Adaptive Ray Packet Reordering

Solomon Boulos[†] Ingo Wald[◇] Carsten Benthin[◇]

[†]Stanford University [◇] Intel Corporation

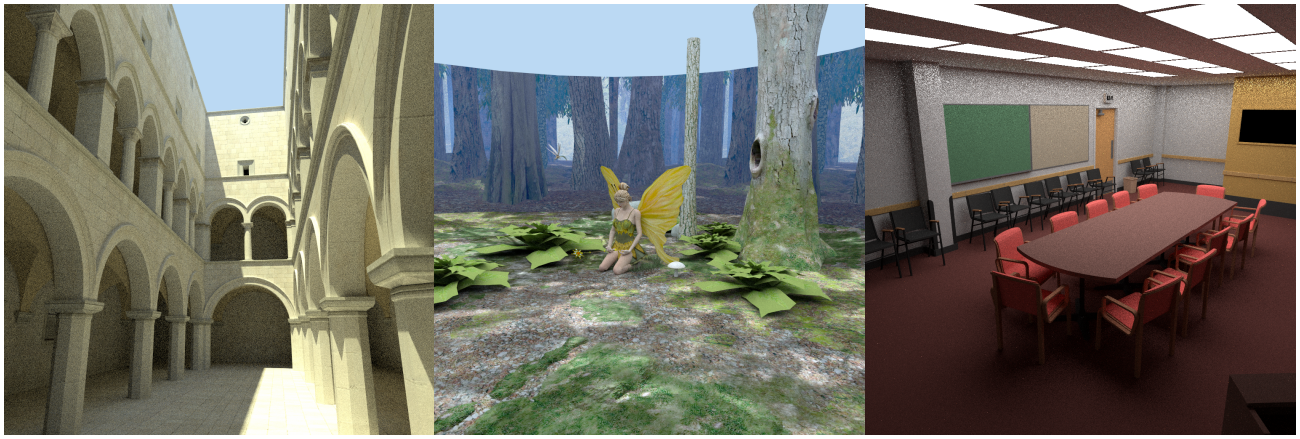

Figure 1: Our three test scenes rendered with two bounces of diffuse path tracing and one light sample per bounce (six rays per path) at 64 paths per pixel. Our reordering method maintains high SIMD utilization even for these incoherent ray distributions, achieving 1.2M rays per second for the conference scene (far right) ($1.5\times$ faster than BVH packet traversal and single ray traversal for 4-wide SIMD). As SIMD width increases, our SIMD speedup increases as well providing more than a $6\times$ reduction in box tests compared to a single ray implementation for 16-wide SIMD.

Abstract

Modern high-performance ray tracers use large ray packets and SIMD instruction sets to decrease both the computational and bandwidth cost compared to a single ray implementation. Current global illumination renderers, however, are still based around single ray implementations and interfaces. The presumption is that while packets have been shown to work well for highly coherent rays, in the presence of less coherent secondary ray distributions the gains of both packet and SIMD techniques dwindle rapidly. With low enough coherence, performance can be reduced to being as slow as reasonable single ray code – if not worse – so the benefit of packets for a global illumination system is assumed to be next to none. With SIMD width expanding in future architectures, leaving SIMD units underutilized means a massive loss in performance compared to the maximum performance achievable. In this paper, we present a method for recovering packet and SIMD coherence for incoherent secondary ray distributions through demand-driven reordering of rays into more coherent packets. We demonstrate that the reordering overhead is outweighed by the increased coherence within a prototypical implementation in the Manta realtime ray tracer among a wide variety of ray distributions, including diffuse path tracing.

1 Introduction

The real advantage of ray tracing is the ability to easily produce fuzzy effects, such as soft shadows, glossy reflections, motion blur, depth of field, and diffuse global illumination. To do so, however, requires either irregular, stochastic sampling [5] or massive amounts of regular, coherent sampling [24]. Recent research has focused immense effort at improving high coherence situations while ignoring the most common use for ray tracing: fuzzy effects including global illumination.

Current interactive ray tracers rely on SIMD instructions and packets to achieve high performance. Packets benefit these systems by reducing the amount of computation and bandwidth required to trace a set of rays. Computation is reduced either by amortizing computations over an entire packet for packets larger than the SIMD width (e.g. 64 rays miss a bounding box) or by using the SIMD instructions available on modern processors to perform SIMD-width computations for approximately the cost of a single computation. Similarly, bandwidth is reduced because geometry or acceleration structure data is only fetched once for a traversal step; fewer traversal steps yields less geometry bandwidth requirements. These techniques explicitly rely on high coherence, either SIMD or packet, to provide any benefit over traditional ray tracers using single rays. Worse still, there is a programming burden for using packets of rays that permeates the architecture of the rendering software.

The drive for interactivity has pushed speed at the cost of quality. This has led to interesting research into previous problem areas for ray tracing such as handling dynamic geometry, but has also led to the bizarre situation that many modern ray tracers produce low quality images (albeit quickly). There are two major contributing factors to this situation: programming difficulty and low performance. Writing complicated shaders using packets and SIMD instructions is painful but solvable through tools [14] The performance issue for tracing incoherent packets of rays has received almost no attention in the interactive ray tracing community (we highlight exceptions to this in the next section).

Worse yet, the focus on primary rays ignores a simple fact: primary rays are the minority of rays in high quality renderings. For renderings that send multiple shadow rays per light source or compute multiple bounces of reflections, the first level of rays does not account for a large percentage of the rendering time. While the "secondary" rays are not necessarily completely random, it is well understood that they do not behave in the same manner as coherent primary rays. Algorithms that are focused on primary rays are in some sense focused on the wrong target.

Current architectural trends also point to an expansion in SIMD

width over recent commodity x86 hardware. Even the initial SSE instruction set [8] implementations were 2-wide in hardware with 4-wide implemented through lock step pairs of instructions; only recently has the hardware truly supported 4-wide instructions. Intel has also announced that future x86 parts will use 8-wide SIMD instructions [9]. Recent GPUs are similar in their use of many parallel floating point units (usually referred to as stream processors in engineering specifications) with modern NVIDIA parts containing up to 128 such units [13] (in similar terms an 8 core Clovertown system has 32 such units).

The upcoming Larrabee [19] architecture will use 16-wide SIMD units further pushing the amount of data parallel performance available on a single general purpose chip. With such high peak performance capability, getting high utilization from incoherent secondary rays seems mandatory. While this also means current methods for tracing coherent rays will instantly become faster, this only means that the difference between what ray tracers are good at and what they should be good at will widen.

We see three possible options to regain benefits from packets for a global illumination system. The first is to throw out algorithms that shoot incoherent rays. While there are approaches that leverage this, we feel the massive amount of oversampling required will almost always be too great. The second is to abandon packets and attempt to utilize SIMD within a single ray. This approach would be prone to the same high bandwidth requirements as single ray code. The last approach is to reorder the rays into more coherent groups that can again be efficiently processed. This approach assumes that there is coherence present, but that it is "hiding". This reordering introduces overhead that may not be easily overcome.

In this paper, we focus on reordering rays into new coherent groups. Instead of grouping rays across small ray packets, we instead will allow the rendering system to generate packets with arbitrary length (up to some maximum size) and ray distributions. Our goal is to regain the benefits of packets enjoyed in previous approaches (both SIMD and otherwise) to reduce the total number of box tests (as we use a BVH), triangle tests, traversal steps and primitive tests given the input ray distribution. We feel this approach allows for maximum flexibility in shader authoring and would allow for ray-scene queries to be cleanly separated from ray generation and sampling. Packets that have hidden coherence will benefit from our approach, however, we do not hope to magically fix packets where no coherence exists. It is our assumption that even for a few bounces of diffuse path tracing, enough coherence exists for our approach to be useful. For more coherent ray distributions, we would expect even larger gains.

2 Related Work

2.1 Packets

Packets of rays were first introduced by Wald et al. [23] to utilize SSE vector instructions. Performance gains were relatively good for coherent primary rays and shadow rays from a point light, but shading and reflection rays were handled in single ray code. In this case, the implementation burden of writing shaders in SSE was a major contributing factor to use of single ray code.

Wald et al. [21] demonstrated a packet algorithm for Bounding Volume Hierarchies (BVHs) that resulted in high performance for dynamic scenes. Reshetov adapted a BVH style traversal to kd-trees to allow packets to remain together even when their directional signs disagreed [17]. Even with this modification, incoherent ray distributions led to extremely low SIMD utilization.

Reshetov [18] showed that a fast primitive culling test allowed for shallower kd-trees with larger numbers of primitives per leaf while maintaining similar rendering performance. The approach was not investigated for incoherent rays; however, the utility of the culling test clearly relies on ray coherence.

2.2 Ray Reordering

Pharr et al. [16] demonstrated the first deferred ray queueing implementation in the Toro system. The goal was to only load and tessellate geometry on demand to render scenes which would not fit into main memory. Their implementation did not focus on fine grained reordering and required the ability to queue a trace call. Subsequently, not every arbitrary shader can be written within this framework; however, any reasonable global illumination BRDF is expressible. The Kilaeua system [10] descheduled shading when a ray trace or photon map lookup was required to hide the latency of performing these computations across a cluster; this allowed them to remove the restriction of requiring shading authors to use "fire and forget" ray tracing from Toro.

Boulos et al. [2] grouped rays based on ray type and found reasonable gains over a single ray implementation. Their system used ray tree attenuation to filter out low importance rays, however, so the ray distributions were skewed towards early bounces and more coherent rays. While this is reasonable, we have chosen to not require ray tree attenuation as shader authors may not be able to compute an accurate estimate of importance (or may simply not do so).

Mansson et al. [12] investigated reordering methods for packets of rays following the spirit of the Toro system. By grouping rays into larger batches than their packet size and by shooting packet sized subsets of this batch in sequence. None of the proposed heuristics performed better than SIMD packet tracing in final rendering performance. Navratil et al. [15] instead reordered rays at queueing points within a kd-tree to reduce geometry traffic through simulated L2 caches. The focus was directed at geometry and ray bandwidth only, and no comparisons were made with respect to absolute performance or reducing computational costs such as ray-triangle tests.

2.3 Breadth First Ray Tracing

Breadth first ray tracing was first investigated by Hanrahan et al. [7]. It was also applied by Mahovsky et al. [11] to amortize the decoding of a compressed BVH format. Finally, Wald et al. [22] investigated breadth first ray tracing with reordering at every step. A practical implementation was not produced, but served as a theoretical precursor to this work. Breadth first ray tracing greatly reduces the number of node traversals required, however, it may introduce a high reordering cost.

3 Penalties for Tracing Incoherent Rays

While rendering complex scenes with global illumination algorithms, rays quickly diverge in both hit points and directions. In particular, current packet based systems rely on coherence so much that they may perform worse than a standard single ray approach. In this section, we outline a variety of problems with current methods and present our own solution that seeks to remedy the issue.

3.1 Naïve Packet Behavior

The BVH packet traversal from Wald et al. [21] performs a speculative "first hit" test and a conservative interval arithmetic based "all miss" test to give packets an algorithmic advantage over single ray traversal. The "all miss" test is conservative, but the worst that can happen is that the test fails and performs no useful work. The "first hit" test, on the other hand, is aggressively speculative, and can, in the worst case, drag a full packet to leaves that only a single ray wants to intersect (see Figure 2).

While this case is perhaps extreme, it demonstrates the inherent problem with the speculative nature of the BVH packet traversal method. This problem has surfaced even for primary rays. In the original paper, Wald et al. [21] attempted to address this by adding a simple "last active" test before primitive intersection. Unfortunately, this simple test can easily be fooled (it too is speculative in that it assumes all rays between the first active and the last active

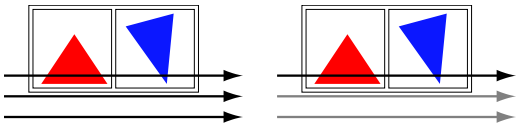

Figure 2: Standard BVH packet traversal (left) brings the other rays along as well performing extra work compared to single ray traversal (right). The first active ray (top) hits a box and descends into a subtree testing two more boxes per ray (last active tests avoid the triangles).

are active as well). This effect can be addressed by testing every ray for overlap with the leaf bounding box before intersecting primitives. While this final per-leaf testing avoids the (potential) $O(MN)$ ray-primitive intersections for N rays and M primitives, it instead performs $O(MN)$ ray-box tests.

As a further case of wasted work, we note that the interval arithmetic data over the packet of rays was never recomputed in the original approach. Essentially, this is another form of speculation: the intervals are assumed to remain approximately similar. This causes the "all miss" test to fail more often than it would otherwise.

3.2 SIMD Packet Traversal

The original SIMD packet tracing method [23] does not have any speculative behavior at all. Instead, whenever ray coherence is low the SIMD benefits are lost, but no additional box or triangle tests outside of the SIMD group occur either. SIMD packet tracing, however, also doesn't provide the opportunity for as large gains as the BVH packet traversal method [21] since the maximum reduction in box and triangle tests is tied to the SIMD width of the machine. Once only 1 ray is active, SIMD packet traversal may be as slow as a comparable single ray implementation.

3.3 Packet Filtering

While it appears as if the problem with BVH packet traversal *exclusively* stems from the speculative first-hit descent, this is not the case. Any packet technique that does not remove inactive rays from the packet will suffer a similar problem albeit at a different scale. We call removing/disabling inactive rays *filtering*.

If a packet is not filtered during traversal, any ray hitting a node will drag the entire packet down into the node's subtree. Consequently, an N-ray packet hitting M different leaves would have to consider all N rays in all M leaves. For SIMD packet tracing, there is not much penalty compared in having inactive rays within a SIMD group, however the gain beyond single ray is immediately diminished.

A simple solution to avoiding this problem is to determine when rays become inactive and to filter the packet by removing those rays. In this sense, the first-active and last-active tracking are a very primitive form of filtering packets. Filtering packets such that they still align on SIMD boundaries is usually done to avoid unnecessary single ray or masked SIMD code (which has a high penalty in SSE, see Section 4.5).

3.4 Exact Filtering

The only way to avoid dragging any inactive rays to leaves would be to filter them out at each traversal step. This requires distinguishing between three cases: all rays hit, all rays miss, or only some hit a node. In the case of a partial hit, it is necessary to determine which rays miss in order to remove them from the packet.

The simplest approach to achieve this result would be to take an input packet of rays and test all of them against the node to determine the "active mask". While this extracts the maximum SIMD utilization possible, it has not been shown that this technique would be feasible due to the amount of reordering necessary.

4 Adaptive Reordering of Ray Packets

We would like to combine the benefits of BVH packet traversal, SIMD packet tracing, and breadth first ray tracing into one new method. Specifically, we would like to retain the quick full packet tests of BVH packet traversal along with the SIMD benefits of breadth first SIMD tracing. However, we would also like to avoid the downsides of both methods: extra work from speculation and expensive reordering costs. At a high level, our combined algorithm is described in Algorithm 1.

Algorithm 1 Psuedo-code for our adaptive BVH traversal. As with the original packet method "all hit" and "all miss" are approximately the same cost as a single box test regardless of packet size.

```
if All Hit Enabled and All Rays Hit Box then
  Intersect Rays with Children
else if All Miss Enabled and All Rays Miss Box then
  Exit Early
else
  Determine number of active rays
  if Active Rays Below Threshold then
    if Only Single Ray or SIMD Group Remains then
      Use Optimized Single Ray/SIMD Path
    else
      Reorder rays
      Compute New Intervals
      Decide which packet tests to enable
      Intersect Resorted Rays with Children
      Update new hit results
    end if
  else
    {All hit has failed and intervals are not updated }
    Disable All Hit Test
    Intersect Rays with Children
  end if
end if
```

4.1 Full Packet Tests

As a first improvement on the BVH packet traversal, we replace the speculative "first hit" with an exact "all hit" technique. Just as for the "all miss" test we compute interval "ray parameters" (the interval arithmetic analogue to the ray parameter interval resulting from a box test) and determine whether every ray in the packet would instead hit the box. To do so, we take the interval ray parameters and determine interval "ray positions" (origin plus ray parameter multiplied by ray direction). If the positions at the end of the t_{min} interval are all inside the box, then every ray will make it inside the box by their own t_{min} value. If this is the case, then all rays hit the box.

The original "all miss" test also did not take into account the ray parameters when performing its conservative all miss test. As a simple improvement, we take this into account in our implementation as this improves our chance of determining that all rays miss a box that is farther away than the current maximum intersection value. We also note that alternatively, we could use a geometric frustum defined by the ray packet [3, 18].

Finally, depending on the coherence of the rays the interval arithmetic tests may not prove very useful. We can quantify the utility of the interval arithmetic tests by computing the benefit of these tests as N (the number of rays) if a test succeeds and -1 if a test fails (since no useful work occurred). In the original paper for coherent rays [21], interval arithmetic success rates of up to 73% are reported for the conference scene (73% of all attempts reported a miss). It seems clear that the success rate will be severely reduced for incoherent rays as the tests rely on ray coherence. Furthermore, the all

miss test may be used more often since the speculative first hit test has been replaced by a conservative all hit test (the speculative first hit test covered 50% of all tests). Currently, we disable the all hit test if we do not recompute the intervals (since it will clearly continue to fail) and disable both interval tests if the packet is smaller than some multiple of the SIMD width (since the maximum benefit is now low).

4.2 Partial Packet Overlap

If both of the full packet tests fail, we resort to determining the active rays intersecting the box. This requires a linear sweep over all the rays while we compute the "active mask". If we decide to reorder our packet, we copy any data necessary to perform intersection tests (i.e. ray origins, directions, hit distance, etc), continue traversal with the new ray packet, and then merge the results back in if new intersections are found.

This reordering can also be done in place as the active mask partitions the input rays into two disjoint groups. However, in our implementation we use a new ray packet and copy the results back instead (stack space is a minor concern on a CPU). In place sorting also removes the need to copy back the newly found intersection results but is more complicated to implement. We note that with BVHs there is no need to transfer the active mask up and down a stack as it is always computable and it would save no work to skip inactive rays within SIMD groups (any ray that misses a parent box will also miss a child box).

With an identical traversal path, this simple method never performs more ray-box tests or ray-triangle tests than a single ray implementation would (excluding the "all miss" and "all hit" tests). Due to whatever heuristic is used for traversal order for a packet, however, the final statistics might differ as compared to a single ray traversal. The success of the original packet method for coherent rendering suggests that it is unnecessary to perform such a resort at every node. It also seems possible that the cost of resorting or computing the active mask might dominate computation time and still produce a renderer slower than a single ray implementation.

4.3 Adaptive Reordering Heuristic

In a packet with many rays, reordering when even one ray becomes inactive seems wasteful. Instead of performing reordering at every step, we only reorder when the "packet utilization" drops below a threshold. We define *packet utilization* to be: A/N, where A is the number of active rays in the packet and N is the total number of rays in the new packet (i.e. the size following the first active and last active filtering). We can compute *SIMD utilization* of a SIMD group of rays for a target SIMD width fairly similarly: $A/(W)$, where A is now the number of active rays in the SIMD group and W is the SIMD width. The average SIMD utilization for the packet is simply the average of the SIMD utilization over all the SIMD groups.

In our implementation, we reorder whenever the packet utilization drops below 50%. Intuitively, this balances between the cost of reordering all the rays for each step and the overhead of low SIMD utilization. Empirically, our threshold of 50% works rather well (for more detailed comparisons see Section 5), but a more complicated heuristic might reorder less often for the same reordering gain.

4.4 Computing Tight Intervals

The original BVH packet traversal computed the "ray intervals" once per packet before traversal, however, our adaptive reordering method recomputes them whenever we reorder the packet. While a first approach would be to simply compute the intervals over the new subset of rays, we can do better.

Because the rays traversing a subtree can only intersect geometry between their ray intervals, we move the ray origins to their intersection points with the current bounding box and similarly clip the ray parameter to the exit point of the bounding box (see Figure 3). This improves tightness for incoherent rays in a similar manner that Reshetov's [18] vertex culling frustum is built over the intersection of the rays with the current bounding box.

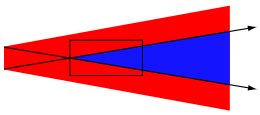

Figure 3: Two initial rays and their bounding region shown in red. After they intersect the bounding box, the new bounding region (shown in blue) is tighter. While this example is a best case (the rays converged), computing tighter bounds in this way is never worse than using the original bounds. This is true for both geometric and interval arithmetic based approaches.

4.5 Single Active Ray or SIMD Group

Another option is to switch to an optimized single ray (or single SIMD group) traversal once coherence is deemed to be too low. For many renderers, there is nearly no benefit to switching to anything lower than the SIMD width of the machine. It is possible, however, that a single ray implementation can be faster than a single SIMD group. For example, in SSE any store needs to be converted into a masked store which involves a load of the previous data, the computation of the mask, three bitwise masking instructions (or, and, and complemented and), and finally the store. In our implementation, we switch to single ray only when our packets reach a single active ray.

4.6 Reordering Shading

Given a packet of shading points from an arbitrary input ray packet the distribution of shaders could vary wildly. In particular, even with only a single shader on all geometry many of the rays could exit the scene and hit the background. With the exception of a hard coded constant background, this requires either shading in "runs" or grouping the resulting hit points into groups as well (see Boulos et al. [2] for the definition of runs versus groups).

Ignoring this step would vastly decrease the number of rays a shader would cast in a path tracing setting. For example, for the packet shown in Figure 4, the maximum shadow packet size is 1 despite a total of 9 necessary shadow rays. To remedy this issue, we reorder the shading requests by grouping them by material model using a single linear pass over the shading points and copy their results back into the original ray packet after they have been shaded.

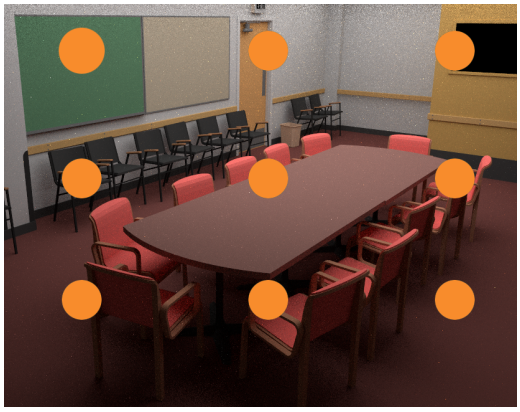

Figure 4: Nine input rays (shown in orange) that hit 9 different objects each with a different diffuse color. All the surfaces are lambertian, however, so they share an identical material model. Without grouping the hit points by material model they would not be shaded together.

As an implementation detail, we have chosen to treat all shaders of the same "material model" as one shader with textured inputs. This then moves the "runs" shading problem into texturing, which is already a scalar code path. This also provides another opportunity to reorder the shading points within a shading packet by the texture they want to fetch from. In systems we have worked with, the only utility of using packets for texture lookups comes from system cache performance and amortizing virtual function calls. In systems with dedicated texture units (such as GPUs) this might provide a larger benefit by making larger requests for a single texture.

5 Experiments and Results

To demonstrate the feasibility of our approach, we extended the Manta Interactive Ray Tracer [1] with our adaptive reordering traversal method. We ran all of our results on a 3.0 GHz Intel Clovertown system running Mac OS X Leopard (10.5). All results are specified in seconds per frame for a single core unless specified otherwise; we feel this allows our results to be more easily compared to other systems and can be scaled based on the number of cores in a given system. In all cases we also compute speedup over our single ray implementation as our goal is to inform the design of global illumination systems.

For our test scenes, we used the Sponza Atrium, Utah Fairy, and MGF Conference scenes. These scenes have varying complexity (76k, 174k, and 282k triangles respectively) of at least reasonable size to demonstrate the utility of our method. In particular, scenes with low geometric complexity (like erw6) do not rely much on traversal performance and have such large primitives as to generate unusually high coherence.

Each test scene was rendered at 1024×1024 resolution with a single area light source and we vary the maximum number of reflection bounces. Because the Sponza and Fairy scenes have open tops many rays will hit the background and not reach the maximum bounce depth. For completeness, two bounces means that we shoot one level of primary rays, two reflection rays, and three sets of shadow rays (see Figure 5). We do not densely sample the light source for direct lighting (e.g. shooting 16 samples per light source) so that we can be sure our method is robust to direct lighting methods used for larger number of light sources [20]. We use large area lights (for the Conference scene the light is almost the full ceiling) to further ensure that we represent this case accurately. Shooting more samples per light source could increase coherence for packet methods, but we would prefer our method to be robust to this likely ray distribution as well.

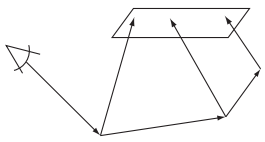

Figure 5: Our rendering setup with 2 bounces.

We chose three basic material types to provide a range of ray distributions: pure specular, glossy, and diffuse. The pure specular material evaluates the Blinn-Phong model and traces a secondary ray in the direction of perfect reflection. The glossy model is a coupled diffuse-specular model that only samples the specular component following Boulos et al. [2] with the same specular exponent as used in that paper. Finally, the diffuse case samples the hemisphere with respect to projected solid angle (cosine sampling).

Unlike the results in Boulos et al. [2], we do not use ray tree attenuation. Ray tree attenuation is a realistic assumption for both Whitted and Distribution Ray Tracing, but is commonly replaced with russian roulette in a path tracing setting. In any case, to be robust to arbitrary shaders (that may not compute importance aggressively), we focus our results on incoherent ray distributions that have not been filtered by ray tree attenuation.

For each scene and rendering style, we compare single ray, single SIMD group, standard BVH packet traversal with 64 rays per packet, and our new adaptive reordering packet traversal with 256 rays per packet. For all packet methods we group the shading points by material to maximize the average ray packet size (i.e. only single ray avoids this step). As Manta does not have an explicit single ray mode, we set the ray packet size as small as possible (4). Not doing so causes an unreasonable penalty in performance for single ray code due to cache behavior; we also explicitly copy the ray elements out of the packet for our single ray traversal to avoid this during BVH traversal.

5.1 Perfect Specular

In Table 1 we compare the four traversal methods on the three scenes at three difference bounce depths (2, 5, and 10). We ignore lower bounce depths for this case to focus on the incoherent rays that occur after many bounces. The results demonstrate that standard BVH packet traversal is still very applicable for short bounce depth specular chains while reordering is more useful with higher bounce depths.

Scene	Single	SIMD	Packet	Reordered
Sponza	7.7	3.4 (2.2×)	2.1 (3.7×)	2.5 (3.1×)
Fairy	5.5	2.9 (1.9×)	2.2 (2.6×)	2.2 (2.5×)
Conference	7.8	4.2 (1.9×)	2.8 (2.8×)	2.8 (2.8×)
Sponza	15.4	8.0 (1.9×)	6.6 (2.3×)	7.3 (2.1×)
Fairy	7.5	4.4 (1.7×)	3.8 (2.0×)	3.5 (2.1×)
Conference	17.3	10.0 (1.7×)	8.2 (2.1×)	7.9 (2.2×)
Sponza	27.6	18.5 (1.5×)	22.1 (1.2×)	21.0 (1.3×)
Fairy	9.5	6.0 (1.6×)	5.8 (1.6×)	4.8 (2.0×)
Conference	32.2	22.8 (1.4×)	24.4 (1.3×)	22.0 (1.5×)

Table 1: Performance in seconds per frame for the three different traversal methods with 2 (top), 5 (middle) and 10 (bottom) perfect specular reflections. For SIMD, packet, and reordered traversal, the relative speedup over single ray is also computed. Reordering is competitive with packet tracing for small numbers of bounces and pulls further away for higher numbers.

5.2 Glossy

Table 2 compares the four traversal methods for our glossy reflection model, continuing our progression from coherent to incoherent distributions. For the no bounces case, we are essentially evaluating the shading model and computing soft shadows. Just as before, the speculation present in the original packet method can pay off for low bounce depths, however, the situation quickly worsens for higher bounce depths.

5.3 Diffuse Path Tracing

Diffuse path tracing is the most incoherent ray distribution we tested. We trace only a single sample for direct lighting and a single hemispherical sample. As we can see in Table 3, our adaptive reordering technique gains ground over the packet and SIMD tracing techniques after only a single bounce. By two bounces, packet tracing is now as slow as a single ray implementation, while our adaptive reordering gives an improvement even for 4-wide SIMD.

5.4 Performance Independent Statistics

We now focus on diffuse path tracing in the conference scene to report our timing independent improvements of our algorithm. The conference scene, while being perhaps overly detailed, is one of the

Scene	Single	SIMD	Packet	Reordered
Sponza	2.4	1.0 (2.4×)	0.5 (4.8×)	0.7 (3.4×)
Fairy	2.1	1.1 (1.9×)	0.8 (2.6×)	0.8 (2.6×)
Conference	2.3	1.2 (1.9×)	0.9 (2.6×)	0.8 (2.9×)
Sponza	5.2	2.9 (1.8×)	2.2 (2.4×)	2.2 (2.4×)
Fairy	4.4	2.5 (1.8×)	2.2 (2.0×)	2.0 (2.2×)
Conference	5.0	2.9 (1.7×)	2.4 (2.1×)	2.0 (2.5×)
Sponza	8.2	5.3 (1.5×)	4.9 (1.7×)	4.3 (1.9×)
Fairy	5.8	3.8 (1.5×)	3.7 (1.6×)	3.0 (1.9×)
Conference	8.0	5.2 (1.5×)	4.9 (1.6×)	3.9 (2.1×)

Table 2: Performance in seconds per frame for the three different traversal methods with 0 (top), 1 (middle) and 2 (bottom) glossy bounces. As before, relative speedup over single ray is given in parentheses.

Scene	Single	SIMD	Packet	Reordered
Sponza	2.4	1.0 (2.4×)	0.5 (4.8×)	0.6 (4.0×)
Fairy	2.1	1.0 (2.1×)	0.8 (2.6×)	0.8 (2.6×)
Conference	2.3	1.2 (1.9×)	0.9 (2.6×)	0.7 (3.3×)
Sponza	6.0	3.9 (1.5×)	4.4 (1.4×)	3.4 (1.8×)
Fairy	4.1	2.7 (1.5×)	3.0 (1.4×)	2.2 (1.9×)
Conference	5.0	3.3 (1.5×)	3.7 (1.4×)	2.6 (1.9×)
Sponza	8.8	7.3 (1.2×)	9.8 (0.9×)	7.2 (1.2×)
Fairy	5.3	4.1 (1.3×)	5.0 (1.1×)	3.1 (1.7×)
Conference	8.0	6.0 (1.3×)	7.7 (1.0×)	5.3 (1.5×)

Table 3: Performance in seconds per frame for diffuse path tracing 0 (top), 1 (middle) and 2 (bottom) diffuse bounces. While many methods are competitive for the shadows only case (0 bounces) BVH packet tracing become as slow as or worse than single ray by the second bounce.

few "closed" models that is freely available and used for comparison. This causes it to actually have high bounce depth computations, whereas in the other models many rays can escape the scene fairly quickly. We note that the conference scene is not actually closed (rays can escape through the vents), so there are still a small number of rays that hit the background.

In Table 4 we report box tests, interval arithmetic tests, rays reordered, triangle tests, traversal steps, and primitive tests divided by the total number of rays. A SIMD box or triangle test counts as a single test, and we do not distinguish between the all hit and all miss tests (each count as one interval test). We feel these timing independent statistics are useful to understand where potential performance improvements come from. As the majority of intersection cost is in these five statistics, we feel the point of any improved traversal algorithm is to reduce these values in relation to single ray code.

	Single	SIMD	Packet	Adaptive	Exact
Box	46.6	23.1	49.4	16.3	14.0
IA	0.0	0.0	6.7	0.2	0.2
Reorder	0.0	0.0	0.0	3.2	8.9
Tri	6.0	4.7	7.5	4.9	4.2
Trav	46.6	23.1	7.9	4.5	4.5
Prim	6.0	4.7	3.1	2.2	2.2

Table 4: Performance independent statistics for 2 bounce diffuse path tracing on the Conference scene. All numbers are averages per ray for the full rendering. A SIMD box or triangle test counts as 1 test.

As we can see from Table 4, our algorithm successfully reduces the number of box tests by a large margin over single ray traversal (2.9×) while also decreasing the total number of triangle tests slightly. As our method is packet based, we also gain a large reduction in traversal steps (10.4×) and primitive tests (2.7×) per ray by amortizing these over large packets. The reduction in traversal steps gives our method a substantial geometry bandwidth advantage compared to SIMD or single ray approaches. Compared to exact reordering, approximately 2.8× fewer rays are reordered on average with only slight changes in other statistics.

As mentioned in Section 4, the interval arithmetic tests are only useful when they quickly report an all miss or an all hit for a large number of rays. These tests rely on ray coherence and so we would assume that they would be less useful as bounce depth increases. In Figure 6, we plot the average interval arithmetic benefit for our three bounce depths without our interval test disabling technique (to evaluate the maximum possible benefit) and see that this is the case. As mentioned in Section 4, in our implementation we disable the interval tests when the number of active rays decreases below a small multiple of the target SIMD width.

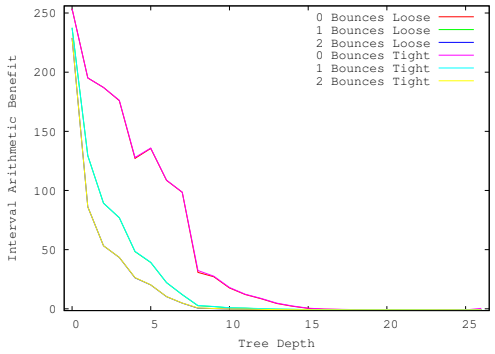

Figure 6: Interval arithmetic benefit versus tree depth for our three bounce depths. IA benefit is the number of ray-box tests that are skipped with interval tests less the number of interval tests used. The tighter intervals are always at least as good as the loose ones, but the difference is only slightly visible.

Finally, we note that increasing the number of samples per pixel has nearly no effect on the coherence of the secondary rays for diffuse path tracing. This is unsurprising as the incoherence comes from random directional scattering which is *not tied* to the amount of spatial antialiasing.

5.5 Increasing SIMD Width

While our results using SSE are already an improvement on previous methods, we highlight the possibility of greater improvements with larger SIMD width. Figure 7 shows the SIMD speedup for both box and triangle tests for the diffuse path traced conference scene with two bounces. For increasing SIMD width, our method achieves an increase in SIMD speedup (SIMD width multiplied by utilization) for box tests but has nearly no effect on triangle tests (on average only 2 rays reach leaf nodes for our method). Our SIMD utilization is highest for 4-wide SIMD, but the decrease in utilization is not by a factor of 2 for each increase in SIMD width (the bars are not flat across SIMD width). It is important to note that simple SIMD packet tracing also benefits from the increase in SIMD width as well. This is especially true as we reach very high SIMD widths where our reordering method still uses only 256 rays per input packet.

The majority of our SIMD utilization improvement comes from the higher levels in the BVH. As shown in Figure 8 for 16-wide SIMD, our method keeps SIMD utilization above standard packet tracing but only while enough rays are available. Once the input set

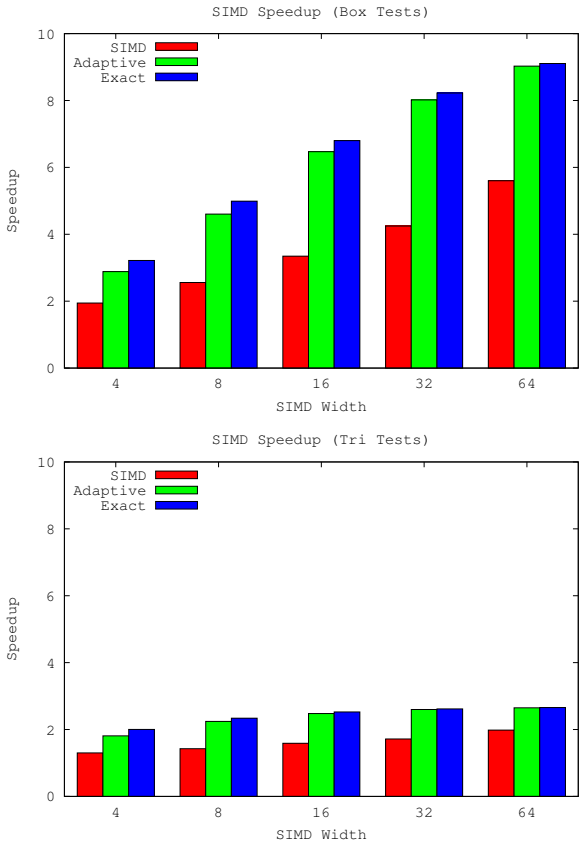

Figure 7: SIMD speedup for box tests (top) and triangle tests (bottom) for two bounce diffuse path tracing on the Conference scene. The SIMD packet tracing uses exactly SIMD width number of rays per packet while both reordering methods use 256 rays per packet. SIMD speedup is SIMD utilization multiplied by the SIMD width (for easier comparison across widths). While utilization goes down as the SIMD width gets larger, the amount of useful work done still increases for box tests. Adaptive reordering also regains nearly as much speedup as exact reordering but with a lower reordering cost.

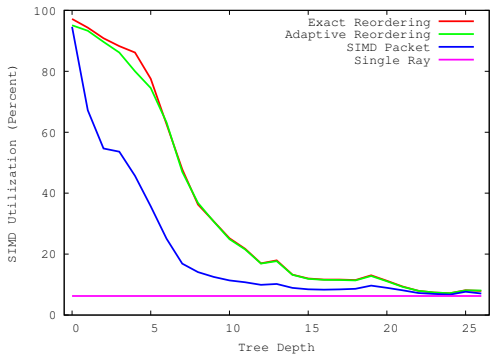

Figure 8: Comparison of SIMD packet tracing and our reordering methods for a target SIMD width of 16 (two bounce path tracing for conference scene). Our reordering method maintains higher SIMD utilization in the upper parts of the tree where SIMD coherence pays off the most. Adaptive reordering also has nearly as much benefit as exact reordering. All methods deteriorate towards the leaves to having only 1 or 2 rays active at a time.

of rays is below the SIMD width of our target architecture, we can no longer provide any useful reordering.

6 Discussion

In this paper, we have presented a new method that combines the benefits of BVH packet traversal with the benefits of breadth first ray tracing. Our method accepts arbitrary input packets and attempts to extract the maximum coherence available. The following are a number of important questions that we feel are important to discuss in relation to this work.

How is this different from previous approaches? The most similar work to our own is the Toro system from Pharr et al. [16]. In Toro, a coarse uniform grid is used as a queuing system for a finer resolution uniform grid. In a sense, this method can be seen as similar to ours where the reordering is chosen statically to be at every node in the coarse uniform grid (which in our system would be similar to grouping at say every few levels of the BVH). This approach, however, unnecessarily penalizes coherent rays by forcing them to queue up and be reordered (SIMD Stream Tracing [22] suffers from the same behavior). Similarly, incoherent rays that might need more fine grained scheduling than the coarse grid resolution chosen would also fall through the cracks.

In contrast, even our simple dynamic utilization threshold creates an effective set of reordering points for each packet individually. We don't foresee an effective method to statically choose a set of BVH nodes that would have similar properties.

Deep Coherent Ray Tracing [12] instead queued rays into a generic bin and attempted to produce packets from this general pool. These packets were then traversed through a kd-tree with traditional SIMD packet tracing. We feel it would be difficult to determine *a priori* which rays within a large group would follow similar traversal paths before actually tracing the rays.

Can we do better? As SIMD width increases for next generation hardware, we believe our approach will provide additional benefits. We have reported the SIMD speedup we would obtain for multiple SIMD widths that match current and potential future hardware systems. Our approach, however, does seem to approach the limits of SIMD utilization through naïvely accepting the input ray packets (at least with our data structure) and certainly leaves much to be desired (our 16-wide SIMD utilization is approximately 45%). While increasing our starting packet improves utilization further (we have separately tested 1024 rays per packet), this has a negative effect on rendering performance for our 4-wide system due to the large amount of cache misses caused by using such a large packet. A more detailed analysis of this tradeoff would warrant future investigation, however, from this simple experiment it is already clear that the resource requirements of extremely large ray packets are most likely untenable on current architectures.

Combining our approach with methods that resort larger batches of incoherent rays [12] might prove fruitful as well. Perhaps more promising, however, would be to switch to a method that could fully utilize SIMD units when the benefits of packets disappear. The method employed by Pixar for subdivision surfaces [4] works particularly well as the SIMD width is naturally tied to the subdivision algorithm. In our approach, this would be allowed through the optimized fast path for single rays. A traversal method of this kind might suggest switching to single rays sooner than we do now. We are cautious, however, of losing the bandwidth savings that packets provide at the higher levels of the tree (where coherence is very high).

This might suggest that building shallower trees as Reshetov [18] has done could be beneficial to increase SIMD utilization again. However, as always the goal is to actually reduce the amount of

work done, so simply using shallower trees would not accomplish this. In the limit, trees would disappear and every ray would be tested against every primitive (albeit at 100% SIMD utilization).

What about shader or texture coherence? The experiments in this paper explicitly avoid focusing on shading coherence. As mentioned in Section 4, we follow Boulos et al. [2] and group the packet by material. Currently, we do this in a linear sweep over the packet. For scenes with many materials, our simple approach would fail to provide enough rays in the initial packet to allow for substantial benefit. In these cases, we feel that previous solutions that could group rays from different shaders [6, 12, 16] seemingly provide the only way out. As in our description of our method, however, we distinguish between actual differences in material model and simply a difference in material inputs (same code, different data).

Do I need to worry about generating coherent packets? The original BVH packet traversal method was very sensitive to the order in which rays appeared in the original packet. In contrast, our method is essentially agnostic to the order the rays appear since we would reorder them almost immediately. However, given a coherent input packet our method like the original BVH packet traversal will perform better than given an incoherent packet. In a scenario with multiple shadow rays per shading point, it may be beneficial to explore different combinations to produce shadow packets (e.g. 64 shadow samples per shading point in each packet, instead of 64 packets with 1 shadow ray per shading point).

7 Conclusions and Future Work

For incoherent ray distributions, our method produces improvements in SIMD and packet coherence for a variety of SIMD targets. Correspondingly, global illumination systems could expect anywhere between a $2\times$ and $6\times$ speedup over a single ray implementation using our method (depending on SIMD width). We note, however, that our method achieves only approximately a $2\times$ gain over SIMD packet tracing for these types of rays (assuming 16-wide SIMD) so automatically generated SIMD packet code may be preferable from an implementation and maintenance standpoint. We believe our adaptive method represents the best performance achievable across arbitrary input ray distributions using ray packets for our current data structure.

Our method is simple and practical enough to demonstrate speedups within a current 4-wide SIMD architecture in the Manta Interactive Ray Tracer. As SIMD width increases, our relative performance gain over single ray code will increase as well. On architectures with support for gather, scatter, and bit counting operations, the relative frequency of reordering may be increased further for slight improvements due to the decreased cost for reordering. Our results with "exact reordering" seem to suggest that our current thresholds are sufficient for all scenes we tested. We also tested larger packets (1024 rays) and found that the resource requirements increased so much as to reduce rendering performance (due to L2 cache misses) . On resource constrained architectures, such as GPUs, this effect would be more likely to occur so there is a open question of choosing the right packet size with our method. While it is clear that SIMD utilization will increase as ray packet size increases if coherence is available, the resource requirements may outweigh the benefits. As a simple choice, we would use whatever size ray packet can be reasonably kept within the on chip resources of a given architecture.

We also found that the seemingly useful interval arithmetic approach from standard BVH packet traversal simply breaks down in the presence of incoherent secondary rays. It is useful to include these interval tests in our method to transparently accelerate coherent rays, however, a system focused on incoherent rays may not need them. It may be interesting to investigate geometric frustum based approaches, but we are unconvinced they would present a significant gain.

References

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

[2] S. Boulos, D. Edwards, J. D. Lacewell, J. Kniss, J. Kautz, P. Shirley, and I. Wald. Packet-based whitted and distribution ray tracing. *Proceedings of Graphics Interface 2007*, pages 177–184, 2007.

[3] S. Boulos, I. Wald, and P. Shirley. Geometric and Arithmetic Culling Methods for Entire Ray Packets. Technical Report UUCS-06-010, 2006.

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

[5] R. L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, 1986.

[6] P. Djeu, W. Hunt, R. Wang, I. Elhassan, G. Stoll, and W. Mark. Razor: An Architecture for Dynamic Multiresolution Ray Tracing. *ACM Transactions on Graphics (conditionally accepted)*, 2007.

[7] P. Hanrahan. Using caching and breadth first search to speed up ray tracing. In *Proceedings of Graphics Interface*, pages 55–61, 1986.

[8] Intel Corp. Intel Pentium III Streaming SIMD Extensions. http://-developer.intel.com/vtune/cbts/simd.htm, 2002.

[9] Intel Corp. Intel AVX. http://softwareprojects.intel.com/avx, 2008.

[10] T. Kato. Kilauea—-parallel global illumination renderer. *Parallel Computing*, 29(3):289–310, 2003.

[11] J. Mahovsky and B. Wyvill. Memory-Conserving Bounding Volume Hierarchies with Coherent Raytracing. *Computer Graphics Forum*, 25(2):173–182, 2006.

[12] E. Mansson, J. Munkberg, and T. Akenine-Moller. Deep Coherent Ray Tracing. *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 79–85, 2007.

[13] NVIDIA Corp. NVIDIA GeForce 9800 GTX Specification. http://-www.nvidia.com/object/geforce_9800gtx.html.

[14] S. Parker, S. Boulos, J. Bigler, and A. Robison. RTSL: a Ray Tracing Shading Language. *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 149–160, 2007.

[15] Paul Arthur Navrátil and Donald S. Fussell and Calvin Lin and William R. Mark. Dynamic Ray Scheduling to Improve Ray Coherence and Bandwidth Utilization. *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, 2007.

[16] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *Proceedings of SIGGRAPH*, pages 101–108, 1997.

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

[18] A. Reshetov. Faster Ray Packets-Triangle Intersection through Vertex Culling. *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 105–112, 2007.

[19] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: A many-core x86 architecture for visual computing. *ACM Transactions on Graphics, to appear*, 27(3), 2008.

[20] P. Shirley, C. Wang, and K. Zimmerman. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, 1996.

[21] I. Wald, S. Boulos, and P. Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 26(1):6:1–6:18, Jan. 2007.

[22] I. Wald, C. P. Gribble, S. Boulos, and A. Kensler. SIMD Ray Stream Tracing - SIMD Ray Traversal with Generalized Ray Packets and On-the-fly Re-Ordering. Technical Report UUSCI-2007-012, 2007.

[23] I. Wald, P. Slusallek, C. Benthin, and M. Wagner. Interactive Rendering with Coherent Ray Tracing. In *Proceedings of EUROGRAPHICS*, pages 153–164, 2001.

[24] D. Wexler, L. Gritz, E. Enderton, and J. Rice. GPU-accelerated high-quality hidden surface removal. *Proceedings of Graphics Hardware*, pages 7–14, 2005.