Packet-based Ray Tracing of Catmull-Clark Subdivision Surfaces

Carsten Benthin[*], *Solomon Boulos*[†], *Dylan Lacewell*[‡] *and Ingo Wald*[§]
[*]Intel Corporation [†]School of Computing, University of Utah
[‡]Walt Disney Animation Studios [§]SCI Institute, University of Utah

UUSCI-2007-011

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

July 31, 2007

Abstract:

Subdivision surfaces are the modeling primitive of choice for most production studios. Scanline rendering of subdivision surfaces via tessellation is relatively straightforward due to the in-order nature of the geometry processing, but directly ray tracing them is more complicated. Intersecting a single ray with a patch of faces requires that the patch be subdivided to a sufficiently dense level; in a naive implementation this process is repeated for every ray that hits the same patch. To amortize the cost of subdivision, previous approaches have either pre-tessellated the entire scene and traced it as triangles (often requiring a large amount of memory), or tessellated parts of the scene on demand and cached them. In this paper, we propose an approach that instead uses large packets to amortize the cost of subdivision. The proposed method performs competitively with pre-tessellation, outperforms a single-ray implementation by up to 16× and Pixars PRMan by up to 11×.

Packet-based Ray Tracing of Catmull-Clark Subdivision Surfaces

Carsten Benthin[†] Solomon Boulos[◇] Dylan Lacewell[‡] Ingo Wald[∓†]

[†]Intel Corporation [◇]School of Computing, University of Utah [‡]Walt Disney Animation Studios [∓]SCI Institute, University of Utah

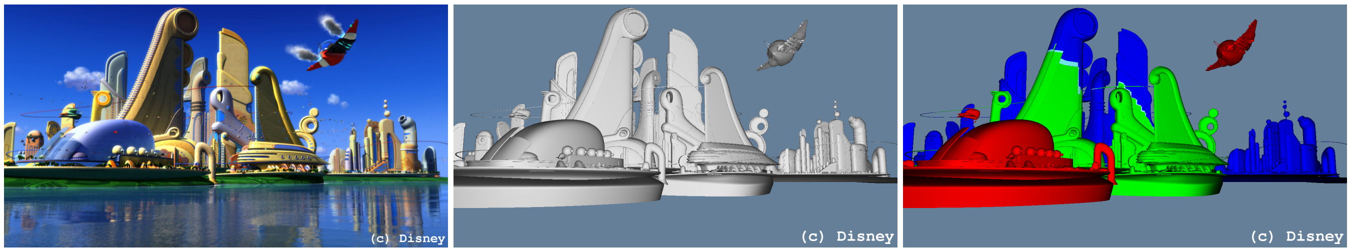

Figure 1: Direct ray tracing of a complex subdivision surface scene containing 1.79M base faces, from Disney's *Meet the Robinsons (c)*. Left: For reference, production rendering using Pixar's PRMan. Center: Pure ray casting of the subdivision surfaces (no water or skydome) with simple shading. Using 5 levels of uniform subdivision, we can render this frame at 2.2 frames per second on a 2.0 GHz dual-CPU Core 2 Quad MacPro (8 cores total, 1384×757 pixels). Right: Using an adaptive subdivision scheme at comparable quality at 4.8 fps (red = 5 subdivisions; green = 4, blue = 3, cyan is crack fixing).

Abstract

Subdivision surfaces are the modeling primitive of choice for most production studios. Scanline rendering of subdivision surfaces via tessellation is relatively straightforward due to the in-order nature of the geometry processing, but directly ray tracing them is more complicated. Intersecting a single ray with a patch of faces requires that the patch be subdivided to a sufficiently dense level; in a naive implementation this process is repeated for every ray that hits the same patch. To amortize the cost of subdivision, previous approaches have either pre-tessellated the entire scene and traced it as triangles (often requiring a large amount of memory), or tessellated parts of the scene on demand and cached them. In this paper, we propose an approach that instead uses large packets to amortize the cost of subdivision. The proposed method performs competitively with pre-tessellation, outperforms a single-ray implementation by up to 16× and Pixar's PRMan by up to 11×.

1 Introduction

Subdivision surfaces have become the standard modeling tool for creating smooth surfaces. Many rendering systems choose to follow the REYES system [9] and render subdivision surfaces through dense tessellation. For scanline methods, this solution is very efficient because the finely tessellated geometry can be thrown away immediately after use. For ray tracing, however, any portion of the scene may be required by any ray, so dense tessellation usually requires a caching system due to the large amount of data.

Despite the large amount of polygons required to render a subdivision surface with adequate smoothness, the initial coarse mesh ("base cage") is relatively compact even for large production scenes. This suggests that there is promise for ray tracing subdivision surfaces without extensive caching if intersection can be made efficient. Geometry caches make intersection more efficient in terms of CPU usage, but also increase memory bandwidth, a resource which historically has grown at a slower rate than processing power. Geometry caches also complicate parallel ray tracing on today's multi-core architectures, since caches have to be synchronized or replicated between processors. Read only access to the base cages in a direct intersection method is therefore more attractive by comparison.

In this paper, we demonstrate that it is possible to efficiently support direct ray tracing of Catmull-Clark subdivision surfaces without a geometry cache. We instead rely on ray packets to provide a similar amortization benefit but with lower memory bandwidth. Due to the depth-first traversal of the implicit hierarchy created during patch subdivision we use only kilobytes of memory, which fits in the processor's cache hierarchy. Compared to either single ray or caching schemes, our packet traversal provides significant speedups even for complicated production level scenes (see Figure 1).

In Section 2, we provide an overview of the specific subdivision rules needed for production scenes. We also discuss recent progress in interactive ray tracing, and specifically discuss ray tracing of higher order surfaces. In Section 3, we detail our approach and discuss implementation details. We demonstrate the effectiveness of our packet based approach in Section 4 where we compare to Pixar's geometry caching system [7], the Razor system [23], and our own single ray implementation.

2 Background

2.1 Subdivision surfaces

Subdivision surfaces define a smooth limit surface by recursively subdividing a polygonal mesh, called either the *control mesh* or *base cage*. Various subdivision rules have been proposed (e.g., [5, 12, 14, 29]), but Catmull-Clark subdivision has become the most common scheme in production rendering since it was adopted by Pixar [10]. This subdivision scheme is well supported in commercial modeling tools such as Maya.

Catmull-Clark subdivision

Catmull-Clark (CC) subdivision surfaces generalize bicubic B-spline surfaces. The limit surface is C^2 smooth except at extraordinary vertices (vertices with valence not equal to 4). For detailed background information, the recent SIGGRAPH course notes [30] are an excellent primer. We will focus on the properties most relevant to our implementation.

To refine one mesh $\mathbf{M^i}$ into another $\mathbf{M^{i+1}}$, the Catmull-Clark subdivision rules are as follows (also see, e.g., [10]):

1. **Create new face points.** For each face, compute it's centroid, and add it as a new face point f^{i+1}.
2. **Create new edge points.** For each edge compute a new edge point e_j^{i+1} by averaging it's two vertices v_j^i, v_{j+1}^i and the midpoints f_j^{i+1} and f_{j+1}^{i+1} of its two neighboring faces.
3. **Refine vertices.** For each vertex v^i, the new vertex v^{i+1} is the weighted average of the adjacent edge points e_j^i, all face points f_j^{i+1} of faces incident to this vertex and v^i itself.
4. **Generate new mesh** by connecting each new face point f^{i+1} to all of its surrounding new edge e^{i+1} and vertex points v^{i+1}.

This already suggests the cost of a subdivision step. Each face processed needs to compute a new vertex, each edge produces another vertex, and each original vertex computes another averaged point. The rules also demonstrate the local nature of subdivision: only a small neighborhood of a face (its 1-ring) is needed to determine the refined faces at the next level.

Though not obvious from the simple rules above, CC subdivision yields a number of nice properties after only one round of subdivision. All faces become quads, and except for boundaries, only one vertex of a quad can have a valence not equal to 4. Also, CC subdivision is invariant under affine transformations and so can be applied component-wise. This means that any data we wish to define at the vertices can be interpolated according to subdivision rules across the surface. Like many researchers, we use these features of CC subdivision to optimize and simplify our implementation.

In addition to the original rules above, later researchers added boundary conditions to handle open-meshes and "sharpness rules" to allow for semi-sharp creases [10]. Alternatively, artists also commonly duplicate edges on the base mesh and place them near one another to simulate a semi-sharp crease. For example, the Disney scene in Figure 1 contains no labeled "crease" edges, but appears to have sharp lines. Pixar's PRMan further supports tagging individual faces of a subdivision mesh as "holes" which are not visible but influence the surrounding geometry. The scene in Figure 1 makes extensive use of this feature.

2.2 Interactive ray tracing

Researchers have demonstrated the feasibility of interactive ray tracing in many settings over the last decade. With the supercomputer work by Parker et al. [18] and then later for commodity hardware with SIMD by Wald et al. [27]. Extending this trend beyond simple hardware gains, Reshetov et al. [21] demonstrated a novel use of packets that allowed kd-trees to achieve amortization beyond SIMD speedups alone. Wald et al. [24, 25] then presented a pair of algorithms that handle dynamic scenes while also including new packet based approaches that go beyond SIMD speedups alone.

In the packet based approaches, gains beyond single ray are only achieved when the rays within packets follow the same traversal path. In the case of primary rays and coherent shadow rays, the previous papers demonstrate excellent amortization results for packet tracing. As noted by Reshetov [20], the speedups gained by recent work in packets may not apply to more general ray tracing. Boulos et al. [4] recently demonstrated, however, that bounding volume hierarchies still allow for both SIMD and algorithmic speedups for both Whitted style and distribution ray tracing [8].

In all of these packet approaches, the gains of using packets is strongly linked to the cost of the repeated operation. If a packet is used to amortize only a few inexpensive operations (e.g., kd-tree plane tests), there is not very much benefit in amortization. In this paper, we demonstrate that ray packets provide an efficient way to amortize subdivision cost, and significantly outperform a single ray implementation.

2.3 Ray tracing higher-order surfaces

Methods for ray tracing higher-order surfaces tend to fall into two broad categories: tessellation and direct intersection.

In a tessellation approach, the higher-order surface is diced up into small polygons (triangles or quads). This works well for streaming geometry in the REYES system [9] but usually requires a caching system to support less coherent ray intersection. In Pharr et. al [19], rays are coherently marched through a coarse scene grid and geometry is tessellated on demand and reused. Christensen et al. [7] achieve high hit rates with a multi-resolution caching scheme, using ray differentials to determine which level of the cache to access.

In contrast to tessellation, researchers have also demonstrated direct ray intersection of smooth surfaces including Bezier surfaces, trimmed NURBS surfaces, and subdivision surfaces. Direct ray tracing of trimmed NURBS surfaces were demonstrated in the Utah Interactive Ray Tracer [15, 18]. More recently, Benthin et al. [2] demonstrated ray tracing of Bezier surfaces and Loop subdivision with a focus on SIMD parallelism. Using multi-core systems it is now possible to directly ray trace trimmed NURBS surfaces interactively [17].

In Whitted's original ray tracing paper [28], support for Catmull-Clark subdivision was provided through a simple recursive method, chosen for simplicity rather than efficiency. Kobbelt et al. [13] present a basis function approach to building a bounding hierarchy over a subdivision surface and directly intersect this surface. Mueller et al. [16] present a method to adaptively ray trace subdivision surfaces. Both methods use single ray implementations and were far from interactive even for relatively simple models.

3 Ray Tracing Subdivision Surfaces

In this section, we explain how we directly intersect a ray with a Catmull-Clark subdivision surface. We first sketch this for a single ray and then discuss important technical issues such as handling of boundaries, termination criteria, and tightness of bounding volumes. Finally, we discuss how to extend this idea to ray packets to improve performance.

3.1 Catmull-Clark patch intersection

We begin by separating the base mesh into individual faces and their 1-rings, which we call "patches". Due to the local support of Catmull-Clark subdivision, the 1-ring is the only information required to subdivide a face.

Next we build an acceleration structure over the initial set of patches (we use a BVH), and begin tracing rays. Once we have determined that a ray has intersected a patch's bounding box (more on that later), we either decide we have reached a sufficient refinement level or continue to subdivide. Subdivision produces four sub-patches assuming we have subdivided the mesh once on input, as is commonly done.

For each of these sub-patches, we recurse until a termination criterion is satisfied. A constant maximum depth is the simplest cri-

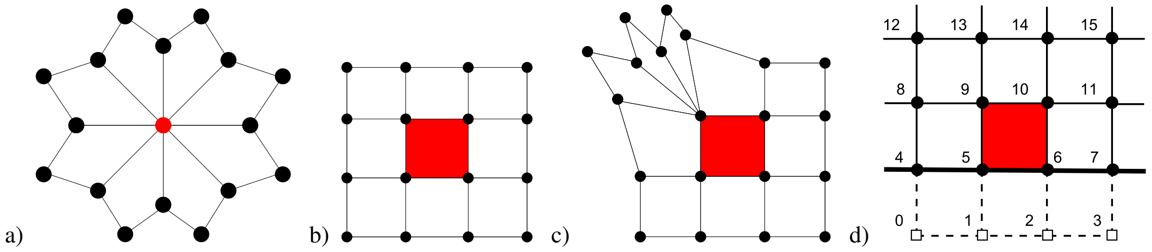

Figure 2: a) 1-ring for a single vertex. The 1-ring of a quad consists of the union of the 1-rings of its four vertices. b) 1-ring for a regular face. The sixteen control vertices can be stored in an efficient 4x4 layout. Explicit connectivity is not required. c) After one subdivision step, a quadrilateral can only contain one irregular vertex. Even for irregular faces, the 4x4 layout is maintained and the 1-ring for the irregular vertex is stored as vector of vertices separately. d) One-ring for a regular boundary face. Vertices 0 through 3 are extrapolated, e.g., $v_0 = 2v_4 - v_8$.

terion, which we use as a baseline, but we also describe a simple adaptive scheme in Section 3.7.

When we reach the maximum depth, a final intersection test is performed. We split the final quad (which may be non-planar) into two triangles and perform intersection tests on each of them. For readers that would like to skip some of the details and simply see an algorithm, please refer to Algorithm 1.

This simple method is straightforward to implement and completely general, although we have not yet presented all the technical details, such as how to determine a tight bounding volume for a patch. The single-ray approach serves as a useful baseline for comparison to our more efficient packet approach in Section 4.

3.2 Efficient data layout

At first glance, it may seem like storing each patch separately is inefficient. However, the vast majority of faces on a mesh are regular (each vertex has valence 4) and form a 4×4 grid as shown in Figure 2 (b). For this common case, connectivity is implicit; we only need to store the 16 vertices. Irregular faces as shown in Figure 2 (c) are relatively rare, and their relative occurrence decreases with the amount of subdivision applied. Furthermore, a face can have at most one extraordinary vertex (EV), so the case illustrated is really the only case, up to rotation and valence of the EV. We only need to store the index and 1-ring of the EV separately.

For quads on the boundary of the mesh, we take advantage of extrapolation to store the "virtual" 1-ring in our efficient 4×4 format (see Figure 2d). Extrapolation produces a B-spline boundary by canceling terms in the bicubic B-spline; this allows us to treat boundary cases as regular subdivision and avoid more complicated control flow.

In order to ensure high data locality we store the 1-ring for each patch in a continuous memory region. Moreover, all vertices are stored using aligned SIMD vectors (each of which is 4 floats). As we wish to support texture coordinates, we allow each vertex to represent a 5-tuple (x, y, z, s, t). Because of the SIMD requirement, we end up storing this as 2 SIMD vectors.

Sometimes, however, rays only need to compute intersections without regard to texture coordinates (e.g. shadow rays). In this case, we subdivide only the first SIMD vector corresponding to the position information. This can lead to almost a $2\times$ speedup for these rays and we make use of this in our implementation when possible (see Section 4 for more concrete results).

3.3 Tight bounding volumes

As with any acceleration structure, tighter bounding volumes produce better traversal results. In this case we seek a tight bounding box for the limit surface of a patch. From subdivision surface theory, the only guarantee that can be made is that the convex hull of the patch bounds the limit surface. By extension, an axis aligned bounding box of the patch also bounds the limit surface. However, the convex hull contains the full one-ring around the current patch, and so is usually very large (for a regular mesh, it's roughly $9\times$ the size of the eventual limit surface). Since the probability of random ray hitting a box is proportional to its surface area, these boxes are not very efficient to use for ray tracing.

In our system, we instead utilize a "look-ahead" scheme, based on a nesting property: since we seek to bound the limit surface, we may replace the bounds for a patch with the combined bounds for its child patches. As part of reading the model, we subdivide each patch a fixed number of times before calculating its bounding box. This greatly reduces the overlap of bounding boxes for neighboring patches and reduces the average number of intersections per packet by up to an order or magnitude for some scenes.

3.4 Amortization using packets

The cost of a subdivision step in the modified Catmull-Clark scheme is fairly high. Even in the regular case (see Figure 2b), we must compute 9 new face vertices, 12 new edge vertices, and 4 new vertex positions. This maps well to an efficient SIMD implementation, but the computational costs are still large.

Extending this traversal algorithm is fairly straightforward for the recursive subdivision method; its behavior with respect to culling and first hit probabilities seems to remain roughly the same as in the triangular case, as can be seen when comparing the culling probabilities in Table 1 with those in [24]. Arguably, the benefit of early culling is higher for subdivision surfaces, as we save more by avoiding a subdivision step than by avoiding a triangle intersection.

As with any packet-based method, the packet size strongly correlates with performance. If the number of rays within the packet is too small, the costs for subdivision cannot be efficiently amortized. On the other hand, a larger number of rays in a packet usually exhibit less coherence and culling efficiency is greatly reduced. In our case a packet size of 64 rays shows the best relation between culling efficiency and amortization, which is in line with previous findings for triangular scenes [24].

3.5 Final intersection

When the termination criteria is reached a final intersection step is performed. Our approach separates the final quadrilateral, which might not be planar, into two triangles which are intersected sequentially. As triangle intersections are typically more expensive than ray-box intersections, we first shrink the packet by determining both the first and the last index of rays in the packet which intersect the current bounding box [24]. Triangle intersections are then

Scene	Killerroo	Forest	Disney
# BVH Traversals	25.09	54.39	72.21
# BVH Leaf Intersections	2.71	7.77	30.19
# Subdivisions	12.54	35.49	803.48
- regular	11.99	35.26	797.55
- irregular	0.54	0.23	5.93
# AABB Culling Tests	79.87%	61.04%	48.98%
- Early Hit Tests	25.82%	26.77%	17.72%
- IA Culling Tests	54.06%	34.27%	31.26%
# Final Intersections	8.64	20.66	119.64
- Active SIMD packets	3.65	2.14	1.71

Table 1: Traversal stats for a ray packet size of 64 rays (max. 16 active SIMD packets) using a predefined subdivision level of 5 (1+4). Each frame is rendered using ray casting (primary rays only) and uses approximately 1M rays.

only performed for these active rays. It is still more likely for a ray to miss a triangle than to hit it, so we perform an inside-outside test before the distance test [1], typically resulting in a 5-8% speedup.

3.6 Pseudo-code and implementation details

Algorithm 1 shows the pseudo code for our patch intersection algorithm. As vertices are stored in a SIMD-friendly layout, the bounding box for each patch can be computed by a sequence of SSE-min/max operations. In combination with the SIMD-implementation of the BVH-first hit and BVH-interval culling test, a patch can be either chosen for subdivision or quickly culled.

During intersection, we must maintain a stack of patches that need to be subdivided. Essentially these patches are like nodes in a standard BVH, but the subdivision rules lead to a branching factor of 4. Each patch is fairly small, however, so the full stack for a reasonable number of subdivisions requires only a few KB of storage. Therefore we don't bother pre-allocating the stack for each thread. As regular and irregular patches have a slightly different memory layout, the subdivision code checks the type and branches to different optimized versions of the "Subdivide" method.

Algorithm 1 Pseudo C++ code for our on-the-fly subdivision intersection. Sub-patches are processed in a depth-first manner which limits the size of the subdivision stack to four times the maximum subdivision level. As the stack typically requires only a few KB of memory, it can be resident within the CPU's L1 cache.

```
while true do
  if stack.isEmpty() == true then
    break
  currentPatch = stack.pop()
  Bounds box = currentPatch.GetBounds()
  if box.CulledByIATest(rayPacket) == true then
    continue
  if box.FirstHitTest(rayPacket) == false then
    if box.AnyRayBoxIntersection(rayPacket) == false then
      continue
  if currentPatch.depth == maximumDepth then
    currentPatch.FinalIntersection(rayPacket)
  else
    Patch sub[4]
    currentPatch.Subdivide(sub)
    stack.push(sub,4);
```

3.7 Adaptive subdivision

A fixed subdivision level is not efficient for scenes with a large range of depth values, where a single resolution is too low for objects near the camera, but too high for distant objects. When rendering from a known camera position it may be possible (although tedious) for an artist to assign a subdivision level to each object; this is not possible in interactive applications.

Some researchers have used adaptive metrics based on ray distance to assign a subdivision level automatically to small units of geometry (e.g., patches in our system). Since the subdivision level is discrete, cracks can appear between adjacent patches with different levels (see Figure 3). Cracking becomes worse if the subdivision level is allowed to vary along a ray; this can produce "tunneling", or cracks between sub-patches within a single initial patch [23].

Like Christensen et al. [7] we avoid tunneling by using a fixed subdivision level for an entire patch, although instead of ray differentials we use a cheaper distance metric: level $= \log_2(\alpha d/D)$, where d is the diameter of the bounding box for the scene, D is the minimum distance from the camera to the center of the patch, and α is a user parameter for the scene. We threshold this level value to determine which of three levels to use: coarse, medium, or fine. If adjacent patches have two different subdivision levels then we stitch cracks on the lower resolution patch as shown in Figure 3.

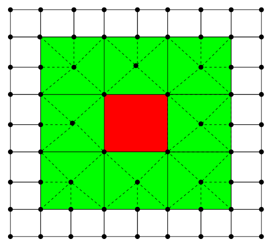

Figure 3: Adjacent patches are subdivided to different depths, so cracks might appear at T-vertices. We fix cracks by creating triangle fans at border sub-patches (green). No crack fixing needs to be done for interior sub-patches (red).

4 Results

In this section, we evaluate our algorithm's performance for a set of test scenes against Pixar's PhotoRealistic Renderman (PRman) [6]. We will compare each different approach under a variety of rendering types: ray casting with simple shading, ray casting with shadows, and general Whitted style ray tracing with 1 bounce of reflections. The ray casting results will establish our best possible performance, while each refinement upon that (shadows and then reflections) further steps along the continuum from coherent towards incoherent ray distributions.

4.1 Comparison methodology

We have chosen a mix of scenes with wildly varying complexity: a simple floating object (the Killerroo, 12K faces), the forest scene from the Razor paper (122K faces), and a large, real-world production scene (1.8M faces) as shown in Figure 4. The Killerroo and Forest scenes are both rendered at a resolution of 1024×1024 pixels, while the Disney scene is rendered at a resolution of 1384×757 pixels to maintain an appropriate aspect ratio.

We have identified five major approaches for rendering these scenes: PRMan's geometry caching system, pre-tessellation, a highly optimized single-ray implementation, the Razor system, and our packet approach. Since pre-tessellation is conceptually different, we will discuss it at the end, and focus first on "true" subdivision surface based ray tracing.

Single-Ray. The single-ray implementation uses the same SIMD-optimized subdivision kernel as the ray-packet approach. Similar

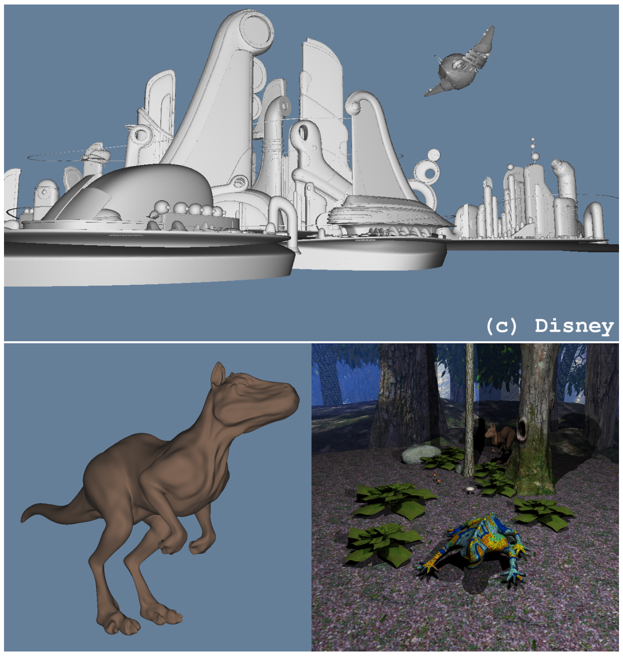

Figure 4: The "killerroo" (12.1K base quads), the "Forest" scene (122K base quads) and the "Disney" scene (1.8M base quads). In these examples, we use a fixed subdivision depth of 5 with 1 subdivision performed as a model preprocess. The scenes then have geometric complexity equivalent to 6.2M, 62.5M, and 918M triangles. On a single 2 GHz core, these examples run at 1.7, 3.92, and 6.98 seconds per frame, casting primary rays and performing subdivision for 8-tuples. For 4-tuples the scenes render at 0.96, 1.92, 3.54 seconds per frame (of approximately 1M rays)

to our packet approach it performs on-the-fly subdivision until a predefined or adaptively selected depth is reached.

PRMan. To compare to PRMan, we place a front-plane in the scene and use a Renderman surface shader which shoots primary rays using the trace shade-op. This also allows us to accurately determine tracing time as PRman reports statistics for the total "shading time" separate from other setup values. Unfortunately, there is no way (that we know of) to definitively compare PRMan's adaptive subdivision quality to our uniform subdivision case. Instead, we have chosen uniform subdivision levels for each scene that appear to produce similar continuity in the surface normal. Those tests implied a higher subdivision level than we would have guessed from simply determining the number of micropolygons generated in relation to the input geometry.

We also attempted to use the new multi-threading feature in PRMan, but only found *reduced* performance when using this feature. Consequently, all comparisons between our approach and PRMan use only a single core for both systems.

Razor. For Razor, we do not have access to the system to run comparisons and thus have to rely on the data available from the original paper. Unfortunately, we also run into the same problem as when comparing to PRman: differences in adaptive subdivision schemes prevent easy direct comparison. We provide a set of subdivision levels to allow the reader to decide but also point out which one we believe matches most closely.

The data in the most recent Razor paper [11] only presents Catmull-Clark subdivision results for the "Forest" scene. The rendering method used for that setup is ray casting with shadows, so we will only have a single point of comparison with the Razor system. We hope to be able to conduct a more detailed comparison with the Razor system at some point in the future. For now, however, most comparisons will be between our single ray implementation, our packet implementation and Pixar's PRman.

Hardware Setup. All of our results are run on a system with 9GB of memory and two Core 2 Quad 2.0GHz processors. None of our scenes utilizes very much of that memory, however (the "Disney" scene uses approximately 1.1GB). As noted earlier, when comparing our results to PRman we will use only a single core. When only comparing our system to itself or to Razor we will use all 8 cores.

4.2 Ray casting with simple shading

We begin by comparing ray casting performance with only simple local shading. As demonstrated in Table 2, packets of rays allow us to perform up to 16.6× faster than a single ray implementation and up to 5.6× faster than a single ray geometry cache. The reason for this improvement is similar to those in previous BVH approaches: the early hit and the interval culling test reduce the traversal cost of the implicit BVH (see Table 1). Note that each traversal step in our subdivision surface intersection is much more expensive than a regular BVH system would use, so the savings are larger.

To demonstrate the large cost of subdivision, we complete the tests for both 4-tuple (position only) and 8-tuple (position and texture) subdivision. As mentioned earlier, working on 4-tuples is significantly more efficient on a 4-wide SIMD architecture than working on 8-wide tuples, translating to significant savings in the subdivision step. For the single-ray implementation—which is dominated by the subdivision cost—this translates to a 3× higher performance; for the packet code—which partially hides the subdivision cost—the savings is somewhat less, but still significant, at roughly 2×.

Even for the more expensive 8-tuples, however, we are $2.4-5.6\times$ faster than PRMan geometry caching, and about $5-15\times$ faster than single ray traversal. In particular, our performance advantage *increases* for realistically complex scenes, making us outperform PRMan in the Disney scene by 5.6× and 11× for 8-tuples and 4-tuples, respectively. Though production rendering obviously *does* need the texture information, the faster 4- tuple case can still be applied for shadow rays which may dominate the total number of rays during rendering.

Scene	Absolute time			speedup over	
	packet	single	PRMan	single	PRMan
4-tuple					
Killerroo	0.96	5.50	4.23	5.7x	4.4x
Forest	1.92	5.62	19.43	2.9x	10.1x
Disney	3.54	37.70	39.03	10.7x	11.0x
8-tuple					
Killerroo	1.70	16.40	4.23	9.7x	2.4x
Forest	3.92	19.47	19.43	5.0x	5.0x
Disney	6.98	115.67	39.03	16.6x	5.6x

Table 2: Ray casting performance in seconds per frame for 1M rays with 4-tuple and 8-tuple subdivision, respectively (using 1 core). For both packet and single-ray implementations we use a predefined subdivision level of 5 (1+4) for all scenes to match the visual quality of the PRMan renderings.

4.3 Ray casting with shadows

Primary rays, however, tend to exhibit higher coherence than other types of rays. In Table 3, we investigate our performance for secondary rays with simple shadows from 2 point light sources. The results demonstrate that packets are already beginning to lose some ground in performance as compared to both single ray and PRman.

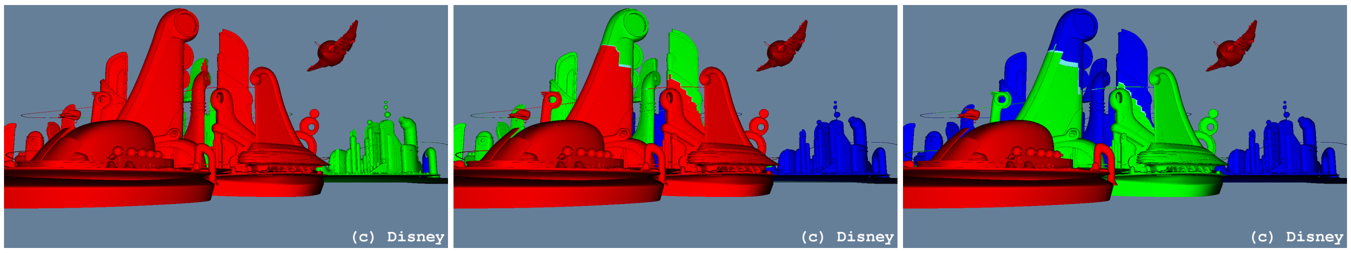

Figure 5: The "Disney scene" rendered using our adaptive termination criterion. In these examples, we set the maximum subdivision depth to 5 with 1 subdivision performed as a model preprocess. With all 8 cores these examples run at 2.67, 3.14, and 4.78 fps, casting primary rays and performing subdivision for 4-tuples. For 8-tuples the scenes render at 1.2 fps, 1.4 fps, and 2.0 fps. By comparison, uniform subdivision runs at 2.2 and 1.1 fps for 4-tuple and 8-tuple subdivision, respectively.

The loss of performance versus single rays is due simply to coherence, while the performance loss to PRman is due to both coherence and PRman's reuse of its cached data for shadow rays. This is the intended use of PRman's geometry cache, however, so this result is expected. For the "Disney" scene there are still some areas of large coherence due to large base cages on some of the buildings. This allows the packets to remain more competitive with the single ray and PRman approaches for this scene. We believe this is a useful point to consider as the "Disney" scene is the only one that was "designed" for subdivision while the others are more or less converted versions of triangle scenes.

Scene	Absolute time			speedup over	
	packet	single	PRMan	single	PRMan
Killerroo	4.48	21.43	5.47	4.8x	1.2x
Forest	9.40	30.54	27.99	3.2x	3.0x
Disney	13.35	174.56	53.64	13.1x	4.0x

Table 3: Rendering performance in seconds per frame including shadows from 2 point lights. For all scenes, shadow rays use 4-tuples for subdivision while primary rays use 8-tuples.

4.3.1 Comparison to Razor

This particular setup—ray casting with two point lights—is also the only one that allows for direct comparison to Razor. This scene is the only one presented in the Razor paper that uses Catmull-Clark subdivision. While Razor does provide three renderings of this scene using different quality setups, our system has not yet been extended for distribution ray tracing so we can only compare to the ray casting with shadows case.

We modify our standard test case of 1024×1024 rays to match the original Razor test of 512×512. In Table 4 we demonstrate performance for this test for a number of subdivision levels (for completeness). Due to the subdivision performed on input, we believe that a subdivision level of 3 matches the Razor subdivision amount most closely.

Level	Absolute time		speedup
	packet	Razor	
1	13.81	.82	16.8x
2	8.69	.82	10.6x
3	4.29	.82	5.2x
4	2.23	.82	2.7x

Table 4: Ray casting with shadows for the Forest scene used by Razor. The frame size is 512×512 using 8-cores with shadows from two point lights. All values are frames per second and the Razor result has been scaled to match our 2.0GHz processor.

As compared to Razor [11], we are using a similar processor but at a lower clock rate (2.0GHz vs 2.66GHz). To take into account this frequency difference, we have scaled their result to match our clock. Depending on which level of subdivision is chosen, we are anywhere between 2.7 and $16.8\times$ faster than Razor for this setup. Though the entire comparison is too "apples-and-oranges" to be conclusive, the overall point is that our approach is at least highly completive and apparently significantly faster. In particular we note that this is only one scene and that Razor is designed to be a distribution ray tracer not a ray caster with point light shadows.

4.4 Whitted style ray tracing

While primary and shadow rays are commonly believed to be highly coherent, ray distributions including specular reflections may behave quite differently [4, 20]. While our system does not have actual production shaders to compare to, we have included a port of the diffuse/specular model used by Boulos et al. [4]. We have currently only focused on single bounce reflections, but hope to do more extensive secondary ray comparisons in the future.

As can be seen from Table 5 we outperform PRMan by about $1-2.5\times$. As expected, PRman's geometry cache performs better and better in relation to our approach as coherence decreases. Compared to our single-ray implementation, the packet performance advantage is roughly $3-8\times$. This is somewhat lower than for primary rays, but clearly indicates that the our approach is not limited to primary rays only. In particular, we maintain an $8\times$ performance advantage even for the Disney scene, in the presence of both less coherent packets *and* incredibly fine geometry (adaptive subdivision is not used).

Scene	Absolute time			speedup over	
	packet	single	PRMan	single	PRMan
Killerroo	7.70	38.08	7.58	4.9x	1.0x
Forest	21.28	66.99	52.63	3.1x	2.5x
Disney	33.88	268.02	66.59	7.9x	2.0x

Table 5: Rendering performance in seconds per frame including shadows from 2 point lights and 1 bounce reflections. Note that shadow rays are cast for both primary and reflected hit points.

4.5 Adaptive subdivision.

In comparing to PRman and our single ray implementation, we have only considered uniform subdivision so far. To demonstrate the possible benefit that adaptive subdivision may provide, we compare uniform subdivision to our simple adaptive heuristic for varying subdivision depths in Table 6. We have chosen to focus on the Disney scene as it has enough depth variability to allow for a useful demonstration.

Table 6 also demonstrates the linear increase in rendering time for our subdivision method. This is due to ray tracing's logarithmic behavior applied to a scene that grows by a factor of $4\times$ for every subdivision step. Our simple adaptive criteria seems to regain about a $2\times$ speedup which suggests it is only performing approximately

Subdiv	ray casting			shadows		
level	uniform	adaptive	speedup	uniform	adaptive	speedup
2	0.0423	0.0357	1.2x	0.1117	0.0993	1.1x
3	0.0924	0.0528	1.8x	0.2559	0.1719	1.5x
4	0.2155	0.1023	2.1x	0.6061	0.3624	1.7x
5	0.4470	0.2083	2.1x	1.2384	0.7298	1.7x

Table 6: Uniform vs adaptive subdivision depth for the "Disney" scene, using 4-tuple subdivision for ray casting (left half) and ray casting with shadows (right half). The adaptive results correspond to the rendering in Figure 1 and Figure 5 right.

one level of subdivision less than the uniform case. Following the trend in the previous results, adding shadows decreases the overall gain to approximately $1.7\times$ (while not explicitly shown in the table, adding reflections decreases it further to approximately $1.5\times$).

As we can see from the chosen levels (Figure 1 and Figure 5 right), a large portion of the geometry uses the finest level of subdivision (5) as the uniform case. Another reasonable portion uses the medium level (4), and only very distant objects use the coarsest level (3). The crack fixing logic introduces overhead, so our currently modest gains are understandable.

4.6 Comparison to pre-tessellation

For many models, it may be feasible to simply subdivide the model as a preprocess and directly ray trace the resulting triangles. An obvious question for our on-the-fly scheme is how close *rendering* performance compares to ray tracing a pre-tessellated result.

Since each additional level of subdivision quadruples the triangle count of the pre-tessellated model, we have performed this comparison only with a relatively coarse base mesh–the Killerroo. At 12K base quads, the equivalent number of triangles after 3 levels of subdivision is already 1.55M triangles ($12K * 4^3 * 2$).

As can be seen from Table 7, our implementation is somewhat slower than tracing a pre-tessellated model. This is not surprising, since we eventually intersect *exactly* the same triangles, but have to generate them on the fly. Also, the BVH over the base cages is naturally looser and has more subtree overlap than in the tessellated case. With this in mind, the performance difference is surprisingly small (5-70% depending on subdivision amount). This is particularly interesting when we consider that the memory use for our approach is only a small fraction of that used for tessellation (see caption for Table 7). For "real" scenes like the Disney scene, full pre-tessellation would far exceed available memory.

subdiv.	number of	Frames per second		speedup
level	triangles	tessellated	direct	
1	97K	10.4	9.87	1.05
2	398K	6.15	4.99	1.23
3	1.55M	3.17	2.19	1.44
4	6.23M	1.75	1.04	1.68

Table 7: Our direct Catmull-Clark subdivision surface ray tracing method vs. ray tracing a pre-tessellated model, for the Killerroo scene with ray casting and 4-tuple subdivision. For every subdivision step the number of triangles quadruple. Assuming a (rather low) size of 40 bytes per triangle, pre-tesselation for this (rather simple) model would require more than 240 MB storage space; for the Disney scene, it would be roughly 40GB triangle data alone.

5 Discussion

In this paper, we have presented an approach to ray tracing subdivision surfaces using packets of rays. For the sake of simplicity, we have so far ignored several important issues such as how to support displacement maps, crease surfaces, or animated models. We now briefly discuss some open problems with our current approach and potentially interesting design alternatives.

Adaptive subdivision using a more advanced metric would probably provide us with greater performance gains than we have already demonstrated. As pointed out in Section 4, we needed to use a subdivision depth of 5 in order to match the quality that PRMan produced for the Disney scene. While our performance when compared to PRman for this scene is still approximately $5\times$ faster for ray casting, we believe that a more advanced adaptive termination criterion would help us gain another factor of $2-4\times$. Our first results with adaptive subdivision already provide a factor of $2\times$.

The limit surface of a Catmull-Clark subdivision surface is usually a regular bicubic b-spline. It might useful to exploit this property to gain further performance. Each subdivision step in the b-spline basis should be less expensive than the more general Catmull-Clark subdivision step and is very amenable to SIMD parallelization. However, around an extraordinary vertex there will always be a small area that cannot be represented in the b-spline basis (see [22]). This limitation also holds in regions where edges or vertices have crease weights assigned to them.

Displacement maps are commonly used in production to add high frequency detail to subdivision surfaces. Subdivision modeling tools like Mudbox (from Skymatter Ltd.) can export a detailed mesh as a low resolution base cage plus a displacement map. All that is required to support displacement maps in our approach is a bound for a displaced patch; caching systems like PRMan have the same requirement. It may be possible to compute sufficiently tight bounds for a displaced patch *without* completely tessellating it.

Creases and other subdivision rules would expand the set of scenes our approach can handle. We believe it would be straightforward to integrate crease rules. It also seems possible to extend our method to support Loop subdivision or other rules; the basic idea of testing a packet against a dynamically subdivided patch would remain valid.

Tighter bounding volumes might be obtained using more advanced methods (see e.g. [3, 13]). We believe this could improve performance, however, we have not yet experimented with adding support for this into our framework.

Memory consumption for our efficient 4x4 control point layout is higher than typical mesh structures which share vertices. In order to ray trace scenes which would not fit into memory, it might be necessary to do the conversion to the 4x4 layout on the fly, before the actual patch intersection step. We have ignored this, however, as even the Disney scene uses approximately 1GB of memory after being subdivided once.

Dynamic models are particularly interesting for any on-the-fly scheme. Under the assumption that the topology of the control mesh remains unchanged, only the acceleration structure over the (comparatively few) base cages has to be rebuilt. The typical bottleneck of building an acceleration structure over the tessellated triangle equivalent is avoided. However, here again the question of how to quickly compute a tight bounding box for each patch requires further investigation.

Distribution ray tracing is the real reason for using ray tracing and will likely generate a majority of rays in future applications. Distribution ray tracing is likely to be less coherent than the kind of rays we have focused on in this paper, and supporting them efficiently will require further investigation. However, this is not much different from cache-based tessellation schemes which also fail for completely random rays. The ray differential caching approach [7] avoids this performance deterioration, and we are interested in seeing how we might apply a similar idea to our approach.

Complex shading often accounts for a large portion of the rendering time for production scenes. The REYES algorithm amortizes shading over a grid of micropolygons, and also determines grid-based derivatives, which are useful for determining texture filter size (PRMan also uses ray differentials to determine filter size for rays). We hope to extract similar benefits from surface patches and ray packets. Real world shaders often use large amounts of stored texture; texture caching is largely orthogonal to our system, although ray packet size and ordering will affect the coherence of texture access.

6 Summary and Conclusion

We have proposed an approach to ray tracing subdivision surfaces using on-the-fly tessellation. Whereas other systems like Razor or PRMan amortize the cost for patch subdivisions by caching geometry, we instead use large packets of rays coupled with an efficient traversal algorithm. Our approach amortizes the cost of subdivision at least as well as geometry caching, and in addition allows for all the other advantages of packet techniques. Consequently, we not only need less memory for caches, but are also faster than both Razor and PRMan.

For scenes with varying depth complexity, we have proposed an adaptive subdivision method. Though crack fixing adds complexity to the system, for the Disney scene it provides additional speedups of up to 2.1×. Adaptive subdivision also becomes particularly interesting when considering packets of less coherent secondary rays, as these can use a coarser scene representation [7, 11].

Performance-wise, our system outperforms both Razor and PRMan by up to 5.2× and 5.6×, and a single-ray implementation of the same algorithm by 16.6×. Compared to pre-tessellated models with pre-built acceleration structures we achieve a roughly competitive performance, but require only a fraction of the memory. Furthermore, we can render even hugely complex scenes which would not fit into memory.

Future Work. Potential extensions to our system abound. All the issues raised in Section 5 are worthy of closer investigation. Among those, we are particularly interested in supporting displacement maps and creases, which are very important for real-world production rendering. Apart from that, the biggest issue for supporting real-world rendering is support for "real" shaders and, in particular, for secondary rays. Since at least some of these will be less coherent than the ones we have considered in this paper, supporting those efficiently will require further investigation.

Acknowledgments

We would like to thank Walt Disney Animation Studios for providing us with a scene from Disney's *Meet the Robinsons*. For modeling and providing us the Razor scene we would like to thank Jeffery A. Williams, Headus (Metamorphosis), Phil Dench, Martin Rezard, Jonathan Dale, and the DAZ studio team.

References

[1] C. Benthin. *Realtime Ray Tracing on current CPU Architectures*. PhD thesis, Saarland University, 2006.

[2] C. Benthin, I. Wald, and P. Slusallek. Interactive Ray Tracing of Free-Form Surfaces. In *Proceedings of Afrigraph*, pages 99–106, November 2004.

[3] J. Bolz and P. Schröder. Rapid evaluation of catmull-clark subdivision surfaces. In *Web3D '02: Proceeding of the seventh international conference on 3D Web technology*, pages 11–17, New York, NY, USA, 2002. ACM Press.

[4] S. Boulos, D. Edwards, J. D. Lacewell, J. Kniss, J. Kautz, P. Shirley, and I. Wald. Packet-based Whitted and Distribution Ray Tracing. In *Proceedings of Graphics Interface 2007*, May 2007.

[5] E. Catmull and J. Clark. Behavior of recursive division surfaces near extraordinary points. In *Computer Aided Design 10(6)*, pages 350–355, 1978.

[6] P. H. Christensen, J. Fong, D. M. Laur, and D. Batali. Ray tracing for the movie 'Cars'. In *Proc. IEEE Symposium on Interactive Ray Tracing*, pages 1–6, 2006.

[7] P. H. Christensen, D. M. Laur, J. Fong, W. L. Wooten, and D. Batali. Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes. In *Computer Graphics Forum (Eurographics 2003 Conference Proceedings)*, pages 543–552. Blackwell Publishers, September 2003.

[8] R. Cook, T. Porter, and L. Carpenter. Distributed Ray Tracing. *Computer Graphics (Proceeding of SIGGRAPH 84)*, 18(3):137–144, 1984.

[9] R. L. Cook, L. Carpenter, and E. Catmull. The REYES Image Rendering Architecture. *Computer Graphics (Proceedings of ACM SIGGRAPH 1987)*, pages 95–102, July 1987.

[10] T. D. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 85–94, July 1998.

[11] P. Djeu, W. Hunt, R. Wang, I. Elhassan, G. Stoll, and W. R. Mark. Razor: An Architecture for Dynamic Multiresolution Ray Tracing. Technical report, University of Texas at Austin Dep. of Comp. Science, 2007. Conditionally accepted to ACM Transactions on Graphics.

[12] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. In *Computer Aided Design 10(6)*, pages 356–360, 1978.

[13] L. Kobbelt, K. Daubert, and H.-P. Seidel. Ray Tracing of Subdivision Surfaces. *Proceedings of the 9th Eurographics Workshop on Rendering*, pages 69–80, 1998.

[14] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[15] W. Martin, E. Cohen, R. Fish, and P. Shirley. Practical Ray Tracing of Trimmed NURBS Surfaces. *Journal of Graphics Tools: JGT*, 5:27–52, 2000.

[16] K. Mueller, T. Techmann, and D. Fellner. Adaptive Ray Tracing of Subdivision Surfaces. *Computer Graphics Forum (Proceedings of Eurographics '03)*, pages 553–562, 2003.

[17] S. M. Oliver Abert, Markus Geimer. Direct and Fast Ray Tracing of NURBS Surfaces. In Wald and Parker [26].

[18] S. G. Parker, W. Martin, P.-P. J. Sloan, P. Shirley, B. E. Smits, and C. D. Hansen. Interactive ray tracing. In *Proceedings of Interactive 3D Graphics*, pages 119–126, 1999.

[19] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering Complex Scenes with Memory-Coherent Ray Tracing. *Computer Graphics*, 31(Annual Conference Series):101–108, Aug. 1997.

[20] A. Reshetov. Omnidirectional ray tracing traversal algorithm for kd-trees. In Wald and Parker [26], pages 57–60.

[21] A. Reshetov, A. Soupikov, and J. Hurley. Multi-Level Ray Tracing Algorithm. *ACM Transaction on Graphics*, 24(3):1176–1185, 2005. (Proceedings of ACM SIGGRAPH 2005).

[22] J. Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 395–404, July 1998.

[23] G. Stoll, W. R. Mark, P. Djeu, R. Wang, and I. Elhassan. Razor: An Architecture for Dynamic Multiresolution Ray Tracing. Technical Report 06-21, University of Texas at Austin Dep. of Comp. Science, 2006.

[24] I. Wald, S. Boulos, and P. Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 26(1):1–18, 2007.

[25] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Trans-*

actions on Graphics, 25(3):485–493, 2006. (Proceedings of ACM SIGGRAPH).

[26] I. Wald and S. G. Parker, editors. *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 2006.

[27] I. Wald, P. Slusallek, C. Benthin, and M. Wagner. Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum*, 20(3):153–164, 2001. (Proceedings of Eurographics).

[28] T. Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–349, 1980.

[29] D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Computer Graphics*, volume 30, pages 189–192, 1996.

[30] D. Zorin, P. Schroeder, T. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for modeling and animation. SIGGRAPH course notes, 2000.