Packet-based Whitted and Distribution Ray Tracing

Solomon Boulos[†] Dave Edwards[†] J. Dylan Lacewell[†] Joe Kniss[‡] Jan Kautz[⋆] Peter Shirley[†] Ingo Wald[†]
[†]University of Utah [‡]University of New Mexico [⋆]University College London

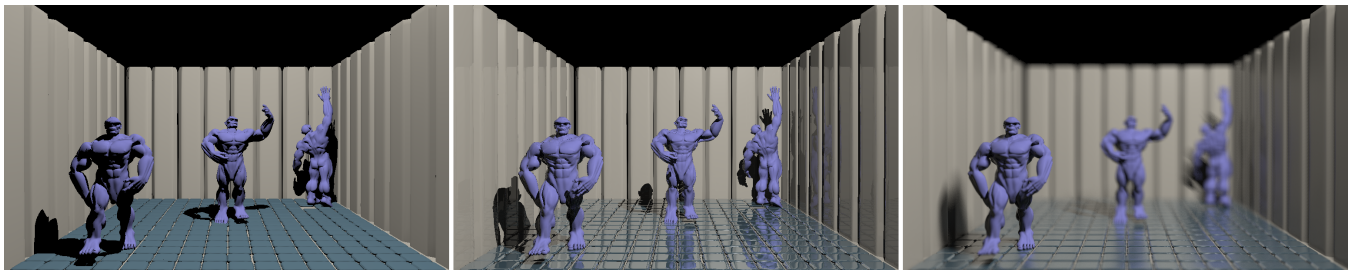

Figure 1: Left: ray casting with shadows (RCS). Middle: Whitted-style ray tracing (WRT). Right: distribution ray tracing (DRT) with 64 samples per pixel. This paper investigates interactive WRT on current hardware and the prospects for interactive DRT on future hardware.

Abstract

Much progress has been made toward interactive ray tracing, but most research has focused specifically on ray casting. A common approach is to use "packets" of rays to amortize cost across sets of rays. Whether "packets" can be used to speed up the cost of reflection and refraction rays is unclear. The issue is complicated since such rays do not share common origins and often have less directional coherence than viewing and shadow rays. Since the primary advantage of ray tracing over rasterization is the computation of global effects, such as accurate reflection and refraction, this lack of knowledge should be corrected. We are also interested in exploring whether distribution ray tracing, due to its stochastic properties, further erodes the effectiveness of techniques used to accelerate ray casting. This paper addresses the question of whether packet-based ray tracing algorithms can be effectively used for more than visibility computation. We show that by choosing an appropriate data structure and a suitable packet assembly algorithm we can extend the idea of "packets" from ray casting to Whitted-style and distribution ray tracing, while maintaining efficiency.

1 Introduction

Some predict that ray tracing will soon replace rasterization as the underlying algorithm for desktop graphics. Others believe this will not happen in our lifetime [1]. Ray tracing has a number of advantages over rasterization, including automatic visibility culling, time complexity sub-linear in the number of objects, and ability to take advantage of multi-core architectures. But the key advantage of ray tracing over rasterization is that it offers higher-quality images when "secondary" rays (e.g., for reflection and refraction) are used. The main drawback of ray tracing is that it is currently slower than hardware-based rasterization for most scenes. In this work, we investigate the practicality of interactive ray tracing with secondary rays, such as reflection and refraction. We also explore the future practicality of interactive distribution ray tracing.

One problem with discussing interactive ray tracing is that *ray tracing* is an overloaded term. In this paper, we use the term *ray casting* to refer to the use of ray tracing for visibility computations only (RCS, Figure 1, left). By adding reflection and refraction to ray casting, we can implement Whitted's [34] well-known algorithm; hence we refer to such a method as *Whitted-style ray tracing* (WRT, Figure 1, middle). The next step beyond WRT is *distribution ray tracing* (DRT, Figure 1, right) [6]. A DRT renderer uses multiple primary rays per pixel to render non-singular effects such as depth of field, glossy reflection, motion blur, and soft shadows.

Recently, interactive ray tracing has been a popular topic for research. There are several current systems that can perform interactive ray casting; some of these implement simple shading by computing direct lighting from point sources. However, very few of these programs implement full WRT. One reason for this limitation is that most interactive ray casters trace packets of rays with shared ray origins, and reflection and refraction rays cannot be placed in such packets. The little evidence that exists about the performance of secondary ray packets is not encouraging [26], so WRT may not be able to take advantage of the techniques that have proven so effective for ray casting.

Despite the uncertain outlook for interactive WRT performance, we believe that rendering with WRT rather than simple ray casting is an important goal. Although ray casting is faster than WRT, it is inferior to current GPU graphics in both performance and image quality. To get out of this "worst of both worlds" situation, we need to develop methods that allow for interactive reflections and refractions. In this paper, we propose a new method for interactive WRT using a combination of generalized ray packets and a bounding volume hierarchy for improved efficiency, and we show that our system can run at interactive rates on current high-end computers. We also examine the overall impact of reflection and refraction rays on rendering performance, and extend these measurements to the types of secondary rays associated with full DRT. Our findings show that an interactive WRT-based renderer is currently viable on a high-end desktop system. We also believe that interactive DRT-based renderer, using general purpose hardware, will be possible within a decade. That would happen much sooner if special-purpose ray tracing hardware becomes available.

2 Background

Ray Casting. Different ray casting projects have targeted shared memory computers [3, 21, 22], clusters [7, 29], traditional GPUs [9, 24], and Cell processors [2]. FPGA and ASIC designs for ray casting hardware have been presented [8, 28, 35, 36]. One common technique for accelerating ray casting is grouping rays into packets to take advantage of coherence. Ray packets allow further efficiency through the use of SIMD instructions as well as packet-based culling [4, 20, 27, 31–33].

Dynamic Scenes. Most of this work is intimately tied to the type of spatial acceleration structure used. For grids, both incremental [25] and complete [12] rebuilding strategies have been proposed. Motion decomposition may be used to build good *k*d trees for dynamic

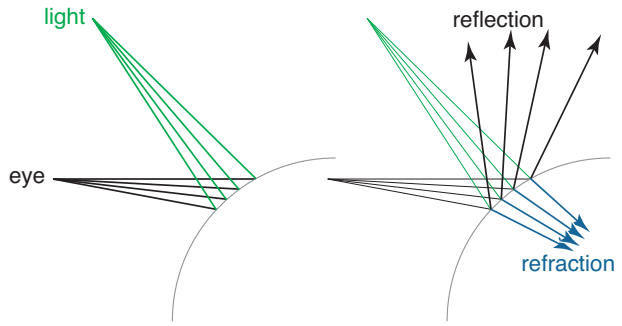

Figure 2: RCS with shadow rays from a point light source (left) and WRT adding reflection and refraction rays (right). In WRT, reflection and refraction rays form "general" packets that lack a common origin. Combining these two sets of rays into a single packet would have an extreme angular spread.

models if the full space of poses is known in advance [10]. Bounding volume hierarchies have been improved using incremental rebuilding schemes borrowed from collision detection [19, 20, 31].

Shadow Rays. Simple shading is often added to ray casting by computing direct lighting from point lights. This is a simple extension of ray casting: determining direct lighting from a point source is analogous to computing visibility from a pinhole camera. Shadow rays can therefore be traced from the light source using exactly the same techniques as primary rays from the camera [31]. A similar argument applies to computing soft shadows at a single point, for example, by sending 16 shadow rays per primary ray [22].

Whitted-style Ray Tracing. As with ray casters, most interactive Whitted-style ray tracers use ray packets to improve performance of visibility rays. When reflection and refraction rays are added to a packet-based ray tracer, it is not clear how packets of secondary rays should be constructed, nor whether such packets will even provide performance benefits. While some interactive ray tracers support reflection [20, 22, 26, 29] and a few have added refraction [3, 22, 30], there is little detail in the literature about the impact of adding such secondary rays on rendering performance. Most of these systems also abandon the use of packets for such secondary rays, presumably because secondary rays lack a shared origin. There are three exceptions to this in the literature. The special purpose hardware from the University of Saarland [28, 35, 36] uses packets of four rays for both primary and secondary rays, but statistics are not provided for the the performance of secondary rays. Bigler et al. [3] use packets for all secondary rays, but they provide no results on the performance of such packets. Reshetov [26] uses packets for reflection rays, and concludes that when using *k*d trees, packets may not help performance.

Distribution Ray Tracing. While DRT is currently a batch algorithm, reducing the number of samples per pixel has been a focus since its invention [6]. Low sampling rates can be achieved using interleaved sampling, which tries to replace low-frequency artifacts with dithering-like structured error in screen space [15, 17]. Although interactive DRT is a worthy goal, it is inherently slower than WRT for two reasons. First, to render non-singular effects, DRT requires more samples per pixel than WRT. Second, the rays generated in a DRT renderer are less coherent (i.e., they have a greater range in both ray origin and direction) than WRT rays, again due to non-singular effects such as depth of field and glossy reflection. This reduced ray coherence could imply DRT rays are intrinsically more expensive than coherent WRT rays.

Summary. Most research indicates that packets are very useful for interactive ray casting, but there is little quantitative evidence to support the usefulness of secondary ray packets for reflection and refraction rays. The overhead cost of replacing ray casting with WRT is also unclear. Finally, even if packets can be useful for WRT, it is not known whether the same will be true for the less coherent rays in DRT. These unknowns are the main topic of this paper.

3 Interactive Whitted-style Ray Tracing

In this section, we describe the problems that arise when extending a packet-based RCS program to allow WRT. The first hurdle is that while viewing rays and shadow rays from a point light source have a common origin, reflection and refraction rays do not (Figure 2). The second problem arises due to the lack of an obvious way in which rays can be grouped under WRT. We now examine possible ways to address these issues. First, we explore the advantages of packets, then, we look into suitable options for assembly of secondary rays into packets, and finally, we describe an acceleration structure that is appropriate for this type of computation.

3.1 The Problem of General Packets

A natural way to create packets of rays in RCS is to group them according to their shared common origin. This is possible due to the pinhole camera model (Figure 2, left). However, reflection and refraction rays necessary for WRT do not share such common origin (Figure 2, right).

When using 4-wide SIMD, a program may be able to perform a single SIMD operation instead of 4 scalar operations per set of 4 rays. This is very useful if all 4 rays would perform the same computation, such as being tested against the scene bounding box. If the 4 rays are not identical, however, they may not follow the same path through the acceleration structure; this leads to a SIMD utilization between 25% and 100% at each step depending on how many of the rays would have performed the operation if traced independently.

Programs using ray packets can also benefit from algorithmic amortization. For example, the MLRT system [27] attempts to find a starting point for sets of rays that is deeper down the kd-tree than the root node. This allows for amortization across packets with more than 4 rays. Similarly, in a bounding volume hierarchy, all rays in a packet can descend on a hit because false positives do not generate incorrect final results. This allows for the use of larger ray packets. In the case of the simple root node example, the entire packet could be processed for the cost of one ray if all the rays in the packet would have hit the first node anyway. Alternatively, if the first ray in a packet misses a node, the algorithm then resorts to a conservative interval arithmetic culling test that may quickly determine that all rays miss the bounding volume.

Both SIMD and algorithmic amortization across packets of rays work well when rays are "similar enough". However, dissimilar rays can reduce, eliminate, or even reverse speed improvements. The reversal is possible because some algorithmic overhead is added to allow amortization. To maintain increased performance over single rays while using packets we must decide how to group rays into packets so they are "similar". If we are able to group rays such that our amortization schemes perform well, we can expect an increase in system performance. Of course if the grouping method has significant overhead, any gain in tracing amortization may be lost. We examine this issue in the next section.

3.2 Assembling General Packets for Secondary Rays

In a packet of primary rays, some or all of the viewing rays will hit surfaces that may or may not share object ID, material properties, geometric proximity, or surface orientation. Any of these properties may be used in deciding how to create packets of secondary rays. Figure 3 illustrates the complexity of assembling secondary rays into packets. For the 16 primary rays shown, 6 shadow rays are generated, 12 specular reflection rays are generated, and 6 specular refraction rays are generated. Among the many options for generating secondary ray packets, we have singled out the following as examples of particular families of approaches for how one incident ray packet generates secondary ray packets:

NO PACKETS. Each of the secondary rays is sent separately.

RUNS. Secondary rays are traced in the same packet if they have some property in common (e.g., intersected material type), and their corresponding primary rays are numerically adjacent to each other.

GROUPS. All secondary rays with some common property (e.g., intersected material type) are grouped in a packet.

RAY TYPES. Three packets are generated: one containing all shadow rays, one containing all reflection rays, and the third containing all refraction rays.

BLIND. One packet is generated, containing all secondary rays.

Figure 3: Sixteen packeted rays hit various objects and materials. Rays 1, 2, 3, and 4 hit diffuse surfaces and only generate shadow rays. Rays 5 and 6 hit the floor and generate both specular reflection and shadow rays. Rays 7, 8, 11, and 12 hit the metal teapot and generate only specular reflection rays. All other rays hit glass objects and generate both specular reflection and refraction rays.

3.2.1 Packet Assembly Example

For the example in Figure 3, the following secondary packets are sent for the runs, groups and ray types method. We assume that intersected material type is the common property used to group rays in the runs and groups methods.

RUNS. Two packets of shadow rays are generated, containing rays $(1,2,3,4)$ and $(5,6)$, respectively. Five packets of reflection rays are traced: $(5,6)$, $(7,8)$, $(9,10)$, $(11,12)$, and $(13,14,15,16)$. Finally, two refraction packets are generated: $(9,10)$ and $(13,14,15,16)$.

GROUPS. The packets are similar to the Runs method, except rays in the same packet need not be adjacent. Once again, two shadow ray packets are traced: $(1,2,3,4)$ and $(5,6)$. However, only three reflection packets are generated: $(5,6)$, $(7,8,11,12)$, and $(9,10,13,14,15,16)$, along with a single refraction packet: $(9,10,13,14,15,16)$.

RAY TYPES. Three packets are traced, one containing all shadow rays $(1,2,\ldots,6)$, one containing all reflection rays: $(5,6,\ldots,16)$, and one containing all refraction rays: $(9,10,13,14,15,16)$.

3.2.2 Analysis of Assembly Methods

RUNS requires only one packet to be worked with at a time because a packet can be scheduled as soon as its run is interrupted. The RAY TYPES method requires no more than three packets to be worked with at a time, so its implementation is also straightforward.

GROUPS is problematic for several reasons. First, the number of groups is bounded only by the number of outgoing rays. For example, if we group outgoing rays into packets such that rays within a small angle θ of each other are assembled into a single packet, it is possible to choose θ small enough so that all outgoing rays end up in different groups. However, it is possible to bound the total number of groups (and hence outgoing packets) allowed. For example, directional binning could generate 8 packets corresponding to the 8 possible direction octants. In fact, the ray type method described above is a particular form of grouping that generates at most three outgoing packets: one for shadow rays, one for reflection and one for refraction. We prefer the ray type grouping method, as it has low overhead and is less sensitive to ray ordering than the runs method. We also tested the octant-based directional grouping method, but it suffers from large space requirements; up to 16 groups are required, 8 for shadow rays and 8 for rays requiring recursive shading.

BLIND is a poor choice because grouping secondary rays of all types will yield packets with poor coherence. This case is made worse by transparent objects, since reflection and refraction rays are placed in the same packet, although the two types of rays will tend to go in very different directions. Furthermore, it is often more efficient to trace shadow rays separately from recursive shading rays; shadow rays only require occlusion tests, which may be more efficient than the full intersection tests needed for other ray types. Unfortunately, the blind method does not allow even this simple separation.

Due to the disadvantages of general grouping and blind assembly, only ray types and runs are likely viable options. An empirical comparison of these two methods can be found Section 5.

3.3 Choice of Acceleration Structure

When implementing an interactive ray tracing system to support RCS there are three popular choices for the acceleration structure: kd-trees, grids, and BVHs. Each of them has proven to be a viable option for RCS. However, the general packed of WRT have a greater angular spread than those of RCS due to reflection and refraction rays. This greater spread influences the different acceleration structures differently.

The kd-tree can be implemented to handle general packets of rays [26]. However, the performance of that implementation suggests that when the reflection rays diverge there is no amortization benefit over single rays. Besides this point, kd-trees currently do not easily support deformable models without predictable deformation. This suggests that kd-trees as currently implemented are not an attractive option for full WRT with packets.

The coherent grid traversal has proven to work well for RCS [32].

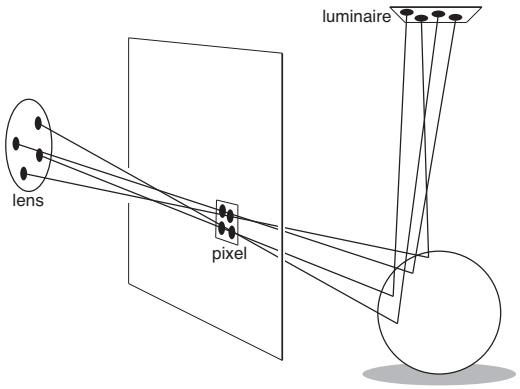

Figure 4: Distribution Ray Tracing.

However, the performance of the grid traversal scheme is intricately tied to the expanse of the frustum surrounding all the rays. The grid method tests all rays in the frustum against all triangles touched by the frustum yielding a high number of primitive intersections. In the original paper [32], this is mitigated by use of mailboxing and primitive frustum culling. While mailboxing helps avoid retesting a particular ray with a particular primitive, it does not remove the penalty of testing rays against primitives they would not have tested in a single ray implementation. However, it is clear that if the frustum is wide, frustum culling will only help for triangles outside the frustum which will only be in a small number of cells. For this reason, we believe that the wider frusta of WRT will quickly destroy the performance of the grid traversal (this result is hinted at by the test of wider 32 pixel by 32 pixel packets in the original paper).

The BVH is not very sensitive to "false negatives" in ray-box intersections [31]. In addition to making that structure well-suited to deformable primitives, it is also robust against some "spread" in ray packets. In addition, the BVH is the fastest structure for many models on current hardware. For these reasons we adopt the BVH for our tests. We use the basic intersection methods from Wald et al. [31] with the interval arithmetic culling described in detail in Boulos et al. [4].

4 Distribution Ray Tracing

Distribution ray tracing differs from single-sample WRT in several important ways. The major difference between WRT and DRT is the ability of DRT to handle non-singular effects such as depth of field, motion blur, soft shadows, and glossy reflections (see Figure 4). In DRT, multiple primary rays are traced through each pixel, and each primary ray originates from a different position on the camera lens, at a different time. Rays that intersect a surface send shadow rays to different positions on area light sources. Rays that hit glossy surfaces send reflection rays that are perturbed from the ideal reflection direction.

The main concern for interactive DRT is that the rays will not exhibit enough coherence to make ray packets worthwhile. In Section 5, we quantitatively examine the cost of ray packets in DRT. In the remainder of this section, we describe the main differences between our WRT and DRT implementations.

In single-sample WRT implementations, rays that hit a dielectric surface must branch into a reflection and a refraction ray. One benefit of the multisampling in DRT is that ray branching is not as vital as it is in single-sample renderers. For example, if a packet of 64 rays hits a surface that it 25% reflective and 75% refractive, instead of tracing N reflection and N refraction rays, we would trace only N rays total, 25% of which are reflection rays, and 75% of which are refraction rays. This cuts down on the branching factor in the ray tree and uses multisampling to average the combined effects of reflection and refraction.

The only aspect of primitive intersection not already handled by our WRT renderer is motion blur. Each frame in a WRT renderer occurs at one exact point in time, and all triangles have a fixed, well-defined position, even in an animated scene. In DRT, however, each frame corresponds to a continuous interval of time, so a moving primitive actually has an entire range of possible locations. As each ray has a fixed time stamp, different rays may potentially see the same triangle at different positions. Therefore, we cannot use the projective triangle test proposed by Wald et al. [33], since this depends on precomputing data for static triangles. Instead, we use a barycentric triangle test similar to one optimized for general packets of rays by Kensler and Shirley [16].

To make DRT interactive, it is imperative to use few samples per pixel. For the amount of computational power that will be available in the foreseeable future, this implies that the number of samples will be smaller than that needed for convergence, and visible error will be present. A sampling method such as Keller and Heidrich's [15] interleaved sampling can be used to decrease the perceptible error for a given sampling rate. Although this makes the sampling code somewhat more complicated than a simple jittering-based DRT implementation, it does not negatively impact performance. Another benefit of using a static sample set such as the one presented in Keller and Heidrich is that it removes the temporal scintillation artifacts that arise when using different random samples in each frame.

We have tried a number of Monte Carlo and quasi-Monte Carlo sampling schemes. Although different sampling methods lead to varying amounts of visible noise (for a given number of samples per pixel), rendering performance is not highly dependent on the sample set used. The one exception is that the mapping from random samples to scene space must be done carefully. For example, when sampling a Phong lobe for glossy reflection, certain mappings transform the origin in sample space to a reflected ray that is perfectly tangent to the object's surface. These tangent rays generate many false positives when performing the BVH traversal, and adversely affect performance.

5 Results

In this section, we compare the performance of our system for both WRT and DRT. We have found that WRT is interactive now on high-end desktop systems and benefits from packets of both primary and secondary rays. For WRT, we also demonstrate that not only do we get a benefit from SIMD packet tracing, but also still achieve an algorithmic improvement beyond SIMD.

We also demonstrate that while DRT is much slower than WRT, this is mainly due to the higher number of rays traced. In terms of raw numbers of rays traced per second, we still achieve algorithmic amortization above that afforded by SIMD. This is enabled by the appropriate choice of data structure and packet assembly algorithm.

Finally, we show that as the number of samples per pixel increases for DRT, the rendering cost is not always exactly equal to a direct scaling. For example, going from 4 samples per pixel to 16 samples per pixel can result in less than a 4x increase in rendering time.

Figure 5: Our three test scenes. Left: pool hall (305,314 triangles). Middle: conference (282,664 triangles). Right: rtrt (83,844 triangles).

5.1 Methodology

To determine the performance of our system, we report both architecture independent metrics as well as absolute render time metrics. The architecture-independent metrics that we report here include the number of box and primitive intersection tests per ray. As the cost of tracing rays is highly dependent on these two simple statistics, we feel it is a good metric for comparing algorithmic differences without requiring implementation details such as use of SIMD, processor clock frequency, cache size, etc. To ensure that the architecture independent improvements are not associated with large hidden costs, we also report the total number of rays cast per second. We also report all numbers for a single core of a 2.4 GHz Opteron 880, despite having a 16 core system. While our system scales linearly with the number of cores, we believe that single core timings allow readers to scale the results to the size of their system.

Our system is implemented in C++ with SIMD extensions, and for ray casting with shadows our performance is similarly to the system demonstrated by Wald et al. [31]. The most important differences to that system included adding support for general packets, reflection, refraction, and interleaved sampling. As noted by Reshetov et al. [27], just adding support for normalized viewing rays, local shading, and display significantly slows down ray casting. We have noted the same phenomenon in our code, and it is similar to the factor of two noted by Reshetov et al. All of our shading models include computation of Fresnel reflectance and other more advanced shading techniques. We believe this is a more accurate depiction of the desired shading models used in high quality rendering.

We ran our system on three scenes (see Figure 5) using camera paths for each scene (the path for the conference scene was originally used by Reshetov [26]). The maximum ray depth allowed was set to 50, but ray tree attenuation keeps the trees much shallower.

5.2 Whitted Ray Tracing

As mentioned from the outset, one concern with extending packet tracing algorithms to WRT is that there might not be enough coherence available to gain anything beyond a single ray implementation. We compared for each of our three scenes, the performance of our system for varying packet sizes and found that our algorithm is able to extract enough coherence to not only gain a benefit from SIMD, but also from the first hit and interval arithmetic test used by our BVH. Table 1 demonstrates how our system performance varies with packet size compared to single ray across our test scenes.

The reason our system increases in performance over single ray is fairly simple. Despite being less coherent than primary visibility rays, our system still amortizes a significant amount of box and primitive tests. The first row in each table compares 2x2 packets of primary rays, with single ray tracing for reflections and shadows. By comparison, the ray type grouping produces around a 2-2.5x speedup even for such small packets.

For the conference scene test, we disabled ray tree pruning for a more direct comparison to Reshetov [26]. Ray type grouping with 2x2 packets is then identical to SIMD packet tracing. If the only benefit our system could expose was due to SIMD, we would not see further increases in performance for increases in packet size. It should be noted that despite a programmer visible SIMD width of 4, that realistically implementation achieve between 1.8-2.5x instead of a theoretical 4x [33].

Increasing packet size allows the BVH to take advantage of algorithmic amortization beyond the natural SIMD width of the system. However, increasing the packet size may also greatly increase the number of primitive intersections performed as more rays "come along for the ride". In Figure 6 we examine the number of primitive and box tests per ray for increasing packet size. The other scenes demonstrate fairly similar behavior.

	Single Ray	Ray Type	Speedup
"Conference" (no ray tree attenuation, bounce depth 5)			
2x2	.37M	.76M	2.02x
4x4	.43M	1.14M	2.65x
8x8	.44M	1.25M	2.85x
16x16	.40M	1.00M	2.53x
"rtrt" (w/ ray tree attenuation)			
2x2	1.09M	1.98M	1.82x
4x4	1.20M	2.85M	2.37x
8x8	1.22M	3.30M	2.69x
16x16	1.14M	3.02M	2.65x
"Pool Hall" (w/ ray tree attenuation)			
2x2	.64M	1.21M	1.89x
4x4	.71M	1.85M	2.60x
8x8	.73M	2.18M	2.97x
16x16	.71M	2.10M	2.96x

Table 1: Millions of rays per second for different scenes, packet sizes, and ray tree attenuation vs no ray tree attenuation. In "conference", all surfaces are reflective and we use a maximum bounce depth of 5 without ray tree attenuation, while "rtrt" and "Pool Hall" have ray tree attenuation turned on. As can be seen in the 2x2 case, we get a speedup of roughly 1.8x to 2.0x through SIMD alone. On top of that, for larger packets, we get an additional speedup of $\sim$1.5x through algorithmic amortization, for a total of 2.5x–3x.

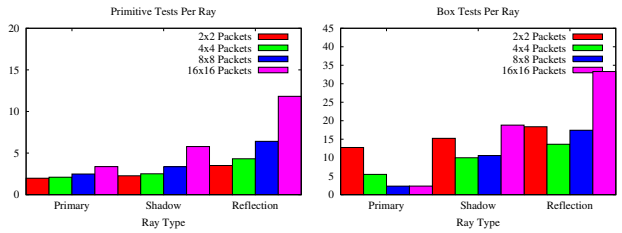

Figure 6: Number of primitive (left) and box (right) tests per ray for WRT for the conference scene as we vary the packet size. The data clearly demonstrates that using 16x16 packets causes an explosion both in the number of primitive and box tests per ray. The 8x8 packets achieve a sweet spot for most data, however, the 4x4 packets produce similar numbers.

5.3 Distribution Ray Tracing

With DRT at 64 samples per pixel, we are factors of hundreds or thousands from interactive performance on commodity hardware. Because DRT uses 64 samples per pixel instead of 1 as in RCS and WRT, we find it more meaningful to compare the total number of rays traced per second. For DRT, the rays per second achieved is about one half that for WRT on the conference scene, and only about 30% worse for the rtrt scene (see Table 2). The reduced performance for the conference scene is partly an artifact of the conference scene material parameters: all surfaces are reflective with a fairly glossy exponent (in essence creating a DRT version of the test by Reshetov [26]).

	RCS rps	WRT rps	DRT rps
conference	3.25M	1.79M	0.88M
rtrt	3.30M	2.00M	1.53M
poolhall	2.83M	.83M	1.23M

Table 2: The number of rays traced per second (rps) in millions for each of our scenes. This data is for a single frame and not the camera paths used from the WRT results.

While the total number of rays cast per second is usually lower in DRT than WRT, if we cast few enough rays we can provide interactive performance. The purpose of DRT, however, is to render fuzzy effects that WRT cannot produce. In practice, rendering these effects requires somewhere between 16 and 64 samples per pixel. As sample density increases, however, we can use larger ray packets and either maintain or increase the number of rays cast per second (see Table 3).

	Single Ray	Ray Type	Speedup
2x2	.42M	.73M	1.86x
4x4	.44M	.88M	2.00x
8x8	.29M	.88M	3.03x

Table 3: Millions of rays traced per second for the conference scene under distribution ray tracing at 4, 16, and 64 samples per pixel. Each setting uses a packet of rays equivalent to 1 pixel.

5.3.1 Performance as DRT features vary

In our tests, most DRT effects display fairly small performance differences across different values and the different scenes. For example, in changing the diameter of the lens aperture from 0 (a pinhole camera) to twice a reasonable size we only see small differences in the number of box and primitive tests at each step (see Figure 7). Changing the light source size only affects the shadow rays as these rays do not cast recursive rays. The shutter time behaves similar to other variables for small values, but performance is non-linear with respect to equal steps in shutter time. When the shutter time is short, primitives in the scene expand the bounding boxes less and to the packet of rays "look like smaller primitives". As this shutter time becomes longer, this effectively creates much larger primitives for the rays to intersect.

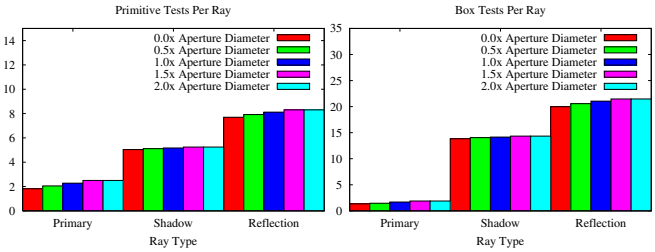

Figure 7: Effect of aperture on primitive tests (left) and box tests (right) per ray for the conference scene. As aperture diameter increases the effect on primitive intersections is fairly small. Similar behavior is seen for other DRT effects such as glossy exponent and light source size.

5.4 Packet Assembly

As shown previously, compared to tracing single rays or even SIMD packets of rays, the ray type assembly algorithm offers increased performance. Our previous results all use the ray type assembly because it is usually 10-20% faster for a given scene over the full animation path than the runs assembly. While this seems like a small improvement, it is important to understand where this improvement comes from.

The difference in performance between ray type and runs assembly can be seen from looking at the behavior of packets of rays as the bounce depth increases (see Figure 8). Both methods perform fairly similarly at first and the difference in overall performance is only around 10%. At higher bounce depths, however, the runs assembly usually demonstrates significantly more primitive and box tests. While the runs method produces slightly less primitive intersections overall, the increased number of box tests counter balances this. This implies that while the ray type method may sometimes produce large bad packets (e.g., when half the rays hit a nearby object and half hit a distant wall), the losses from the conservative decisions made by the runs method are a more serious problem in our tests. As the performance gap between CPUs and memory increases, reducing box tests will reduce memory accesses and should widen the gap between ray type assembly and runs assembly [26]. Similarly, if the number of rays traced at deeper bounces becomes more important (as for path tracing or caustics from long specular chains) the ray type assembly should pull further ahead.

6 Conclusions and Discussion

We have demonstrated what we believe is the first interactive WRT system to support deformable scenes. We have shown that ray packets and reflection/refraction rays are not necessarily incompatible. We have also shown that DRT is not severely more expensive *per ray* than WRT, and that most of the cost difference is due to necessary multisampling.

The following are a number of important questions we have not definitively answered, along with our best current answers. We believe all of these topics deserve further study.

What applications benefit from ray tracing? The sub-linear time complexity of ray-scene intersections is a primary advantage of ray

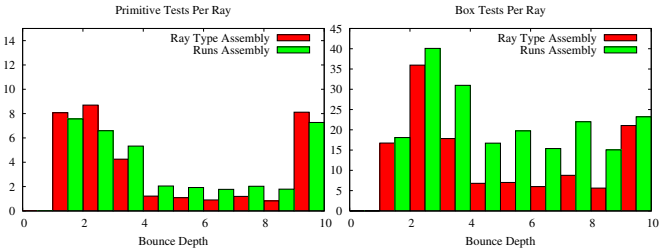

Figure 8: Ray type assembly and runs assembly for primitive tests (left) and box tests (right) with increasing bounce depth. The final column is an overall average taking into account the number of rays cast at each depth. It is not an equal average of all columns due to ray tree pruning. The ray type assembly method produces substantially fewer box and primitive tests as bounce depth increases.

tracing, which allows us to interactively render densely tessellated surfaces with complex lighting and materials. The process of character posing often involves manipulating a bone rig and previewing the influence on a low-resolution mesh in a limited lighting environment. The reason for this is due in large part to polygon rasterization limits and the demands of complex lighting. Interactive DRT offers the potential for animation setups in a lighting environment that is closer to the final frame quality. DRT also holds promise for interactive games with substantially more detailed models and more general lighting. This approach avoids special case approximations such as environment maps and low quality shadow maps by directly and efficiently simulating reflection and visibility. The character model seen in Figure 1 can be posed in our system at interactive rates (at lower sample rates, 1-16 samples per pixel), allowing an animator to work in a more representative lighting environment.

Is DRT enough? Several extensions to WRT and DRT allow computation of global illumination effects; among these are path tracing [14], bidirectional path tracing [18], and photon mapping [13]. These are certainly useful for some applications, and if enough computational power becomes available, they are worth pursuing. However, we think DRT will be sufficient for many applications including most games, and simpler additions such as ambient occlusion will be almost as valuable as global illumination.

How many samples per pixel are needed? WRT benefits from multiple samples per pixel for antialiasing, and DRT requires multiple samples per pixel for acceptable image quality. The number of samples needed will depend on scene and display characteristics; in our experience, 16 to 64 samples per pixel have been sufficient for high quality results.

Should rasterization be used for visibility? This is the trend in the computer-generated film industry because of the high number of procedural objects (e.g., displacement-mapped subdivision surfaces) that are used [5]. However, we believe interactive applications will benefit more from enhanced lighting effects than from complex procedural geometry that requires on-the-fly computation. Since visibility computations are an inherent part of ray tracing, we believe that future interactive applications may simply use ray tracing alone, rather than computing visibility with rasterization.

Shouldn't GPUs be used for ray tracing? So far, GPU ray tracers are not as fast as CPU ray tracers. If reflection and refraction from curved surfaces are not needed, then the GPUs rasterization unit can be used to compute reflections, and the accumulation buffer [11] could be used to great effect. However, we think that such reflections and refractions are desirable.

What hardware will WRT and DRT run on? WRT performs reasonably well on commodity CPUs, and with teraflop processors, WRT should run very fluidly on most scenes. DRT, on the other hand, may require special purpose hardware, especially if high-resolution images are needed.

Are BVHs needed for secondary packets? It remains to be seen whether grids, kd-trees, or some other acceleration scheme can be used to make packets useful for secondary rays. The native advantages of the BVH make the use of packets naturally independent of SIMD, but some clever as yet unmade observation might make the kd-trees and grids similarly natural.

Are ray packets a good idea? Ray packets trade software complexity for speed, and for secondary rays, the trade-off is not as clear as it is for ray casting. Although it is not often discussed in the graphics literature, most researchers are well aware that there is a high hidden cost for software complexity. More research into automatic ray scheduling in the spirit of Pharr et al. [23] could combine the best qualities of packet-based and single-ray code.

What is the new bottleneck? Shading time is now competing with total tracing time as the bottle neck (a profile of our code reveals that shading is now 1/3 of the total with traversal and primitive intersection the remaining 2/3), as long as some reasonable packet grouping is used. This implies that one of the most important tasks for future work is to be able to group shading operations and perform common sub-expressions in a parallel manner. Alternatively, robust methods that amortize shading costs (similar to irradiance caching) may provide the same sort of improvement for shading cost that we have seen in tracing costs.

What are the main limitations of this paper First, it is not clear how sensitive our results are to current CPU characteristics; as new cores become more "friendly" to ray tracing, the details of performance tradeoffs could change. Second, our results are on models composed of triangles, and applications that use subdivision or spline surfaces directly may have different characteristics. Finally, even for applications that use models composed of triangles, the details of what triangles and shaders are used may make results different in significant ways.

ACKNOWLEDGMENTS

Thanks to Margarita Bratkova for her helpful suggestions on the writing of the paper and the reviewers for their detailed and helpful comments. This work was supported by the University of Utah, the State of Utah Center of Excellence program, the U.S. Department of Energy through the Center for the Simulation of Accidental Fires and Explosions under grant W-7405-ENG-48, and the National Science Foundation under grant 03-06151.

REFERENCES

[1] Kurt Akeley, Brad Grantham, David Kirk, Tim Purcell, Larry Seiler, and Philipp Slusallek.

[2] Carsten Benthin, Ingo Wald, Michael Scherbaum, and Heiko Friedrich. Ray Tracing on the CELL Processor. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 15–23, 2006.

[3] James Bigler, Abe Stephens, and Steven G. Parker. Design for parallel interactive ray tracing systems. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 187–195, 2006.

[4] Solomon Boulos, Ingo Wald, and Peter Shirley. Geometric and Arithmetic Culling Methods for Entire Ray Packets. Technical Report UUCS-06-010, 2006.

[5] Per H. Christensen, Julian Fong, David M. Laur, and Dana Batali. Ray tracing for the movie 'Cars'. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 1–6, 2006.

[6] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of SIGGRAPH*, pages 137–145, 1984.

[7] David E. DeMarle, Steven Parker, Mark Hartner, Christiaan Gribble, and Charles Hansen. Distributed interactive ray tracing for large volume visualization. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 87–94, 2003.

[8] J. Fender and J. Rose. Estimating performance of a ray-tracing ASIC design. In *Proceedings of IEEE International Conference on FPT*, pages 7–14, 2003.

[9] Tim Foley and Jeremy Sugerman. KD-tree acceleration structures for a GPU raytracer. In *Proceedings of HWWS*, pages 15–22, 2005.

[10] Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek. Ray Tracing Animated Scenes using Motion Decomposition. In *Proceedings of EUROGRAPHICS*, pages 517–525, 2006.

[11] Paul Haeberli and Kurt Akeley. The accumulation buffer: hardware support for high-quality rendering. In *Proceedings of SIGGRAPH*, pages 309–318, 1990.

[12] Thiago Ize, Ingo Wald, Chelsea Robertson, and Steven G. Parker. An Evaluation of Parallel Grid Construction for Ray Tracing Dynamic Scenes. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 47–55, 2006.

[13] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, pages 21–30, 1996.

[14] James T. Kajiya. The rendering equation. In *Proceedings of SIGGRAPH*, pages 143–150, 1986.

[15] Alexander Keller and Wolfgang Heidrich. Interleaved sampling. In *Proceedings of theEurographics Workshop on Rendering*, pages 269–276, 2001.

[16] Andrew Kensler and Peter Shirley. Optimizing ray-triangle intersection via automated search. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, 2006.

[17] Thomas Kollig and Alexander Keller. Efficient multidimensional sampling. In *Proceedings of EUROGRAPHICS*, pages 557–563, 2002.

[18] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Proceedings of Compugraphics '93*, pages 145–153, December 1993.

[19] Thomas Larsson and Tomas Akenine-Möller. Strategies for Bounding Volume Hierarchy Updates for Ray Tracing of Deformable Models. Technical Report MDH-MRTC-92/2003-1-SE, MRTC, February 2003.

[20] Christian Lauterbach, Sung-Eui Yoon, David Tuft, and Dinesh Manocha. RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 39–45, 2006.

[21] Michael J. Muuss. RT and REMRT - shared memory parallel and network distributed ray-tracing programs. In *Proceedings of the 4th Computer Graphics Workshop*, pages 86–98, 1987.

[22] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. In *Symposium on Interactive 3D Graphics*, pages 119–126, 1999.

[23] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *Proceedings of SIGGRAPH*, pages 101–108, 1997.

[24] Timothy Purcell, Ian Buck, William Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In *Proceedings of SIGGRAPH*, pages 703–712, 2002.

[25] Erik Reinhard, Brian Smits, and Chuck Hansen. Dynamic acceleration structures for interactive ray tracing. In *Proceedings of the Eurographics Workshop on Rendering*, pages 299–306, 2000.

[26] Alexander Reshetov. Omnidirectional ray tracing traversal algorithm for kd-trees. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 57–60, 2006.

[27] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. In *Proceedings of SIGGRAPH*, pages 1176–1185, 2005.

[28] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. SaarCOR – A Hardware Architecture for Ray Tracing. In *Proceedings of the SIGGRAPH/Eurographics Conference on Graphics Hardware*, pages 27–36, 2002.

[29] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, 2004.

[30] Ingo Wald, Carsten Benthin, and Philipp Slusallek. OpenRT – a flexible and scalable rendering engine for interactive 3D graphics. Technical Report TR-2002-01, Saarland University, 2002.

[31] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 26(1), 2006. Available as SCI Institute, University of Utah Tech.Rep. UUSCI-2006-023.

[32] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. In *Proceedings of SIGGRAPH*, pages 485–493, 2006.

[33] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive Rendering with Coherent Ray Tracing. In *Proceedings of EUROGRAPHICS*, pages 153–164, 2001.

[34] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[35] Sven Woop, Erik Brunvand, and Philipp Slusallek. Estimating performance of a ray-tracing ASIC design. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 7–14, 2006.

[36] Sven Woop, Jörg Schmittler, and Philipp Slusallek. RPU: A programmable ray processing unit for realtime ray tracing. In *Proceedings of SIGGRAPH*, pages 434–444, 2005.