Packet-based Ray Tracing of Catmull-Clark Subdivision Surfaces

Carsten Benthin[†] Solomon Boulos[◇] Dylan Lacewell[‡] Ingo Wald[∓†]
[†]Intel Corporation [◇]Stanford University [‡]Walt Disney Animation Studios [∓]SCI Institute, University of Utah

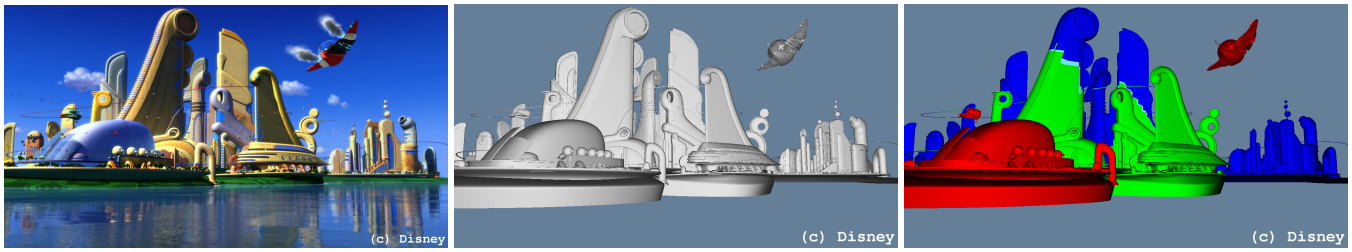

Figure 1: *Direct ray tracing of a complex subdivision surface scene containing 1.79M base faces, from Disney's* Meet the Robinsons ©. *Left: For reference, production rendering using Pixar's PRMan. Center: Pure ray casting of the subdivision surfaces (no water or skydome) with simple shading. Using 5 levels of uniform subdivision, we can render this frame at 2.2 frames per second on an 8-Core 2.0 GHz Core 2 Quad MacPro (*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

Efficient ray tracing of subdivision surfaces is an important problem in production rendering, and for interactive applications in the near future. The current hardware trends for both CPUs and GPUs suggest that compute power is outpacing bandwidth. Despite this, current approaches for ray tracing subdivision surfaces favor geometry caches or full pre-tessellation. We demonstrate that directly ray tracing subdivision surfaces using ray packets uses much less bandwidth, while still providing amortization benefits. Our proposed method performs competitively with pre-tessellation even on current hardware, outperforms a single-ray implementation by up to $16\times$ and Pixar's PRMan 13.0 geometry caching by up to $23.1\times$.

1 Introduction

Subdivision surfaces (Subds) have become the most widely used organic modeling primitive in the animation industry, and we expect games to follow this trend in the future. The advantage of subds is that smooth surfaces can be described at a very coarse level, with the generation of a temporary, dense polygonal mesh deferred until late in the rendering pipeline (e.g., as in a REYES renderer [CCC87]). Deferred tessellation places the strain on compute power rather than on the memory system, which will have increasing benefits over geometry caching as compute power continues to increase more quickly than memory bandwidth.

We also expect future games to demand increasing amounts of ray tracing. At first ray tracing might be used for occlusion queries (e.g. exact hard shadows, easy soft shadows, collision detection), and later for indirect lighting effects such as reflections and global illumination. Not having the ability to ray trace subds will quickly become a barrier to the adoption of ray tracing in games.

In this paper, we show that it is possible to efficiently support direct ray tracing of subds, specifically Catmull-Clark surfaces, without a geometry cache. We instead rely on ray packets to provide a similar amortization benefit but with lower memory bandwidth; we use only kilobytes of memory per packet, which fits in the processor's local cache hierarchy. Compared to either geometry caching schemes or single-ray intersection, our packet traversal provides significant speedups even for real-world film production scenes (see Figure 1).

2 Background

2.1 Subdivision surfaces

Subdivision surfaces define a smooth limit surface by recursively subdividing a polygonal mesh, called the *control mesh*. Various subdivision rules have been proposed (e.g., [DS78; CC78; Loo87; ZSS96]), but Catmull-Clark subdivision has become the most common scheme in production rendering since it was adopted by Pixar [DKT98].

Catmull-Clark subdivision

Catmull-Clark (CC) subdivision surfaces generalize bicubic B-spline surfaces. For detailed background information, the recent SIGGRAPH course notes [ZSD*00] 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., [DKT98]):

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}.

Though not obvious from the rules above, CC subdivision has a number of useful properties for purposes of efficient ray tracing. After one round of subdivision all faces become quads, and except for boundaries, at most one vertex of a quad can be an extraordinary vertex (EV), with valence not equal to 4. Also, CC subdivision is invariant under affine transformations, which means any number of data "channels" (e.g., position plus texture coordinates) can be stored at the vertices, and every channel can be subdivided independently. Most importantly for our purposes, subdivision is local: only a small neighborhood of adjacent faces (the 1-ring) is need to subdivide a face.

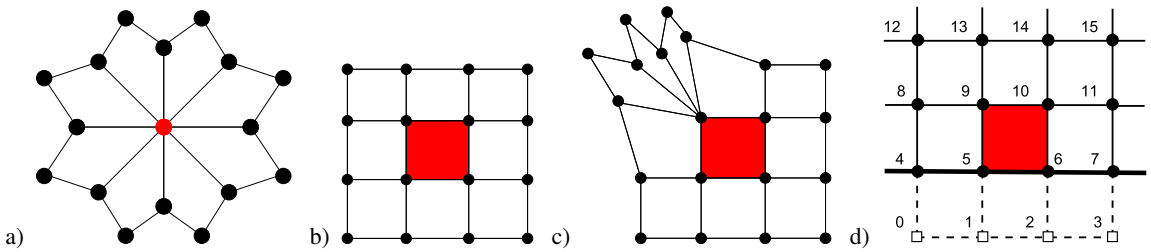

Figure 2: *a) 1-ring for a single vertex. The 1-ring of a quad consists of the union of the 1-rings of its four vertices. b) 1-ring for a regular face. The sixteen control vertices can be stored in an efficient 4x4 layout. Explicit connectivity is not required. c) After one subdivision step, a quadrilateral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

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. [PMS*99] and then later for commodity hardware with SIMD by Wald et al. [WSBW01]. Extending this trend beyond simple hardware gains, Reshetov et al. [RSH05] demonstrated a novel use of packets that allowed kd-trees to achieve amortization beyond SIMD speedups alone. Wald et al. [WIK*06; WBS07] 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 [Res06], the speedups gained by recent work in packets may not apply to more general ray tracing. Boulos et al. [BEL*07] recently demonstrated, however, that bounding volume hierarchies still allow for both SIMD and algorithmic speedups for both Whitted style and distribution ray tracing [CPC84].

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 [CCC87] but usually requires a caching system to support less coherent ray intersection. In Pharr et al. [PKGH97], rays are coherently marched through a coarse scene grid and geometry is tessellated on demand and reused. Christensen et al. [CLF*03] 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 [PMS*99; MCFS00]. More recently, Benthin et al. [BWS04] 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 [OA06].

In Whitted's original ray tracing paper [Whi80], support for Catmull-Clark subdivision was provided through a simple recursive method, chosen for simplicity rather than efficiency. Kobbelt et al. [KDS98] present a basis function approach to building a bounding hierarchy over a subdivision surface and directly intersect this surface. Mueller et al. [MTF03] present a method to adaptively ray trace subdivision surfaces. Both methods use single ray implementations and were far from interactive.

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 the extension to ray packets for improved 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". Note that the 1-ring of a quad consists of the union of the 1-rings of its four vertices, see Figure 2 (a). 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.

For each of these sub-patches, we recurse until a termination criterion is satisfied. A constant maximum depth is the simplest criterion, 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. Note that we do not retain any of the subdivided patches in memory once the ray terminates.

3.2 Efficient data layout

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 and 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 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.

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 or 8-tuple.

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. As detailed in Section 4, this can lead to almost a $2\times$ speedup for these rays.

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 a 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 children. As part of reading the model, we subdivide each patch a fixed number of times before calculating its bounding box. (We do not retain the child patches in memory, only the bounding box and the base patch). The tighter bounding boxes greatly reduce the overlap of 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 the BVH packet traversal algorithm [WBS07] is fairly straightforward for the recursive subdivision method. The behavior with respect to culling and first hit probabilities seems to remain roughly the same as in the triangular case (comparing the culling probabilities in Table 1 with those in [WBS07]). The benefit of early culling is also higher for subdivision surfaces, as we save more by avoiding a subdivision step than avoiding a triangle test.

Scene	Killerroo	Forest	Disney
# BVH Traversal Steps	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.*

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 also in line with previous findings for triangular scenes [WBS07].

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 tests are typically more expensive than ray-box tests, 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 [WBS07]. Triangle intersections are then 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 [Ben06], 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.

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.

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 kilobytes of memory, it can be resident within the CPU's cache hierarchy.

```
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);
```

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). With discrete subdivision levels, however, cracks can appear between adjacent patches with different levels. 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 [SMD*06].

Like Christensen et al. [CLF*03] we avoid tunneling by using a fixed subdivision level for an entire patch, although instead of ray differentials we use a cheaper *ad hoc* distance metric: level $= \log_2(\alpha d/D)$, where d is the diameter of the scene bounding box, 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 value to determine which of three levels to use: coarse, medium, or fine. The finest level corresponds to the same value as uniform subdivision, while medium and coarse reduce that value by 1 level each. If adjacent patches have different subdivision levels then we stitch cracks on the lower resolution patch (see Figure 3). Note that our scheme ensures that we never have to stitch patches that are more than one level apart.

This simple adaptive scheme allows our crack fixing logic to be both simple and efficient. While more advanced adaptive subdivision may result in larger gains, we present our method as a simple example of what might be gained through adaptivity.

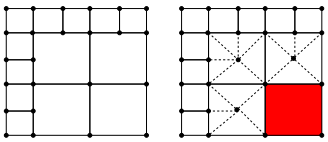

Figure 3: *Left: Adjacent patches are subdivided to different depths, so cracks might appear at T-vertices. Right: We fix cracks by creating triangle fans at border sub-patches. No crack fixing needs to be done for interior sub-patches (red).*

4 Results

In this section, we evaluate our algorithm's performance for a set of test scenes against some competing approaches for rendering subdivision surfaces. We will compare each approach under three rendering types: ray casting with simple shading, ray casting with shadows, and Whitted style ray tracing with 1 bounce of reflections.

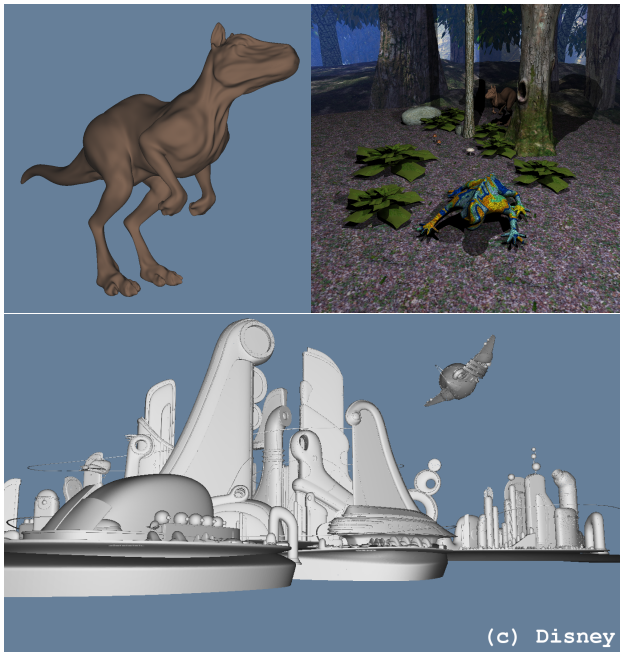

Figure 4: *The "Killerroo" (12.1K base quads), the "Forest" scene (122K base quads) and the "Disney" scene (1.8M base quads). In these examples, we use a fixed subdivision depth of 5 with 1 subdivision performed as a model preprocess. The scenes then have geometric complexity equivalent to 6.2M, 62.5M, 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).*

The ray casting results will establish our best possible performance. Each refinement upon that (shadows and then reflections) will step along the continuum from coherent towards incoherent ray distributions. As with any packet based method, it is expected that performance advantages present in the ray casting case will decrease as the ray distributions become incoherent.

4.1 Comparison methodology

For a fair comparison, we have chosen a mix of scenes with varying complexity: a simple floating object (the Killerroo, 12K base quads), the forest scene from the Razor paper (122K base quads), and a large, real-world production scene (1.8M base quads) 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 for film.

We have identified three competing approaches for ray tracing subdivision surfaces: PRMan's multi-resolution geometry caching [CFLB06], the Razor system [DHW*07], and full pre-tessellation of the surfaces. Since pre-tessellation is conceptually different, we will discuss it in Section 4.7, and focus first on "true" subdivision surface based ray tracing. We also present the results for our simple adaptive subdivision scheme in Section 4.6, which we consider an additional improvement over our basic method.

PRMan. PRMan does not directly trace primary rays from the camera, so we place a transparent "window" in front of the camera to do so using the `trace` shade-op. This also allows us to accurately separate tracing time from setup time as PRMan reports statistics for the total "shading time" which now includes the `trace` shade-op. All time spent in "shading" is essentially ray tracing time.

PRMan uses a multi-resolution geometry cache which uses adaptive subdivision of the surfaces based on the REYES shading rate [CFLB06] and does not have a uniform subdivision mode. Instead of attempting to match their adaptive subdivision method, we have chosen to match their rendering quality even when using uniform subdivision. This comparison results in a large disadvantage for our system, but we feel quality is the only adequate metric for comparison. To further improve our performance, yet retain the same quality requirements, we use our own adaptive subdivision scheme. In Section 4.6 we provide a comparison between our two approaches.

To determine which uniform subdivision level matches the quality of PRMan's adaptive subdivision, we rendered the same scene in both PRMan and our system using a highly specular shader with a high frequency procedural environment map. Using simple diffuse shaders instead would have allowed us to use a lower subdivision level (2 levels lower in most cases), as diffuse shaders obfuscate high-frequency detail.

We attempted to use the multi-threading feature in PRMan 13.0, but found that employing more than 1 thread *reduces* performance. Consequently, all comparisons between our approach and PRMan are done using a single thread for both systems.

Razor. As we do not have access to the Razor system to run comparisons, we rely on the data available from the original paper. The data in the most recent Razor paper [DHW*07] 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 Razor at some point in the future.

Razor's adaptive subdivision scheme is different from ours, just as is the case when comparing to PRMan. We compare our uniform subdivision to Razor's adaptive subdivision, while trying to match the same quality. To do so, we ensured that for a given subdivision level, none of the final triangles in our method exceeded 8 pixels in area (matching the Razor quality metric); however, our subdivision amount applies to all rays, whereas Razor is able to decide a subdivision amount per ray regardless of type. While Razor may choose a higher subdivision amount for some shadow rays, we feel it is unlikely that the amount is vastly different.

4.2 Hardware Configuration

We run all of our results on a system with 9GB of memory. However, our scenes do not utilize much of that memory. For example, the Disney scene (which is the largest) uses approximately 1.1GB for the control mesh after it has been subdivided once and split into patches.

Our system has two Core 2 Quad 2.0GHz processors. As noted earlier, when comparing our results to PRMan we use only a single core for performance reasons. When comparing our system to itself or to Razor, we use all 8 cores.

4.3 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 $16.6\times$ faster than a single ray implementation and up to $5.6\times$ 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.

As mentioned earlier, the size of the tuples being subdivided can greatly increase the cost of subdivision. In particular, on a 4-wide SIMD machine, a 4-tuple is particularly well suited to the architecture. We show the cost for both 4 and 8-tuples for our ray casting results to demonstrate the possible optimization for rays that don't need more than position information (e.g. shadow and other binary visibility queries). It is also interesting to note that for wider SIMD architectures, it may be possible to subdivide more parameters than simply position and a single texture coordinate.

Even for the more expensive subdivision of 8-tuples we are $2.4 - 5.6\times$ faster than PRMan's geometry caching, and about $5 - 15\times$ faster than single ray traversal. In particular, our performance advantage *increases* for realistically complex scenes, as we outperform PRMan for the Disney scene by $5.6\times$ and $11\times$ for 8-tuples and 4-tuples, respectively.

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.*

4.4 Ray casting with shadows

Primary rays 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.

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 geometry 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 control meshes on some of the buildings. It should be noted, however, that this is a "real world" instance of subdivision surface modeling: large and simple control meshes can be used to describe otherwise complicated smooth surfaces. This allows the packets to remain more competitive with the single ray and PRMan approaches for this scene.

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.*

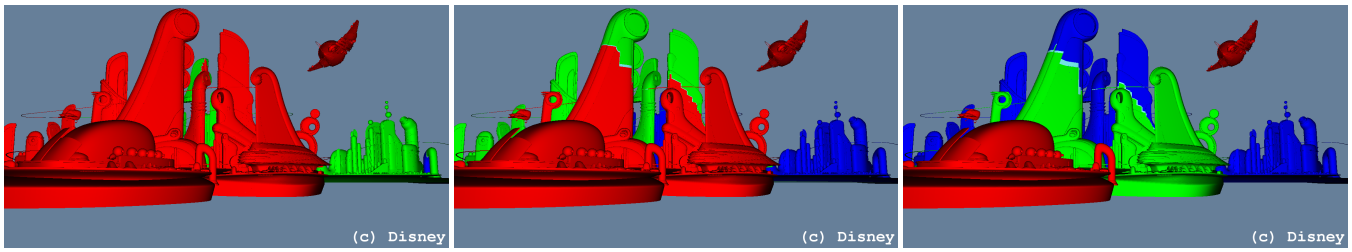

Figure 5: *The Disney scene rendered using our adaptive termination criterion. In these examples, we set the maximum subdivision depth to 5 with 1 subdivision performed as a model preprocess. With all 8 cores these examples run at 2.67, 3.14, and 4.78 fps, casting primary rays and performing subdivision 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.*

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 (based on the visible micropolygon area at level 3).

As compared to Razor [DHW*07], we are using a similar processor but a lower clock rate (2.0GHz vs 2.66GHz). To take into account this frequency difference, we scale 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 for this rendering style. 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.

Level	Frames per second		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.*

4.5 Whitted style ray tracing

While primary and shadow rays are commonly believed to be highly coherent, ray distributions including specular reflections behave quite differently [Res06; BEL*07]. While our system does not have actual production shaders to compare to, we have ported the diffuse/specular model used by Boulos et al. [BEL*07]. 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 multi-resolution 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, packets have an $8\times$ performance advantage over single rays 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.*

4.6 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 in addition to the speedups already demonstrated, 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.

Level	ray casting			shadows		
	uniform	adaptive	speedup	uniform	adaptive	speedup
2	0.04	0.036	1.2x	0.11	0.10	1.1x
3	0.09	0.053	1.8x	0.26	0.17	1.5x
4	0.22	0.10	2.1x	0.61	0.36	1.7x
5	0.45	0.21	2.1x	1.24	0.73	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.*

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 one level of subdivision less than the uniform case on average (i.e. it is choosing the medium level). Following the trend in the previous tables, 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 $1.5\times$).

If we combine our adaptive rendering speedup with the uniform subdivision tables already presented against PRMan, our method is approximately $23.1\times$ faster for ray casting with 4-tuples, $6.8\times$ faster for ray casting with shadows, and $3.0\times$ faster for ray tracing with single bounce reflections. The gains over our single ray implementation are even more pronounced.

As we can see from the chosen levels (Figure 5 right), a large portion of the geometry uses the finest level of subdivision (5) matching 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 but encouraging.

4.7 Comparison to pre-tessellation

For small models it may be feasible to 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 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 can only compare with a relatively coarse base mesh–the Killerroo. At 12168 base quads, the equivalent number of triangles after 4 levels of subdivision is already 6.23M triangles ($12168 * 4^4 * 2$). Assuming a (rather low) size of 40 bytes per triangle, pre-tessellation for this (rather simple) model would 250 MB storage space; for the Disney scene, it would be roughly 40GB of triangle data alone.

4.7.1 Performance

As can be seen from Table 7, our implementation is slower than tracing a pre-tessellated model. As a reminder, our claim is that our algorithm will provide benefits over caching and pre-tessellation in the future as compute power exceeds bandwidth. Given that we are running on a current CPU our lower ray casting performance is not surprising, since we eventually intersect exactly the same triangles, but have to generate them on the fly. Furthermore, the bandwidth required even for the tessellated version of this scene per frame is not enough at current frame rates to be prohibitive on a modern CPU.

Level	number of triangles	Frames per second tessellated	Frames per second direct	slowdown
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. For this setup, our performance is always slower but the slowdown is always less than* $2\times$.

The BVH over the subdivision control meshes 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. For "real" scenes like the Disney scene, full pre-tessellation would far exceed available memory.

4.7.2 Off-cache bandwidth

As our algorithm directly intersects a subdivision surface, the only model data required is the initial patch. Because we recurse in a depth-first manner, only a small amount of stack space for "to be subdivided" patches needs to be maintained. Current CPUs can hold this stack mostly in their L2 cache which significantly reduces off-cache (external memory) bandwidth. In order to measure the off-cache bandwidth we used the *cachegrind* cache profiler [Val07] to track L2 cache misses.

Table 8 shows that direct ray tracing requires a very low and constant off-cache bandwidth. Compared to ray tracing a pre-tessellated scene, it requires $2.4 - 64.7$ times less bandwidth (per frame), depending on the subdivision level.

Level	number of triangles	Bandwidth (MB) tessellated	Bandwidth (MB) direct	reduction
1	97K	5.2	2.2	2.4x
2	398K	18.5	2.2	8.4x
3	1.55M	60.4	2.2	27.5x
4	6.23M	142.4	2.2	64.7x

Table 8: *Required off-cache (main memory) bandwidth for direct Catmull-Clark subdivision surface and ray tracing a pre-tessellated model, for the Killerroo scene with ray casting and 4-tuple subdivision. For high subdivision levels on-the-fly subdivision is able to reduce the bandwidth by a factor of 64.*

On future architectures that have more compute resources than caching, less bandwidth should translate into a significant performance advantage. For now, however, a modern CPU memory hierarchy can easily sustain the required bandwidth for this scene even for the highest level of subdivision.

4.8 Impact of ray packet size

Like any packet method, reduced coherence will reduce the amortization benefits (see Section 4.5 for evidence). As the coherence is typically related to the number of rays per packet we tested our method using a varying ray packet sizes as shown in Table 9. Like previous work [WBS07] we achieve the best performance for a ray packet size of $8 \times 8 = 64$ rays.

Level	packet size 4x4	8x8	16x16	32x32
1	.77 (.89)	1.0 (1.0)	.94 (.87)	.48 (.39)
2	.74 (.80)	1.0 (1.0)	.95 (.90)	.36 (.27)
3	.70 (.75)	1.0 (1.0)	.97 (.92)	.27 (.21)
4	.70 (.76)	1.0 (1.0)	.99 (.93)	.20 (.20)

Table 9: *Ray casting (Whitted style ray tracing) performance for varying ray packet size and subdivision level (Killerroo scene). All results are shown to the normalized performance for* $8x8 = 64$ *rays per packet: Having less than* 64 *rays per packet makes amortization of subdivision less effective, while performance for larger packets suffers under reduced ray coherence. For our setup, a size of* $8x8$ *rays has been determined to be the most efficient.*

5 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 is competitive with geometry caching, and in addition allows for all the other advantages

of packet techniques. Consequently, we not only need less memory than geometry cache approaches, 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 $2.1\times$. Adaptive subdivision also becomes particularly interesting when considering packets of less coherent secondary rays by using a coarser scene representation [CLF*03; DHW*07].

Performance-wise, our uniform subdivision approach outperforms both Razor and PRMan by up to $5.2\times$ and $5.6\times$, and a single-ray implementation of the same algorithm by $16.6\times$. Our adaptive subdivision is even more competitive as it adds an additional factor of $2\times$ on top of these results. Compared to pre-tessellated models with pre-built acceleration structures we achieve roughly competitive performance, but require only a fraction of the storage and bandwidth requirements. We also demonstrated our approach for a complex film scene which would not fit into memory if tessellated.

The compute power of future graphics architectures, regardless of being more CPU or GPU oriented, is expected to grow much faster than their memory bandwidth. This makes it essential to reduce the off-chip bandwidth to achieve optimal utilization. Our approach is well suited for these architectures as it requires only a small amount of on-chip memory per core.

Future Work. Potential extensions to our system abound. We are particularly interested in supporting displacement maps, which are important for real-world production rendering. More advanced methods for adaptive subdivision would further reduce the number of required subdivisions per patch. Apart from that, the biggest issue for supporting real-world rendering is support for complex shaders and textures. Using packets of secondary rays for dynamic ambient occlusion and indirect diffuse lighting, would further stress the coherence needs of our method.

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

BOULOS S., EDWARDS D., LACEWELL J. D., KNISS J., KAUTZ J., SHIRLEY P., WALD I.: Packet-based Whitted and Distribution Ray Tracing. In *Proceedings of Graphics Interface 2007* (May 2007).

BENTHIN C.: *Realtime Ray Tracing on current CPU Architectures*. PhD thesis, Saarland University, 2006.

BENTHIN C., WALD I., SLUSALLEK P.: Interactive Ray Tracing of Free-Form Surfaces. In *Proceedings of Afrigraph* (November 2004), pp. 99–106.

CATMULL E., CLARK J.: Behavior of recursive division surfaces near extraordinary points. In *Computer Aided Design 10(6)* (1978), pp. 350–355.

COOK R. L., CARPENTER L., CATMULL E.: The REYES Image Rendering Architecture. *Computer Graphics (Proceedings of ACM SIGGRAPH 1987)* (July 1987), 95–102.

CHRISTENSEN P. H., FONG J., LAUR D. M., BATALI D.: Ray tracing for the movie 'Cars'. In *Proc. IEEE Symposium on Interactive Ray Tracing* (2006), pp. 1–6.

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. In *Computer Graphics Forum (Eurographics 2003 Conference Proceedings)* (September 2003), Blackwell Publishers, pp. 543–552.

COOK R., PORTER T., CARPENTER L.: Distributed Ray Tracing. *Computer Graphics (Proceeding of SIGGRAPH 84) 18*, 3 (1984), 137–144.

DJEU P., HUNT W., WANG R., ELHASSAN I., STOLL G., MARK W. R.: *Razor: An Architecture for Dynamic Multiresolution Ray Tracing*. Tech. Rep. UTCS TR-07-52, University of Texas at Austin Dept. of Comp. Sciences, Jan. 2007. Conditionally accepted to ACM Transactions on Graphics.

DEROSE T. D., KASS M., TRUONG T.: Subdivision surfaces in character animation. In *Proceedings of SIGGRAPH 98* (July 1998), Computer Graphics Proceedings, Annual Conference Series, pp. 85–94.

DOO D., SABIN M.: Behavior of recursive division surfaces near extraordinary points. In *Computer Aided Design 10(6)* (1978), pp. 356–360.

KOBBELT L., DAUBERT K., SEIDEL H.-P.: Ray Tracing of Subdivision Surfaces. *Proceedings of the 9th Eurographics Workshop on Rendering* (1998), 69–80.

LOOP C.: *Smooth subdivision surfaces based on triangles*. Master's thesis, University of Utah, 1987.

MARTIN W., COHEN E., FISH R., SHIRLEY P.: Practical Ray Tracing of Trimmed NURBS Surfaces. *Journal of Graphics Tools: JGT 5* (2000), 27–52.

MUELLER K., TECHMANN T., FELLNER D.: Adaptive Ray Tracing of Subdivision Surfaces. *Computer Graphics Forum (Proceedings of Eurographics '03)* (2003), 553–562.

OLIVER ABERT MARKUS GEIMER S. M.: Direct and Fast Ray Tracing of NURBS Surfaces. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing* (2006).

PHARR M., KOLB C., GERSHBEIN R., HANRAHAN P.: Rendering Complex Scenes with Memory-Coherent Ray Tracing. *Computer Graphics 31*, Annual Conference Series (Aug. 1997), 101–108.

PARKER S. G., MARTIN W., SLOAN P.-P. J., SHIRLEY P., SMITS B. E., HANSEN C. D.: Interactive ray tracing. In *Proceedings of Interactive 3D Graphics* (1999), pp. 119–126.

RESHETOV A.: Omnidirectional ray tracing traversal algorithm for kd-trees. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing* (2006), pp. 57–60.

RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-Level Ray Tracing Algorithm. *ACM Transaction on Graphics 24*, 3 (2005), 1176–1185. (Proceedings of ACM SIGGRAPH 2005).

STOLL G., MARK W. R., DJEU P., WANG R., ELHASSAN I.: *Razor: An Architecture for Dynamic Multiresolution Ray Tracing*. Tech. Rep. 06-21, University of Texas at Austin Dep. of Comp. Science, 2006.

VALGRIND TOOL SUITE: Cachegrind. http://valgrind.org/info/tools.html, 2007.

WALD I., BOULOS S., SHIRLEY P.: Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics 26*, 1 (2007), 1–18.

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

WALD I., IZE T., KENSLER A., KNOLL A., PARKER S. G.: Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics 25*, 3 (2006), 485–493. (Proceedings of ACM SIGGRAPH).

WALD I., SLUSALLEK P., BENTHIN C., WAGNER M.: Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum 20*, 3 (2001), 153–164. (Proceedings of Eurographics).

ZORIN D., SCHROEDER P., DEROSE T., KOBBELT L., LEVIN A., SWELDENS W.: Subdivision for modeling and animation. SIGGRAPH course notes, 2000.

ZORIN D., SCHRÖDER P., SWELDENS W.: Interpolating subdivision for meshes with arbitrary topology. In *Computer Graphics* (1996), vol. 30, pp. 189–192.