-Kaufmann, 2004. 7

[Sta98] STAM J.: Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH* (1998), pp. 395–404. 4

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *ACM SIGGRAPH* (1995), pp. 351–358. 9

[TFM15] TEJIMA T., FUJITA M., MATSUOKA T.: Direct ray tracing of full-featured subdivision surfaces with bezier clipping. *Journal of Computer Graphics Techniques (JCGT) 4*, 1 (2015), 69–83. 3, 9

[TLK09] TANG M., LEE M., KIM Y. J.: Interactive hausdorff distance computation for general polygonal models. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2009) 28*, 3 (2009), 74:1–74:10. 5

[WMG*09] WALD I., MARK W. R., GUNTHER J., BOULOS S., IZE T., HUNT W., PARKER S. G., SHIRLEY P.: State of the art in ray tracing animated scenes. *Computer Graphics Forum 28*, 6 (2009), 1691–1722. 1

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph. 33*, 4 (2014), 143:1–143:8. 10

[ZS00] ZORIN D., SCHRODER P.: Subdivision for modeling and animation. *ACM press* (2000). SIGGRAPH 2000 Tutorial N.23 - Course notes. 3