Accelerating Ray Tracing with Origin Offsets

T. Zoet
Utrecht University
Domplein 29
3512 JE Utrecht
ts.zoet@gmail.com

J. Bikker
Utrecht University
Domplein 29
3512 JE Utrecht
bikker.j@gmail.com

Abstract

A significant portion of rendering time in ray tracing consists of the traversal of an acceleration structure. While methods designed to reduce traversal cost often target the top of the tree, we instead propose a method that targets the lowest levels, by offsetting ray origins just before starting traversal. Our method moves ray origins out of nodes deep in the tree, avoiding the scattered memory access associated with these rarely visited nodes. We propose a precomputed set of hemispheres, placed on the surfaces, which is guaranteed not to contain any geometry, thus allowing rays starting inside a hemisphere to be moved to the hemispheres boundary. Our method is compatible with most acceleration structures and does not require access to the actual traversal implementation.

Keywords

Ray tracing, acceleration structures

1 INTRODUCTION

Ray tracing has wide applications. It is used in collision detection algorithms, simulations of physical phenomena and graphics. Recent advances in ray tracing hardware made real-time ray tracing accessible to the masses. In the past year, the first video games that employ ray tracing for rendering were released.

To perform efficient intersection tests between rays and a collection of objects, acceleration structures are used. The two most common acceleration structures in modern high-performance renderers, the kD-tree and bounding volume hierarchy (BVH), aim to bring the average case time complexity of tracing a single ray down to near logarithmic.

In this paper, we propose a method that aims to improve ray tracing performance by offsetting extension rays just before starting traversal. This is based on the observation that many rays visit the leaf node they originate from without finding an intersection. These unnecessary visits are relatively expensive, due to a scattered memory access pattern. Our aim is to prevent these visits by moving the ray origins along the ray direction. This shortening will often move the rays out of the leaf nodes.

In our method a set of hemispheres is placed on the surface of the scene geometry. The radius of each hemisphere is such that it does not to contain any geometry. For each ray origin we fetch the nearest hemispheres, which are then used to replace the ray origin by the intersection of the ray and the hemispheres. If the origin is moved far enough, the ray is moved out of the leaf node it originated from and potentially one or more nodes higher in the acceleration structure. Our aim is to outweigh the overhead of offset calculation by the reduced traversal cost.

2 RELATED WORK

Accelerating the intersection of rays and scene geometry is achieved using various data structures and algorithms. These methods can be put into roughly three categories:

1. Acceleration structures. The most commonly used acceleration structures, the kD-tree and BVH, aim to bring the cost of tracing a single ray from linear to near logarithmic time. They do this by recursively partitioning space and objects, respectively.

2. Amortization of traversal cost. By grouping rays together in packets, the cost of fetching the acceleration structure data can be amortized over many rays.

3. Efficient traversal of the acceleration structure. These methods often make use of high-level knowledge about rays, such as the difference between nearest-hit and any-hit (occlusion) rays, or connectivity between rays, to guide traversal towards parts of the tree that are more likely to intersect the rays.

The above categorization will not cover all methods. Some can be put into more than one category, whereas others will defy all categorization. The following sections provide a brief summary of the main works in each category, providing a context for the contributions in this paper.

2.1 Acceleration structures

Fujimoto et al. [FTI86] use a uniform grid to store scene geometry. As a ray traverses the grid, it intersects the objects in each cell it passes. Glassner [Gla84] uses an adaptive scheme, placing objects in a recursively partitioned octree. Cleary and Wyvill [CW88] provide an analysis of the uniform grid, showing that it, at that time, outperformed hierarchical methods.

Earlier, the kD-tree (or binary tree) was introduced by Bentley [Ben75], although not in the context of ray tracing. The kD-tree has found wide application. Goldsmith and Salmon [GS87] introduce a simple heuristic to guide the construction of spatial subdivision trees. MacDonald and Booth [MB90] propose the refined surface area heuristic (SAH) and show that it performs much better than using a fixed split order.

Clark [Cla76] introduced the BVH, which partitions objects rather than space. Construction of the BVH is similar to the kD-tree, and allows the use of e.g. the surface area heuristic.

Havran [Hav00] provides an in-depth analysis of a wide range of acceleration structures, including the kD-tree and BVH, and concludes that the kD-tree gives the best ray tracing performance. Wald [Wal07] greatly improves the speed of BVH construction and Stich [SFD09] solves the problem of scenes with non-uniformly sized triangles. Together, this closes the performance gap between the kD-tree and BVH. Today the BVH is the most widely used acceleration structure in renderers.

2.2 Amortization

Wald et al. [WSBW01] group several rays together in a single packet. The size of this packet is generally chosen to be the width of the vector registers on the CPU. For SSE capable CPUs, packet traversal yields a 2-3x performance improvement when compared to single ray traversal.

Reshetov et al. [RSH05] search for the first node in an acceleration structure for which both subtrees contain a leaf node that intersects a group of rays. Once the entry point has been found, individual rays start traversal at this point. The process is further optimized by Fowler et al. [FCM09], who find deeper entry points, with fewer traversal steps.

Overbeck et al. [ORM08] use large packets of up to 1024 rays and propose a traversal algorithm specifically aimed at reflection and refraction rays. These rays suffer from quickly degrading coherence as the number of bounces for each path increases.

2.3 Improved traversal

Boulos and Haines [BH10] show that occlusion rays often dominate the total rendering time. It is therefore worthwhile to devise specialised traversal strategies for occlusion rays and nearest-hit rays. MacDonald and Booth [MB90] were the first to propose an alternative traversal algorithm. They add neighbor links between the leaves of a kD-tree. Once a ray has been traced, any extension ray originating at the intersection point can then be traced by following these links. This traversal method prevents visiting many internal nodes, at the cost of additional memory usage. Havran et al. [HBZ98] show that ropes result in a 10-20% reduction of total rendering time, compared to standard kD-tree traversal. The gains are later improved to up to 35% by Havran and Bittner [HB07]. Hendrich et al. [HPMB19] use convex frustum shafts to cull parts of a BVH. Djeu et al. [Djeu09] reduce the traversal time of occlusion rays, by marking leaf nodes contained by geometry as volumetric occluders and terminating traversal in these nodes. Ize and Hansen [IH11] introduce the Ray Termination Surface Area Heuristic (RTSAH), which significantly reduces the number of visited nodes for occlusion rays.

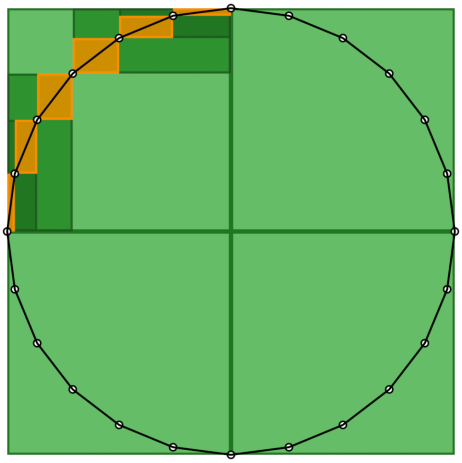


Figure 1: Part of a BVH built on a circle. In shades of green internal nodes. In orange leaf nodes. Note that the distance required to move a ray starting on the surface out of a leaf node is small.

The work presented in this paper falls under the third category. We build a set of hemispheres on top of the acceleration structure that are guaranteed not to contain any geometry. We use the hemispheres to calculate offsets for extension rays, skipping deep parts of the hierarchy.

3 OFFSETTING RAYS

Testing a ray against a leaf node is more expensive than testing against an internal node. This has two main reasons. First, leaf nodes are visited less frequently than the shallower internal nodes, and are therefore less likely to be in the cache. The same holds for the primitive data in the leaf node. Second, intersecting a leaf

node involves intersecting the primitives it stores. Most methods that aim to accelerate traversal focus on skipping internal nodes or amortizing traversal costs over multiple rays and quickly reaching the leaf nodes. We argue that it could be beneficial to focus on skipping leaf nodes, avoiding scattered memory access and primitive intersection tests.

Consider rendering an opaque, tessellated sphere (Figure 1). After the primary rays have been traced, extension rays starting on the sphere's surface will not hit any geometry, as the sphere is a convex object and all rays will travel away from its surface. Still, traversal will first traverse down into the leaf node where the previous ray ended, since the new ray's origin is contained in the leaf's bounds. Offsetting the ray origin along a short distance could be enough to move it out of the leaf node. A larger offset would then cull additional nodes above and adjacent to the leaf node. While a sphere is the best-case scenario due to its convexity, the idea of offsetting ray origins extends to more complex scenes.

To calculate offsets we use an auxiliary data structure, in the form of a set of hemispheres. In a preprocessing step we place one or more hemispheres on each triangle. At each hemisphere location, we determine the nearest intersection on the positive side of the plane. The distance to this intersection is then stored as the radius of the hemisphere. The volume of this hemisphere is thus guaranteed to be empty. During rendering, for each ray, we retrieve the hemispheres attached to the triangle it originates from. The origin offset is then calculated using a ray-sphere intersection test. If the ray intersects multiple hemispheres, the greatest intersection distance is used. After offsetting the ray origin, traversal is started. Note that our method is oblivious to the chosen acceleration structure and the actual traversal, and thus orthogonal to other optimizations.

We limit our discussion to polygonal scenes and reflected rays, although the concepts are potentially applicable to other scenes and transmitted rays.

In the next two subsections we will describe various potential hemisphere placement strategies. Section 3.1 covers how the hemisphere locations can be chosen such that they best achieve our goal: moving the rays out of as many leaf nodes as possible. While the most optimal solution is of course placing an infinite number of hemispheres, to optimize execution time a balance between traversal savings and overhead must be found. Section 3.2 describes low-level optimizations to reduce overhead, such as a carefully chosen memory layout, caching behaviour and vectorization.

3.1 Hemisphere sets

The set of hemispheres must be chosen in such a way that the average ray origin offset is maximal. At the same time, we wish to limit the number of hemispheres,

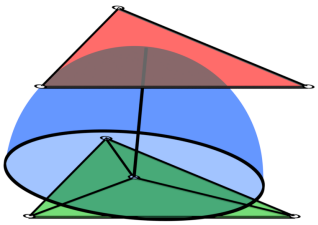


Figure 2: A hemisphere (blue) placed in the center of a triangle (green). The radius is limited by the triangle (red) above it.

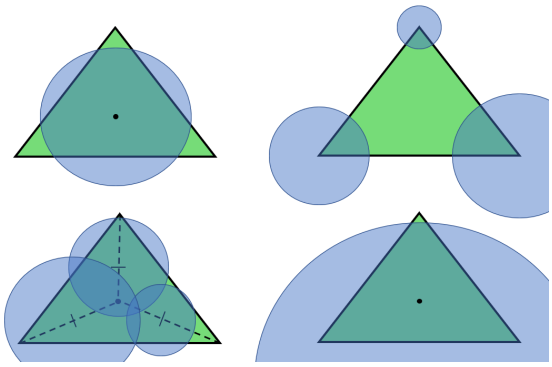


Figure 3: Visualization of the placement strategies. Top row: center sphere set and vertex sphere set. Bottom row: median sphere set and polygon sphere set.

to reduce query overhead. We investigate four placement strategies that balance these factors. The strategies are visualized in Figure 3.

Center Sphere Set. The simplest method is to store a single hemisphere located at the center of each triangle (Figure 2). At this location the hemisphere will in general have the most potential for moving a ray out of the triangle's bounding box.

Vertex Sphere Set. Alternatively, we can place a hemisphere on each triangle vertex. This method is more resistant to occluding geometry at the center: if only one area of the triangle is occluded there is still potential for a large enough hemisphere being placed on one or more of the vertices to offset nearby rays far enough to leave the triangle bounds. There is an important downside: if a vertex is part of a concave corner of the surface geometry the radius of the hemisphere will be 0.

Median Sphere Set. To solve the problem of concave geometry we can move the hemispheres closer to the triangle center. One option is to place the hemispheres halfway between each vertex and the triangle center, the medians, i.e. the line segments from the vertices to the midpoint of the opposite edges. Together, these hemispheres can cover a large area of the triangle.

Polygon Sphere Set. In some situations the center sphere set could perform better with only a slight modification. Consider a scene with many groups of adjacent triangles lying in the same plane, essentially forming (non-convex) polygons. A triangle in such a group

could make use of the hemisphere of one of its neighbours. This neighbouring hemisphere might be much larger if the triangle itself lies closer to an occluding part of the scene than the neighbour. If the neighbouring hemisphere covers a large part of the triangle, it can offset many rays for much larger distances than the triangle's own hemisphere. Selecting the optimal hemisphere from all neighbours is a hard optimization problem. A potentially good approximation would be to select the largest of all hemispheres and assigning this to each triangle of the polygon.

3.2 Optimization

By placing a fixed number of hemispheres per triangle, retrieval has constant time complexity. It also allows keeping the querying code simple, minimizing the strain on the instruction cache. Furthermore, it opens up options for aligning the data to cache lines and efficient vectorized intersection tests.

Center Sphere Set. The center sphere set has the lowest storage requirements. A hemisphere record requires 16 bytes: 4 for the radius and 12 for the position, which comfortably fits in a single cache line. Alternatively, the hemisphere position can be deduced from the triangle vertex positions. Although this limits storage to 4 bytes, this is only beneficial if the vertex data is in the cache.

Vertex Sphere Set. This method uses 3 times more memory than the center sphere set. This may still fit in a single cache line, although padding to 64 bytes may be necessary to avoid extra cache line accesses. Alternatively, the hemisphere data can be stored with the vertex data, adding only the hemisphere radius to each vertex position. Obviously, this change will impact other parts of the renderer. It could degrade performance for the acceleration structure or might simply be impossible to implement (e.g. when using indexed triangles, where vertices can be shared by multiple triangles).

Median Sphere Set. The median sphere set can be used with or without the center hemisphere. Also storing the center hemisphere will not add much overhead. If the medians are recalculated from the vertices, the first step is calculating the center, and storing the 4^{th} radius allows aligning the data to 16 bytes. If the hemisphere centers are stored, adding the extra hemisphere will increase the per triangle data to 64 bytes, exactly a cache line.

Polygon Sphere Set. The polygon sphere set can be stored and queried in two distinct ways. First, using the method for the center sphere set. In that case the same optimizations can be applied. Second, using an indexed lookup. Since hemispheres are used by multiple triangles, they need not be stored multiple times. This can reduce memory overhead, if we assume that we store the hemisphere centers as well as the radii. Total memory usage would be one index per triangle to fetch the hemisphere, and one hemisphere per polygon. The number of polygons will depend on the nature of the scene and how many triangles are in the same plane as their neighbours.

Further optimization can be done using SIMD instructions. For example, using SSE allows querying the center sphere set with 4 rays at once, amortizing memory cost over the 4 rays.

4 RESULTS

The scenes used in our experiments are shown in Table 1. For each of the scenes several viewpoints are selected. Results are averaged over these viewpoints. Three ray classes are tested to account for different behaviours: ambient occlusion (AO) rays, diffuse rays and shadow rays. AO rays are rendered using several radii set to a fixed percentage of the scene size. Each primary ray that finds an intersection is used to generate exactly one AO ray. For the diffuse renders each path is extended in a random direction until it finds a light, leaves the scene or reaches a maximum recursion depth. For the shadow rays a single point light is added to each scene.

Two types of statistics are collected: the number of visited nodes per ray (split by internal nodes and leafs) and runtimes. Note that for the diffuse renders the number of nodes is calculated per ray rather than per path. Generating and tracing primary rays, writing to the image buffer or any other steps are excluded from the measurements.

All experiments were run single-threaded on a 2.60GHz Intel i7-6700HQ with 8GB DDR4 RAM. For BVH construction and intersection tests the Intel Embree ray tracing library (version 3.5.2) was used [WWB*14]. All images were rendered at 4096x4096 pixels.

4.1 Traversal

Table 2 shows the number of visited nodes and leafs for AO rays. The rays were given a length, or radius, of a percentage of the scene size. The **baseline** column shows the absolute number of visited internal nodes and leaves. These numbers increase as the radius increases. When using any of the hemisphere sets to offset the rays, the average number of visited nodes and leaves decreases, as shown by each column corresponding to the respective hemisphere set.

The median hemisphere set consistently outperforms all other sets. This is only by a small margin however, suggesting that a larger number of hemispheres per triangle has little added benefit. When comparing the results of the vertex sphere set with the center sphere set, we see that it performs better in some scenes, but worse in others. This seems related to the nature of the geometry:



buddha	bunny	sibenik cathedral	dragon	hairball	living room	rungholt	sponza crytek	sponza dragon
1.1M	144k	76K	871K	2.9M	581K	5.8M	262K	1.1M
16.6	2.2	1.1	13.3	43.9	8.9	88.7	4.0	17.3

Table 1: Overview of the scenes used in the experiments. For each scene we report the triangle count and the memory cost (in MB) of the 16-byte center sphere set.

the architectural scenes that contain more concave geometry, such as the Cathedral, make the vertex sphere set less useful. The polygon sphere set shows no improvement over the center sphere set, suggesting that selecting the largest hemisphere is a poor optimization strategy.

Scenes such as Buddha, Bunny and Dragon show a reduction of visited leaf nodes by up to 85% for short rays. For longer rays reductions are not as large, although still at least 30%. The strong reduction for short rays is due to many rays receiving an offset that is larger than their length, causing traversal to terminate in the root node. Figure 4 shows this for the Dragon scene: the fraction of rays for which the calculated offset (using the center sphere set) is larger than the ray length is high for short radii and then drops quickly. For scenes that do not contain both large open spaces and convex geometry, this fraction drops much faster.

The Rungholt scene contains almost exclusively axis-aligned geometry. Because of this, the majority of extension rays do not start inside the bounding box of the leaf node the primary ray ended in, and there is little improvement with offsets. Also shown in Table 2 are results for the Rungholt scene rotated by 15°. The baseline shows that the number of visited internal nodes increases by about 20%, while the number of visited leaves doubles. When using offsets, this increase is partially undone.

The Sponza Dragon scene is the Sponza Crytek scene with the dragon model added in its center. While the Sponza Crytek scene shows negligible reduction for long rays, with the dragon relative improvements become larger, showing that the sphere sets adapt to varying geometry. The Sponza Dragon scene could be seen as a common use case for e.g. games, where detailed character models move through buildings.

Table 3 and Table 4 show the results for diffuse and shadow rays. The overall behaviour for all scenes and sphere sets is similar to that of the ambient occlusion rays, with some slight differences. The scenes in which the AO rays saw the largest reductions do not fare as well. Conversely, some of the worst performers for long AO rays (such as the Living Room and Sponza Crytek scenes) do show more significant reductions.

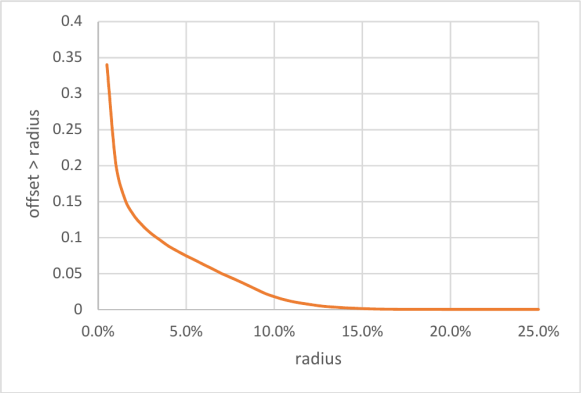


Figure 4: Fraction of AO rays for which the offset is larger than the radius (in the Dragon scene with the center sphere set).

4.2 Runtimes

The traversal results have shown that the median sphere set consistently outperforms all other methods. However, this is only by a small margin. The center sphere set shows improvements within a range of several percent. Because the center sphere set uses only a quarter of the memory that is required by the median sphere set, the results in this section only cover the center sphere set. The implementations of the alternatives have not been thoroughly optimized, and would probably not perform better, as calculating the offsets is mostly memory bound. The center hemispheres were stored as a simple array, retrievable using the triangle index. Each hemisphere occupies 16 bytes: 3 floats for the center, 1 float for the radius.

Figure 5 shows the ambient occlusion ray trace times. For short rays (1% of the scene size) the Dragon scene is traced using only 73% (827ms vs 1132ms) of the original time. As the ray length increases this difference becomes smaller, with 85% (1339ms vs 1576ms) at 50% of the scene size. Rungholt is traced slower by up to 3% at all ray lengths. For all but the shortest rays, the same happens for the Cathedral and Sponza Crytek scenes. In all cases, performance improvements are in accordance with the traversal improvements reported in the previous section: more saved traversal steps means shorter trace times.

scene	radius	baseline	center	vertex	median	polygon
buddha	1%	7.71 / 0.80	46% / 19%	43% / 17%	44% / 15%	45% / 19%
buddha	5%	8.45 / 0.94	61% / 34%	59% / 32%	59% / 31%	60% / 34%
buddha	10%	8.73 / 1.01	63% / 39%	61% / 38%	62% / 37%	63% / 40%
buddha	25%	8.90 / 1.06	64% / 43%	63% / 42%	63% / 41%	64% / 44%
bunny	1%	6.50 / 0.26	47% / 40%	38% / 32%	41% / 33%	45% / 39%
bunny	5%	7.10 / 0.33	63% / 54%	58% / 47%	59% / 48%	62% / 53%
bunny	10%	7.40 / 0.39	66% / 61%	62% / 55%	63% / 56%	66% / 60%
bunny	25%	7.55 / 0.52	71% / 72%	68% / 68%	69% / 68%	70% / 71%
cathedral	1%	5.78 / 0.12	68% / 84%	79% / 93%	57% / 73%	80% / 86%
cathedral	5%	6.16 / 0.18	85% / 90%	89% / 95%	81% / 83%	91% / 91%
cathedral	10%	6.60 / 0.26	89% / 94%	93% / 97%	86% / 89%	92% / 94%
cathedral	25%	7.51 / 0.62	93% / 98%	95% / 99%	90% / 96%	95% / 98%
dragon	1%	6.32 / 0.43	55% / 17%	49% / 15%	50% / 15%	54% / 17%
dragon	5%	7.08 / 0.57	66% / 40%	62% / 39%	62% / 39%	66% / 40%
dragon	10%	7.43 / 0.65	69% / 50%	66% / 49%	66% / 49%	69% / 50%
dragon	25%	7.41 / 0.66	71% / 53%	67% / 52%	68% / 52%	70% / 53%
hairball	1%	9.18 / 1.45	76% / 58%	71% / 64%	66% / 43%	73% / 58%
hairball	5%	12.12 / 2.74	93% / 81%	93% / 83%	91% / 74%	92% / 81%
hairball	10%	14.16 / 3.76	94% / 87%	94% / 89%	93% / 83%	94% / 87%
hairball	25%	16.08 / 4.66	96% / 90%	95% / 92%	94% / 87%	95% / 90%
livingroom	1%	5.17 / 0.10	60% / 70%	64% / 71%	49% / 64%	77% / 77%
livingroom	5%	5.62 / 0.18	84% / 85%	84% / 85%	77% / 82%	88% / 88%
livingroom	10%	5.92 / 0.24	90% / 89%	89% / 89%	86% / 87%	93% / 92%
livingroom	25%	6.39 / 0.35	92% / 93%	92% / 93%	90% / 92%	95% / 95%
rungholt	1%	6.55 / 0.22	83% / 99%	80% / 98%	81% / 98%	82% / 99%
rungholt	5%	7.59 / 0.44	93% / 100%	91% / 99%	92% / 99%	93% / 100%
rungholt	10%	8.07 / 0.54	94% / 100%	92% / 99%	93% / 99%	94% / 100%
rungholt	25%	8.18 / 0.59	95% / 100%	93% / 99%	94% / 99%	95% / 100%
rungholtrotated	1%	7.98 / 0.45	77% / 70%	73% / 65%	74% / 64%	78% / 82%
rungholtrotated	5%	9.15 / 0.74	88% / 84%	86% / 82%	86% / 81%	90% / 91%
rungholtrotated	10%	9.49 / 0.89	90% / 88%	88% / 86%	89% / 86%	92% / 93%
rungholtrotated	25%	9.66 / 0.95	91% / 90%	89% / 88%	90% / 88%	92% / 94%
sponzacrytek	1%	10.33 / 0.18	86% / 80%	90% / 79%	79% / 74%	90% / 82%
sponzacrytek	5%	11.05 / 0.28	95% / 89%	95% / 88%	92% / 86%	96% / 90%
sponzacrytek	10%	11.62 / 0.39	96% / 93%	96% / 92%	94% / 91%	96% / 93%
sponzacrytek	25%	9.16 / 0.60	98% / 98%	98% / 97%	98% / 97%	99% / 98%
sponzadragon	1%	10.96 / 0.26	86% / 71%	87% / 70%	82% / 67%	87% / 73%
sponzadragon	5%	12.50 / 0.43	93% / 85%	93% / 84%	92% / 83%	94% / 86%
sponzadragon	10%	13.66 / 0.60	95% / 91%	95% / 90%	94% / 89%	95% / 91%
sponzadragon	25%	11.60 / 0.74	97% / 95%	97% / 95%	96% / 95%	97% / 96%

Table 2: Ambient occlusion traversal. **radius**: ray length relative to scene size. **baseline**: #visited internal nodes / #visited leaf nodes per ray without offset. **sphere sets**: visited nodes / leaves relative to baseline.

The diffuse trace times are displayed in Figure 6. The results follow the same pattern as for the AO rays: a loss of 3% for Rungholt and gains up to 20% for the Dragon. The results for shadow rays, as shown in Figure 7, again follow the same pattern.

When combining the observations about visited nodes and trace times, we can draw an important conclusion about the computational overhead of offset calculation: this overhead is small. Consider the Rungholt scene, where the reduction of visited leaf nodes is 1% or less for all ray types, and trace times increase by only 3%.

5 CONCLUSION

We presented a simple auxiliary data structure to accelerate ray tracing. We precompute a set of hemispheres located on the surface of the scene's geometry that are guaranteed not to contain any objects. When given an extension ray, we retrieve the hemispheres that start on the same geometric primitive and offset the ray origin. With a large enough offset, the ray will be moved out of the leaf node it started in, thus saving expensive intersection tests at the bottom of the tree during traversal. Additional internal nodes may also be culled if the ray is moved out of their bounding box.

Improvements vary between scenes. In the most optimal case, the number of visited leaf nodes can be reduced by several factors. Additionally, there is a large reduction of the number of visited internal nodes. However, this is only for short occlusion rays in scenes containing large open spaces and convex geometry. On the

scene	baseline	center	vertex	median	polygon
buddha	9.93 / 2.38	68% / 32%	67% / 31%	67% / 29%	68% / 32%
bunny	8.12 / 2.24	74% / 45%	70% / 42%	71% / 40%	73% / 45%
cathedral	10.09 / 2.29	94% / 91%	96% / 97%	91% / 86%	96% / 93%
dragon	9.49 / 2.42	74% / 48%	71% / 42%	71% / 40%	74% / 49%
hairball	19.62 / 8.39	95% / 83%	95% / 87%	94% / 78%	95% / 84%
livingroom	9.21 / 2.18	92% / 87%	92% / 88%	90% / 82%	95% / 92%
rungholt	9.35 / 1.02	95% / 99%	93% / 97%	94% / 97%	95% / 99%
rungholtrotated	11.99 / 2.16	89% / 70%	87% / 66%	88% / 65%	92% / 85%
sponzacrytek	16.20 / 4.20	96% / 93%	96% / 94%	94% / 91%	97% / 94%
sponzadragon	18.77 / 4.87	95% / 91%	95% / 91%	94% / 89%	95% / 92%

Table 3: Diffuse traversal. **baseline**: #visited internal nodes / #visited leaf nodes per ray without offset. **sphere sets**: visited nodes / leaves relative to baseline.

scene	baseline	center	vertex	median	polygon
buddha	10.18 / 0.40	72% / 47%	70% / 44%	71% / 44%	72% / 47%
bunny	10.42 / 0.15	81% / 64%	78% / 57%	79% / 58%	80% / 64%
cathedral	7.27 / 0.05	93% / 89%	94% / 94%	90% / 80%	96% / 90%
dragon	8.15 / 0.18	71% / 34%	69% / 32%	69% / 32%	70% / 34%
hairball	16.66 / 1.43	97% / 90%	97% / 91%	96% / 85%	96% / 89%
livingroom	6.38 / 0.05	91% / 77%	90% / 76%	88% / 70%	93% / 83%
rungholt	6.96 / 0.08	93% / 99%	91% / 97%	92% / 98%	92% / 99%
rungholtrotated	8.78 / 0.20	88% / 79%	86% / 72%	87% / 73%	87% / 82%
sponzacrytek	15.61 / 0.09	97% / 86%	96% / 83%	95% / 81%	97% / 87%
sponzadragon	17.71 / 0.14	96% / 78%	95% / 76%	95% / 74%	96% / 79%

Table 4: Shadow traversal. **baseline**: #visited internal nodes / #visited leaf nodes per ray without offset. **sphere sets**: visited nodes / leaves relative to baseline.

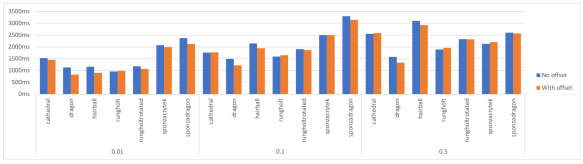


Figure 5: Ambient occlusion ray trace times.

other end of the spectrum there are scenes for which there is next to no reduction of visited nodes. Changes in actual trace time range from speedups of up to 25% to a loss of 3%. This last number indicates that the overhead of offset calculation is low.

The memory overhead added by the hemisphere set is linear in the scene size, requiring a fixed number of bytes per triangle. The optimized hemisphere set used in this paper uses 16 bytes per triangle. The hemisphere set is also completely independent of the acceleration structure, making it compatible with any form of traversal of the BVH or even alternative acceleration structures.

6 FUTURE WORK

Only the center sphere set implementation was thoroughly optimized. While the alternatives offer little extra reduction of visited nodes at the cost of several times more memory overhead, it might be worthwhile to investigate their performance when properly optimized.

The polygon sphere set could perform better than the center sphere set with a better heuristic for selecting

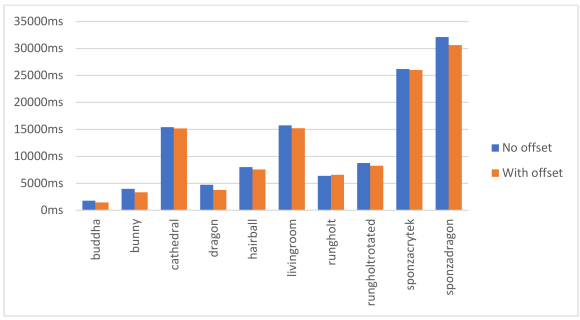


Figure 6: Diffuse ray trace times.

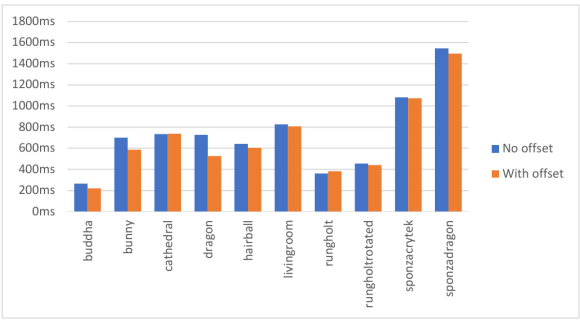


Figure 7: Shadow ray trace times.

the optimal hemisphere. Taking into account additional properties, such as the area of overlap between a hemisphere and the triangles, might result in a higher quality hemisphere set.

Concave geometry has been shown to result in little reduction of visited nodes. This may be improved by using other methods to offset rays, such as other geometric primitives that better represent empty spaces in these areas.

A deciding factor for the viability of hemisphere sets is how they perform in a GPU renderer. Offset calculation will have a different overhead and the traversal algorithms used on the GPU tend to be different.

Efficient construction of the hemisphere sets was not a subject of this work. This will become especially relevant for dynamic scenes, where quick reconstruction of (parts of) the hemisphere set will be needed.

7 ACKNOWLEDGEMENTS

Test scenes are taken from Morgan McGuire's Computer Graphics Archive [McG17]. We thank the anonymous reviewers for their valuable comments.

8 REFERENCES

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Communications of the ACM 18*, 9 (1975), 509–517.

[BH10] BOULOS S., HAINES E.: Sorted bvhs. *Ray Tracing News 23*, 2 (2010), 6.

[Cla76] CLARK J. H.: Hierarchical geometric models for visible surface algorithms. *Communications of the ACM 19*, 10 (1976), 547–554.

[CW88] CLEARY J. G., WYVILL G.: Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer 4*, 2 (1988), 65–83.

[FCM09] FOWLER C., COLLINS S., MANZKE M.: Accelerated entry point search algorithm for real-time ray-tracing. In *Proceedings of the 25th Spring Conference on Computer Graphics* (2009), ACM, pp. 59–66.

[FTI86] FUJIMOTO A., TANAKA T., IWATA K.: Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications 6*, 4 (1986), 16–26.

[Gla84] GLASSNER A. S.: Space subdivision for fast ray tracing. *IEEE Computer Graphics and applications 4*, 10 (1984), 15–24.

[GS87] GOLDSMITH J., SALMON J.: Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications 7*, 5 (1987), 14–20.

[Hav00] HAVRAN V.: *Heuristic ray shooting algorithms*. PhD thesis, Ph. d. thesis, Department of Computer Science and Engineering, Faculty of â, 2000.

[HB07] HAVRAN V., BITTNER J.: Ray tracing with sparse boxes. In *Proceedings of the 23rd Spring Conference on Computer Graphics* (2007), ACM, pp. 49–54.

[HBZ98] HAVRAN V., BITTNER J., ZÁRA J.: Ray tracing with rope trees. In *14th Spring Conference on Computer Graphics* (1998), pp. 130–140.

[HPMB19] HENDRICH J., POSPÍŠIL A., MEISTER D., BITTNER J.: Ray classification for accelerated bvh traversal. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 49–56.

[IH11] IZE T., HANSEN C.: Rtsah traversal order for occlusion rays. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 297–305.

[MB90] MACDONALD J. D., BOOTH K. S.: Heuristics for ray tracing using space subdivision. *The Visual Computer 6*, 3 (1990), 153–166.

[McG17] MCGUIRE M.: Computer graphics archive, July 2017. `https://casual-effects.com/data`. URL: `https://casual-effects.com/data`.

[ORM08] OVERBECK R., RAMAMOORTHI R., MARK W. R.: Large ray packets for real-time whitted ray tracing. In *2008 IEEE Symposium on Interactive Ray Tracing* (2008), IEEE, pp. 41–48.

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-level ray tracing algorithm. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 1176–1185.

[SFD09] STICH M., FRIEDRICH H., DIETRICH A.: Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009* (2009), ACM, pp. 7–13.

[Wal07] WALD I.: On fast construction of sah-based bounding volume hierarchies. In *2007 IEEE Symposium on Interactive Ray Tracing* (2007), IEEE, pp. 33–40.

[WSBW01] WALD I., SLUSALLEK P., BENTHIN C., WAGNER M.: Interactive rendering with coherent ray tracing. In *Computer graphics forum* (2001), vol. 20, Wiley Online Library, pp. 153–165.

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: a kernel framework for efficient cpu ray tracing. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 143.