# Ray-Specialized Acceleration Structures for Ray Tracing

Warren Hunt[*]
University of Texas at Austin

William R. Mark[†]
Intel Graphics Research
University of Texas at Austin

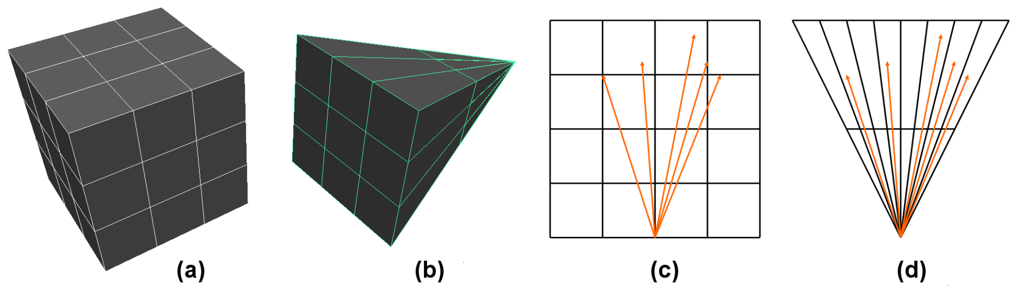

Figure 1: Ray tracing acceleration structures can be made more efficient by choosing split planes that are parallel or nearly-parallel to the rays being traced (subfigure d). For rays that share a common or near-common origin, this choice can be made most simply by building an acceleration structure that uses axis-aligned split planes specified in a space transformed by a perspective projection (subfigure b).

## Abstract

The key to efficient ray tracing is the use of effective acceleration data structures. Traditionally, acceleration structures have been constructed under the assumption that rays approach from any direction with equal probability. However, we observe that for any particular frame the system has significant knowledge about the rays, especially eye rays and hard/soft shadow rays. In this paper we demonstrate that by using this information in conjunction with an appropriate acceleration structure – a set of one or more perspective grids – that ray tracing performance can be significantly improved over prior approaches. This acceleration structure can easily be rebuilt per frame, and provides significantly improved performance for rays originating at or near particular points such as the eye point and the light source(s), without sacrificing the ability to trace arbitrary rays. We demonstrate true real-time frame rates on a game-like scene rendered on an eight-core desktop PC at 1920x1200 resolution for primary visibility, and hard shadows, along with lower frame rates for Monte Carlo soft shadows. In particular, we demonstrate the fastest hard shadow ray-tracing results that we are aware of. We argue that the perspective grid acceleration structure provides insight into why the Z buffer algorithm is faster than traditional ray tracing and shows there is a useful continuum of visibility algorithms between the two traditional approaches.

**Index Terms:** I.3.7 [Computing Methodologies]: COMPUTER GRAPHICS—Three-Dimensional Graphics and Realism

[*]e-mail:whunt@cs.utexas.edu
[†]e-mail:billmark@cs.utexas.edu

## 1 Introduction

Visible surface computations are some of the most costly computations in a typical real-time rendering system. The choice of a particular visible surface algorithm also determines many of the capabilities and limitations of a rendering system.

In modern rendering systems, the visible surface problem is typically solved by either a Z-buffer-like algorithm or by ray tracing. The Z buffer algorithm and its relatives such as the REYES algorithm [5] are fast, but strictly limit the kinds of rays that can be traced.

In this paper, we show that Z-buffer-like performance can be achieved in a ray-tracing system by using multiple specialized acceleration structures. Each acceleration structure is specialized to provide very high performance for particular kinds of rays or beams. More specifically, we describe an acceleration structure that is optimized for rays or beams originating at or near a particular point. One such acceleration structure is used for eye rays, and an additional such acceleration structure is used for shadow rays to each light source. A single geometric object or polygon may be stored in more than one of these acceleration structures. Typically, these acceleration structures must be rebuilt every frame as the eye and light positions change even if the geometry in the scene is static. Fortunately, we show that the cost of rebuilding the specialized acceleration structures can be kept low. In fact, the per-ray savings over traditional acceleration structures more than pays for the construction time of multiple structures when the scene is of moderate complexity. This means that, for any number of lights in the scene it is actually faster to rebuild a different acceleration structures for each one than it would have been to reuse a single acceleration structure over and over again. Because of this, it makes sense (is more efficient) to have a specific acceleration structure for *each* camera and light in the scene and one additional general purpose acceleration structure for reflections and refractions.

The specific specialized acceleration structure that we use is a projective grid, that is, a 2D or 3D grid spatial data structure in a space that has been transformed by a projective projection (Figure 1b). This data structure has obvious similarities to that used by the traditional Z buffer, hinting at why it provides such high performance for eye rays and shadow rays. The data structure is also similar to that used by the ZZ buffer [24], which was motivated by similar goals of achieving Z-buffer-like performance for broader classes of visual effects. However, the ZZ buffer considers its data structure to be an image-space buffer that is limited to specific visibility queries, whereas we consider our projective grid data structure to be a 3D ray tracing acceleration structure that can support arbitrary ray traversal, albeit with reduced efficiency for rays not originating near the center of projection. This distinction is subtle but critical, and leads to important differences as compared to the ZZ buffer in how our system handles more complex visibility queries such as those for soft shadows.

The techniques we present are integrated into a high performance implementation that is many times faster than state-of-the-art ray tracers for eye and hard shadow rays, and similar in performance to software Z buffer renderers for eye rays. For example, our system renders primary visibility for the fully dynamic courtyard scene at 1920x1200 at over 130 frames per second and hard shadows at over 30 frames per second, making ours by far the fastest ray-tracing visibility engine that we know of (for eye and hard-shadow rays). Our implementation also includes a number of algorithmic optimizations, some of them only valid for models that are closed (i.e. with each polygon exclusively front-facing or back-facing). In particular we use front-face culling (rather than back-face culling) for shadow rays to significantly improve performance.

Although the performance of our software implementation approximately matches that of software Z buffer renderers, it is not yet performance competitive with Z buffer graphics hardware. However, the trend in graphics hardware is towards increasingly programmable designs, which within a few years could implement all rendering algorithms in highly parallel software. Therefore, our results indicate that the potential exists for rendering systems based around a ray tracing framework to be performance competitive with rendering systems based around a Z buffer framework while also providing additional flexibility.

The ideas we present also show that the traditional clear-cut distinction between Z buffer algorithms and ray tracing algorithms is somewhat artificial, and that in fact there is a whole spectrum of algorithms lying between the two traditional approaches. The concept of a perspective acceleration structure provides a link between the two approaches.

## 2 Background

Each frame, a rendering system must find the intersection points between many rays and many polygons. The cost of testing each ray against each polygon is prohibitive, so such systems typically process (sort or hash) either the polygons or the rays or both in order to reduce the number of ray/polygon intersection tests that must be performed. This sorting/hashing may be either explicit or implicit. In a traditional ray tracer [32], the polygons are explicitly sorted or nearly-sorted, yielding an acceleration structure [4] such as a bounding volume hierarchy [23], a 3D grid, or a BSP-tree. Until recently, it was common to assume that the polygons comprising the scene did not move from frame to frame, which allowed the acceleration structure to be built once and reused for many frames. In such systems, the acceleration structure is optimized to minimize ray/triangle intersection cost based on the Surface Area Heuristic (SAH), which assumes that rays are equally likely to come from any direction [7, 18, 12]. This assumption is reasonable for an acceleration structure that will be reused over many frames.

### 2.1 Rebuild Each Frame

Over the past few years there has been a surge of interest in interactive ray tracing of scenes containing moving and deforming objects [29]. For such scenes, the acceleration structure must be updated or rebuilt every frame. This need has led to the development of a variety of techniques for rapid construction of both simple and high-quality acceleration structures. Interestingly, once the acceleration structure is being rebuilt every frame, it is no longer necessary for the SAH to assume that rays are equally likely to come from any direction. In fact, a rendering system knows quite a bit about the origin and direction of rays in any particular frame, especially the eye rays and shadow rays. The system may also have considerable local knowledge about ray directions for other kinds of rays. In this paper we show that by exploiting this information, it is possible to build specialized acceleration structures that lower the overall cost of finding the intersections between rays and polygons. We will describe the approach and our system, and follow that with a more detailed discussion of related work.

### 2.2 Surface Area Heuristic

When a rendering system is building an adaptive ray tracing acceleration structure such as a kd-tree or bounding volume hierarchy, there are many valid acceleration structures that can be built for a particular scene. For example, there are many possible locations for each splitting plane in a kd-tree. In order to achieve good ray tracing performance, the system must make choices during acceleration structure construction that are likely to lead to good performance during ray traversal. More formally, the system attempts to make split-plane choices that minimize an objective function representing estimated traversal cost. To date, no technique is known for efficiently finding the true global optimum for this function, so the standard practice is to use a greedy technique that locally minimizes an objective function at each node in the acceleration structure. The standard objective function that is locally minimized, called the Surface Area Metric [7, 18, 12], is the following:
$cost_{traversal} = c_{node} + \sum_{children} P_{child} * C_{child}$
where $c_{node}$ is a constant per node cost, $P_{child}$ is the probability of intersecting a child and $C_{child}$ is the cost of intersecting a child.

One of the terms in this objective function is the estimate of the probability of traversing a child node. Traditionally, this term is computed by assuming that there is a uniform directional distribution of rays hitting the node, in which case the probability is proportional to the surface area of the node. However, if the direction of the rays is known, then the probability is instead proportional to the surface area seen from that direction (i.e. the area after projecting onto a plane that is perpendicular to the ray direction) [11]. Figure 2 shows that when this information about ray directions is available, split planes are chosen differently than they are without this information, and the cost of visiting a node is significantly reduced. In particular, it is much more favorable to choose split planes that are parallel to the direction of the rays. This example shows that it is necessary to think differently about the construction of acceleration structures if the direction of the rays is known. A detailed discussion of the surface area heuristic with known direction is described in [11].

## 3 New Acceleration Structures

If a system uses information about ray directions to choose better split planes within a traditional acceleration structure such as an axis-aligned kd-tree or bounding volume hierarchy, we would observe an improvement to performance over the traditional approach [11]. However, an even bigger improvement is possible if we choose the right kind of acceleration structure, such that it is always possible to pick split planes that are parallel or nearly-parallel to the (known) ray directions.

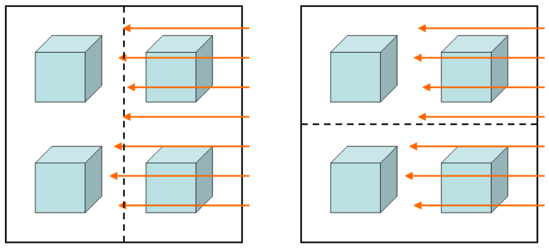

Figure 2: Considering ray direction when building an acceleration structure can significantly reduce the cost of traversing the structure. Consider two different locations for a split plane (dotted line). The vertical split illustrated on the left doesn't reduce the number of ray/object intersections at all. However, the horizontal split illustrated on the right reduces the number of intersections by 50%, even though these two splits are considered equally good by the traditional, direction-independent surface area metric.

Consider first the simplest case: eye rays. The rays share a common origin. Most traditional acceleration structures use axis-aligned split planes. Unfortunately, these planes will only be parallel to a few of the rays (Figure 1c).

We propose instead the use of an acceleration structure in which the split planes are axis-aligned in *perspective* space. Figure 1d illustrates this acceleration structure and shows that two out of the three splitting directions are always parallel to all of the rays.

The straightforward way to traverse this acceleration structure would be to do so in world space – as is typical in a ray tracer – in which case most of the planes would not be axis-aligned in the coordinate system used for traversal. This approach is possible and would still yield the benefits of having splitting planes that are parallel to the rays, but would lead to some of the same numerical robustness issues that appear for non-axis-aligned BSP acceleration structures.

Instead, we take advantage of the fact that under a projective transformation, lines remain lines. Thus, we can transform rays into the projective coordinate system and implement a standard traversal of these rays through the acceleration structure in the projective coordinate system. In this coordinate system, all of the splitting planes are axis aligned, yielding the usual robustness and efficiency advantages. As we will show in later sections of this paper, this acceleration structure has extremely good performance for rays with known directions such as eye rays and shadow rays.

At this point, this visibility algorithm has many similarities to a Z buffer, but several important differences remain:

- Although this acceleration structure is most efficient for rays originating at its center of projection – e.g. eye rays – we can traverse *any* ray through it. Note that to represent all regions of space it is necessary to use a cubemap-like arrangement of six of these acceleration structures. Construction of these six acceleration structures containing approximate one sixth of the geometry each should not be significantly more expensive than constructing one that contains all of the geometry.

- Ray origins and directions are explicitly stored, rather than implicitly represented via a formula.

- Ray/object intersection testing is usually performed in a 3D space. This 3D space can be either the 3D projective space or the original 3D world space. For example, ray/sphere or CSG intersections are more efficiently computed in world space.

- The acceleration structure retains a third dimension (in depth). This 3rd dimension is lacking in a traditional Z buffer although it partially re-appears in Z-buffer systems with occlusion culling capabilities [8]

- Although splitting planes are axis-aligned, they are not necessarily regularly spaced as they are for the pixels in a Z buffer.

- The overall processing order for rays and objects is ray order rather than object order. That is, the system makes an explicit pass over all objects at the start of the frame to build the acceleration structure.

If desired, it is possible to remove various combinations of these remaining differences. In the limit, one obtains an algorithm that is the same as the traditional Z buffer.

Once an acceleration structure is specialized for certain kinds of rays, there is also an opportunity to store additional information in the acceleration structure that is relevant to those rays and requires preprocessing of the geometry beyond just 3D sorting. For example, we can store a silhouette edge or the minimum distance between any triangle in a cell and a particular point in space that is the origin of the rays.

## 4 Ray Tracing with the Perspective Grid

Next we will address the question of how to apply the insights about acceleration structures just discussed to a ray tracer. First we'll focus on tracing a general (e.g. soft shadow) ray from a localized area (e.g. light source), then we'll discuss optimizations that can be made for rays that originate at a point.

To trace a general ray from an area light source, the first step is to build a perspective grid acceleration structure for that light. This requires transforming geometry into the perspective space. For polygonal geometry, this is done by transforming each vertex of the polygon into the perspective space. Non-polygonal geometry may be represented by a transformed bounding box. The build process in perspective space is nearly identical to a grid build in normal space except for this transform. The center of projection for a grid acceleration structure is chosen to lie on or near the light source. One difference of that should be noted is the "perspective singularity" which occurs at $z = 0$. The perspective transform has an asymptote at this location. The easiest way to avoid this (which we implement) is to clip all rays, geometry and bounding boxes to a near-plane.

Once we have constructed a perspective grid, tracing rays through it is very similar to tracing rays through a normal acceleration structure. The ray is first transformed into perspective space, and then traversed as it normally would in a grid acceleration structure. Packets may be supported by simply transforming all rays in a packet before traversing them. Intersections are performed in the normal manner, but if triangle interpolants are needed, they must be perspective corrected. Depth of field may be supported in the same way soft shadow rays are. Motion blur rays have common or near common origin and can be traversed in the same fashion.

For rays sharing a common origin some additional optimizations are possible. The center of projection for the perspective acceleration structure is located exactly at the origin point. This implies that as rays traverse the perspective grid acceleration structure they will step across Z planes, but never step across X or Y planes. As a result, the traversal algorithm can be simplified to step only in Z. Furthermore, if depth complexity is low, the Z dimension of the grid acceleration structure can be omitted completely, which further simplifies the traversal algorithm. In this case each ray traverses exactly one cell. For eye rays, the resulting technique turns out to be very similar to a sort-middle tiled Z buffer algorithm, but arrived at through a clear series of simplifications and optimizations from a fully general ray tracing algorithm.

In the next several sections of the paper, we present a high performance system that uses these techniques to trace eye rays, hard shadow rays, and soft shadow rays.

## 5 System Design

As in a standard Whitted ray tracer, the overall flow of control in our system is driven by the tracing of eye rays, which in turn trigger the tracing of shadow rays. Throughout this section, we will discuss how to use the perspective grid to achieve high performance for different sets of rays, using our system as an example.

### 5.1 Eye Rays

Eye rays are exceptionally well behaved: They share a common origin, their directions are evenly distributed and can be computed from a formula rather than stored, and these rays can be easily grouped in any desired way. Thus, the acceleration structure and traversal algorithm may be highly optimized for these conditions. With these optimizations, our algorithm for eye rays can be considered to be either a degenerate (no splits in Z) version of the perspective grid technique or a modified tiled Z buffer renderer of the sort-middle variety [19] with additional capabilities such as the ability to render non-polygonal geometry.

Our implementation of the acceleration structure for eye rays is a perspective grid with no splits in the $Z$ dimension. In the current implementation the grid uses a resolution such that each cell is approximately 100x100 pixels. This ensures that the color and depth values for ray hit points associated with a cell fit into L2 cache for our processor. For high-depth complexity scenes, the third dimension of the grid could be restored. The acceleration structure is constructed using the method discussed in the previous section. We do not pre-process scene objects for faster intersection in any way.

Our system processes one grid cell at a time, along with all of the geometry and eye rays that intersect that cell. Within each cell, intersection tests are performed as they would be in a Z-buffer rasterizer, and in particular are performed in unsorted object order rather than ray order. For each object, our system finds all rays that intersect the perspective aligned bounding box of that object, and intersects them with that object. This is computationally efficient because primary rays may be found/computed via a formula. Rays are processed in packets of 4, using the x86 4-wide SIMD instructions. A per-tile distance buffer is checked and conditionally updated for each intersection test. A floating-point color buffer is also updated with a color computed with a simple shading model (dot product of normal with light vector). These intersection and shading algorithms are very similar to the ones used by a tiled Z-buffer renderer.

After shadows/shading, simple tone mapping based on min/max intensity from the entire previous frame is performed for all rays in a grid cell, converting 32-bit float per component color down to 8-bit per component color before being written to system memory. As the Results section will show, the performance of this algorithm is much faster than traditional ray tracers, and comparable to that of software Z buffer renderers.

### 5.2 Hard shadow rays

After processing the geometry in a grid cell to find all intersection points, our system casts shadow rays for all hit points found in that cell. However, if the intersection point is on a back-facing polygon (with respect to the light), it is assumed to be in shadow without tracing the shadow ray.

Hard shadow rays are almost as well behaved as light rays. However, ray directions must be stored explicitly since they are not regularly spaced. Also, intersection testing is done in ray order since shadow rays are generated on the fly from eye rays. Thus, our system's processing of hard shadow rays is much more like traditional ray tracing than the processing of eye rays.

Our system uses one perspective grid per light. These grids also have no splits in the $Z$ dimension. Thus, as with eye rays, each hard shadow ray traverses exactly one grid cell. In the current implementation the hard shadow perspective grid is 200x200 cells, much finer than that used for eye rays. The perspective grid acceleration structure for each light is built at the start of the frame. Only back-facing objects (with respect to the light) are transformed and added to the grid. The system assumes that models are closed.

There are several advantages to culling the front facing triangles [30, 34]. At each grid cell, the system stores the distance between the light and the closest geometric primitive in the cell. This distance is more aggressive than it would be if back faces were culled. This form of hierarchical culling skips all intersection tests for up to 90% of the non-shadowed rays. Using front-face culling also has the advantage that rays cast off of the surface of a front face cannot instersect that surface, eliminating shadow acne [35].

When rendering a scene with many lights and lots of geometry, it is clear that the use of multiple acceleration structures will consume more memory than a single accleration structure would. In cases where memory consumption is a problem, it could be addressed in a variety of ways; including building the acceleration structures on-demand or by only maintaining one hard shadow acceleration structure at a time. This second approach requires either storing primary hit locations before casting any shadow rays or recasting primary visibility rays for each light (which is particularly inexpensive using the perspective grid).

As the results section will show, the performance of this algorithm is much faster than traditional ray tracers. Traditional Z buffer renderers cannot support this visibility query at all except using shadow mapping [33] which causes significant artifacts by approximating the query. The Irregular Z-Buffer [16, 3] can support this visibility query with high performance and without artifacts, but it uses an object-order system organization which integrates poorly with other ray tracing queries.

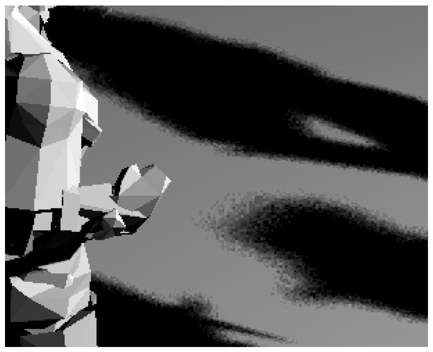
Figure 3: Soft shadows rendered by our system (Courtyard scene).

### 5.3 Soft shadow rays

Soft shadow rays (Figure 3) require a full 3D traversal of the perspective grid acceleration structure, and thus provide the best example of the fact that a perspective grid is a true 3D acceleration structure, capable of supporting traversal by any ray.

The acceleration structure is a 3D perspective grid (200 x 200 x 4). There are fewer splits in the third dimension as suggested by the theoretical analysis provided in the Surface Area Heuristic section. As before, each light has its own perspective grid acceleration structure. The acceleration structure is built at the start of the frame, ignoring fully front-facing triangles (from any point on the light).

As with hard shadow rays, the soft shadow rays are processed in batches of approximately 100x100 driven by the eye rays, but there are now 8 shadow rays per eye hit point. Rays are traversed in 4-wide packets for SIMD efficiency, with the packets consisting of four shadow rays from a single eye-ray hit point. This guarantees that shadow packets are always as coherent as the area light allows. The traverser is a slice-based ray packet traverser for the grid acceleration structure, following the technique described in [28].

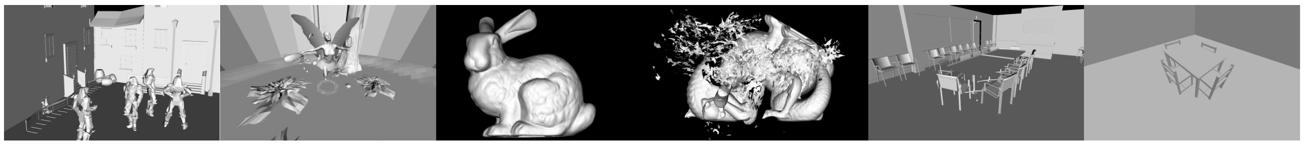

Figure 4: Scenes and viewopints used to gather results. From left to right: courtyard, fairyforest020, bunny69, dragon-bunny5, conference and erw6.

| End to End runtime Results for our System | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Scene | Polys | Primary | | | Hard Shadows | | | Soft Shadows | | |
| | | FPS | build | Mray/s | FPS | build | Mray/s | FPS | build | Mray/s |
| Courtyard | 31k | 30 | 5% | 68 | 8 | 10% | 36 | 0.26 | 0% | 5.3 |
| FairyForest020 | 174k | 17 | 17% | 39 | 6 | 33% | 27 | 0.11 | 1% | 2.3 |
| Bunny-69k | 69k | 44 | 18% | 100 | 11 | 27% | 50 | 0.41 | 1% | 8.4 |
| Bunny-16k | 16k | 81 | 7% | 185 | 24 | 21% | 109 | 0.57 | 0% | 11.7 |
| Bunny-4k | 4k | 120 | 3% | 274 | 31 | 13% | 141 | 0.65 | 0% | 13.3 |
| Bunny-1k | 1k | 154 | 1% | 351 | 37 | 10% | 169 | 0.72 | 0% | 14.8 |
| Dragon-Bunny | 252k | 17 | 25% | 39 | 4 | 50% | 18 | 0.30 | 4% | 6.2 |
| Conference | 282k | 16 | 33% | 36 | 5 | 45% | 23 | 0.19 | 1% | 3.9 |
| ERW6 | 1k | 56 | 1% | 128 | 15 | 3% | 68 | 0.19 | 0% | 3.9 |
| Subset of the 8-Cores in Parallel Performance Table (Secondary Result) | | | | | | | | | | |
| Courtyard | 31k | 150 | 27% | 342 | 34 | 41% | 155 | 2.0 | 2% | 41 |
| FairyForest020 | 174k | 55 | 56% | 125 | 14 | 78% | 64 | 0.51 | 4% | 15 |
| Bunny-16k | 16k | 339 | 31% | 773 | 71 | 62% | 324 | 4.2 | 3% | 86 |

Table 1: Performance of our system for various scenes and for various quality settings, all at 1920x1200 resolution. The results include the time for per-frame build of the acceleration structure. The top nine rows use one core of a 2.66 GHz Xeon X5355, 1333MHz FSB. Hard shadows use one eye ray and one hard shadow ray per pixel. Soft shadows use one eye ray and eight Monte Carlo soft shadow rays per pixel. Hard shadows use a 200x200x1 perspective grid, except FairyForest (800x800x1) and Conference (600x600x1). Soft shadows use a 200x200x4 perspective grid. The bottom three rows show parallel performance on eight of the same cores. All phases of the system except acceleration-structure build parallelize well, but since build is not yet parallelized it becomes the bottleneck in the parallel scenario.

As the results section will show, traversing the perspective grid is substantially cheaper than traversing a standard 3D grid. As expected, performance is best for small lights, and degrades as the light becomes larger, because the shadow rays are less well aligned to the primary projective axis.

## 6 Results

This section presents results gathered from the system we have built to implement the algorithms described and compares these results to alternative approaches. First, we will present performance of our system on various scenes for eye rays, hard shadows, and soft shadows. Second, we will compare our system's performance to that of other interactive ray-tracing systems and software Z-buffer renderers. Third, we present operation counts showing that soft-shadow rays traversed through a perspective grid require fewer traversal steps than the same rays traversed through a regular grid.

We demonstrate performance many times faster than other dynamic ray-tracing systems across a range of scenes. For several scenes we also demonstrate performance of greater than 100 million rays/second for primary visibility on a single core. Additionally, we achieve real time performance (over 30fps) at 1920x1200 for hard shadows on a game-like scene (courtyard) using eight cores. Finally, our eye-ray performance approximately matches that of modern software Z-buffer renderers. To the best of our knowledge, no other real-time ray-tracing system has these capabilities.

### 6.1 Overall performance

Table 1 shows measured system performance for a variety of models on a single CPU core. Our system is focused on measuring visibility performance and so we exclude expensive local shading operations that would make the results more difficult to interpret. In particular, we do not implement texture mapping, since modern CPUs lack the memory bandwidth and specialized hardware needed for high performance texture mapping. We do implement per-pixel diffuse shading of interpolated artificial colors to demonstrate that our technique supports interpolated vertex parameters efficiently. The system can interpolate normals similarly, but we usually run with this capability disabled because many of our scenes lack correct per-vertex normals. We report performance for regular eye rays, hard shadows with regular eye rays, and soft shadows with regular eye rays.

Several conclusions can be drawn from these results.

First, hard shadow rays are not quite as fast as eye rays by the metric of ray-segments / sec but still perform very well compared to other ray tracing systems.

Second, soft shadow rays perform substantially slower than hard shadow rays by the metric of Ray segments / sec. There are a couple of reasons for this: Most soft shadow rays are not perfectly parallel to the Z axis, and so they must make traversal steps in X and Y as well as Z. The mere fact that the traversal algorithm supports stepping in X, Y, and Z makes it significantly slower than the special-case algorithm used for hard shadow rays, even if it is used to trace rays parallel to the Z axis which do not step. Additionally, the perspective grid suffers from the ailment of all uniform acceleration structures known as the "teapot in a stadium" problem. A companion paper to this one addresses soft shadows using adaptive perspective space acceleration structures [13].

Table 1 also reports the fraction of the frame time that was spent on building the acceleration structure(s), and these results show some interesting trends. First, the acceleration structure build is somewhat more costly for hard shadows than for primary rays, because as discussed earlier the acceleration structure used for shadow rays is more complex than that for primary rays. Second, for models of 100k+ polygons, build cost is a significant fraction (0.10-0.50) of the total frame time for eye rays, and hard shadows, but not for soft shadows. Since grid acceleration structures are typically less efficient for traversal than adaptive data structures such as kd-trees, these results suggest that soft shadows would benefit from the additional computation required to build an adaptive acceleration structure such as a perspective kd-tree or perspective bounding-volume hierarchy. This is explored in our companion paper [13]

To achieve real-time frame rates on the models used in typical interactive applications, the techniques used in this paper would need to be parallelized for multi-core architectures. We have already parallelized the traversal algorithm, and as expected we observe good scaling (around 90% of linear with 8 cores). Parallelizing the construction of the perspective-grid acceleration structures is still future work, but we believe that a combination of ideas from traditional Z-buffer parallelization [19], regular-grid parallelization [15] and on-demand parallel build of acceleration structures from hierarchy [14] can be successfully applied to this problem. We provide results for some scenes using 8 cores.

### 6.2 Comparison to other ray tracing systems

In this section we compare the performance of our system against the performance of other recent high-performance ray tracing systems for eye rays and hard shadow rays. Table 2 compares the performance of our system against others that support dynamic and semi-dynamic scenes. Table 3 provides a similar comparison for static scenes (i.e. excluding acceleration structure build time for comparison systems but not ours). We make a good-faith effort to make fair comparisons, but for a variety of reasons – especially incomplete data in previous publications and finite space in this one – it is difficult to make such comparisons as precise and exhaustive as one would like. In both of these tables, we adjust previously published results measured on Pentium 4 3.2 GHz systems and Opteron 2.6 GHz systems upward by a conservative factor of 1.5x to estimate their equivalent performance on the 2.66 GHz Core2 system that we use. This adjustment allows each system to be evaluated on the platform for which it was optimized. The viewpoints used for gathering our results have been visually matched to be as close as possible to the ones used in the publications we compare to, and we use the same resolution as the previous results, 1024x1024.

We first compare to Wald's grid system which uses an ordinary grid acceleration structure [28], and thus supports arbitrary dynamic scenes just like our system. Our system is faster than the grid system by a factor of 2.2x for eye+hard shadow rays on the one model (Fairy Forest) for which we can compare. For eye rays, our system is even faster, by over 4.6x in all cases for which data is available. More precisely, when comparing our system *including* build time against the original grid paper *excluding* build time, our system is 4.6x to 5.0x faster. With build time excluded for both systems, our system is 5.1x to 7.9x faster.

We also compare to Wald's bounding volume hierarchy system [27], which can be considered to be semi-dynamic because the topology of its optimized bounding volume hierarchy is precomputed in a slow preprocessing step. Only the bounds in the hierarchy are updated each frame. Our system is always faster than the BVH system for eye rays, with conservative speedup ranging from 1.37x to 2.83x. Our system is faster than the BVH system in most cases for eye+shadow rays, with speedup ranging from 0.94x (i.e slight slowdown) to 2.36x. The largest speedups are seen with the Fairy Forest model, which is the only truly animated model and thus is the most reasonable point of comparison.

Finally we compare to MLRTA [22], a high performance ray tracer for static scenes which uses a highly-optimized pre-build acceleration structure. With our system rebuilding its lightweight acceleration structure every frame, it matches or outperforms MLRTA. This result demonstrates that our dynamic ray-tracing system can be as fast as the fastest static ray-tracing systems.

Unfortunately, we cannot compare to any other high performance rendering systems on the Courtyard scene which we believe to be the best proxy for modern game scenes. Our system performs especially well on this scene for a variety of reasons, including the scene's relatively uniform polygon size.

| | | Eye Rays | | | Eye + Hard Shadow Rays | | |
|---|---|---|---|---|---|---|---|
| Scene | Polys | Us | grid | BVH | Us | grid | BVH |
| ERW6 | 1K | 106 | – | ∼64 | 36 | – | ∼23 |
| Fairy | 174K | 26 | – | ∼9.2 | 7.6 | ∼3.4 | ∼3.2 |
| Conf | 282K | 22 | – | ∼16 | 6.8 | – | ∼7.2 |

Table 2: Comparison of dynamic scene performance, all in frames/sec at 1024x1024 resolution. Our system is running on one core of a 2.66 GHz Xeon 5355; Other results are taken from recent publications with different hardware, but adjusted in this table to estimate performance on a single 2.66 GHz Xeon 5355. Notes: (1) Adjustment for processor differences is an estimate; see text. (2) The BVH algorithm is restricted to certain types of dynamic scenes because it computes its acceleration structure topology off-line (taking an adjusted 2.16 sec for Fairy) and only updates the bounds each frame. (3) The BVH render times include basic texture mapping; others do not.

| | | Eye Rays | | | |
|---|---|---|---|---|---|
| Scene | Polys | Us | MLRTA | grid | BVH |
| ERW6 | 1K | 106 | ∼ 76 | ∼ 21 | ∼ 49 |
| Fairy | 174K | 26 | – | – | ∼ 12 |
| Conf | 282K | 22 | ∼ 23 | ∼ 4.8 | ∼ 14 |

Table 3: Comparison of static scene performance, all in frames/sec at 1024x1024 resolution. Results for our system include the time for acceleration structure build (since it is view dependent), while results for other systems exclude build time. Our system is running on one core of a 2.66 GHz Xeon 5355. Other results are taken from recent publications with different hardware, but adjusted to estimate performance on the 2.66 GHz Xeon 5355.

| | Our System | Pixomatic (x1.5) |
|---|---|---|
| Triangle Rate | 8.32 | ∼7.92 |
| Transform & Project Rate | 32.1 | ∼35.0 |

Table 4: Comparison of our system's performance for eye rays vs. published results for Pixomatic, a high performance software Z-buffer renderer. Notes: (1) Pixomatic is performing texture mapping but our system is not; see text for details. (2) Pixomatic results are adjusted upward to account for hardware differences.

### 6.3 Comparison to Z-Buffer systems

When our system is tracing just eye rays its functionality and algorithms for visibility are similar to those of a tiled Z-buffer renderer. Thus, it is appropriate to compare our system's performance to an optimized software Z-buffer renderer. In Table 4, we compare to Pixomatic 2.0 [1], which is sold commercially by RAD software for use as a fallback renderer in PCs lacking fast graphics hardware. RAD software reports performance results for their system on a 3.3 GHz Pentium-4 [20], and we adjust these results upward

|  | Regular 3D Grid | Perspective 3D Grid |
|---|---|---|
| Grid Size | 54x54x54 | 200x200x4 |
| Traversal Steps/Ray | 52.0 | 12.7 |

Table 5: When tracing soft shadow rays, the perspective grid requires fewer grid traversal steps than a regular 3D grid. These results are measured using our system with the Courtyard scene at 1920x1200 resolution.

by a factor of 1.5x to estimate performance on our 2.66 GHz Xeon X5355.

We measure performance using the 69.5K bunny model under conditions that are as similar as possible as the ones used by RAD given the different natures of the two systems. It is important to note that Pixomatic is configured to perform texture mapping, while our system is not. However the effects of this difference should be small because the triangle rate in both systems is measured with only 5% of the window covered and the transform and project rate is measured with the model off-screen. Thus, although these results must be interpreted with caution, we believe they do show that the performance of our system for eye rays is comparable to that of a high performance software Z-buffer renderer.

### 6.4 Perspective Grid vs. Regular Grid

Although execution time is the ultimate metric for most real-time rendering algorithms, more specific measurements can provide deeper algorithmic insights. To assess the effectiveness of the perspective grid acceleration structure as compared to an ordinary grid, in Table 5 we compare the number of traversal steps needed used when rendering soft shadows for the Courtyard scene at 1920x1200. The total number of cells in each grid is essentially the same but the results show that the perspective 3D grid requires less than 1/3 the number of traversal steps required by the regular 3D grid.

## 7 Previous Work

There are a variety of earlier systems and techniques that have some similarity to the ideas we have presented. We discuss this previous work here and compare it to our work.

### 7.1 ZZ-Buffer

The ZZ-buffer [24] is motivated by a similar goal of achieving ray tracing quality with Z-buffer-like performance for effects such as hard shadows and soft shadows. In our terminology the ZZ-buffer is a perspective coordinate acceleration structure, although its designers describe it as an image space buffer and only briefly mention its connection to ray tracing acceleration structures. The ZZ-buffer's X and Y dimensions are organized as a grid, just like the data structure we have presented. The Z dimension is different from ours; it is a sorted list of object bounding ranges. For eye rays and shadow rays, the ZZ-buffer is used much like our perspective-grid data structure. However, the ZZ-buffer handles soft shadow rays and depth of field rays in a way that is fundamentally different from our system. In our system, the perspective grid is treated as a true acceleration structure that can be used to trace arbitrary rays using grid traversal algorithms. In contrast, the ZZ-buffer must be constructed differently to support off-axis rays, with the maximum off-axis distance baked into the acceleration structure. The ZZ-buffer supports off-axis rays by duplicating object pointers is nearby cells in the acceleration structure (one can think of this as blurring the objects within the data structure in the X and Y dimensions), and then traversing rays through a single X/Y cell as if they were on-axis rays. If the soft-shadow penumbra size is large or there is significant depth of field blur, our approach should be significantly more efficient than the ZZ buffer for tracing off-axis rays.

The ZZ-buffer supports anti-aliasing using a super-sampling based approach. It also supports partial transparency, which is not currently supported by our system but could easily be; and CSG operations [25], which we have not examined.

### 7.2 Tiled Z-buffer systems

When tracing eye rays, our system's algorithm is similar in practice to a tile-based Z-buffer renderer (also known as bucketing, chunking or zone-based renderers), see e.g. [6]. Such systems are sort-middle algorithms in Molnar et al's taxonomy of parallel Z buffer algorithms [19], and they sort (or bin) all polygons into relatively coarse tiles before performing final Z-buffer visibility testing separately in each tile.

### 7.3 Ray tracing acceleration structures

For static scenes, ray tracing acceleration structures have been studied for a long time (e.g. [23, 4]). Recently there has been a surge of interest in techniques for rapidly building acceleration structures for dynamic scenes; see [29] for a recent review of this work. Since our acceleration structure is a perspective grid, the most closely related work is Wald's work on rapid construction of traditional grid acceleration structures [28].

The idea of using special data structures to assist with tracing eye and shadow rays has been explored before, mostly with strictly 2D data structures. Weghorst et al's item buffer [31] is generated using a conventional Z buffer algorithm and used instead of a general-purpose acceleration structure when tracing eye rays. The Vista buffer [10] and ZF-buffer [17] implement variations of this approach for eye rays and for reflection rays from a plane. Haines and Greenberg's light buffer [9] uses a similar approach for hard shadow rays. The ZZ-buffer, discussed earlier, can also be considered to be a specialized ray tracing acceleration structure, and has partial support for a third dimension in the structure. Finally, Reshetov describes an approach that creates a transient frustum (i.e. very simple acceleration structure) every time a packet of rays visits a leaf node [21]. As with our approach, this transient frustum makes use of information about the rays that are being traced.

Agrawala et al [2] render soft shadows by tracing shadow rays through one or more shadow maps. These shadow maps can be considered to be perspective-space acceleration structures, although these structures are different from ours in that the geometry is discretely sampled (i.e. converted into an image-based representation) before being inserted into these acceleration structures. Layered depth images [26] share this same characteristic.

Arvo and Kirk lazily create a hierarchical 5D acceleration structure. Each subtree of this acceleration structure can be considered to be a ray-specialized acceleration structure, since each subtree holds geometry that is potentially visible from rays whose origins are within in a particular volume and whose directions are within a particular solid angle. Arvo and Kirk also discuss further specialization of the 5D data structure for rays sharing a common origin, yielding a hierarchical ray-specialized 2D acceleration structure.

## 8 Discussion and Conclusion

In this paper we have shown that on modern hardware it is useful to build ray tracing acceleration structures that are specialized for rays with specific origins and/or directions, and we have provided a theoretical explanation for these results using the surface area heuristic. In particular, we have shown that a perspective grid acceleration structure can provide very high performance for rays originating at or near a common point. In some cases – such as when all rays are known to share a common origin – the traversal algorithms themselves can be simplified to further improve performance. These results are helpful in understanding the continuum between the traditional Z-buffer algorithm and the traditional ray tracing algorithm. Additionally, we have demonstrated the highest hard-shadow performance (per core) that we know of. This high performance, combined with the relatively simplicity of the grid structure and its traversal, make this approach worth considering for hard shadow rendering in future real-time graphics systems.

Future work includes: (1) studying the performance of these acceleration structures in a fully-featured rendering system; (2) parallelizing perspective grid build; and (3) combining the ideas presented in this paper with recent advances in lazy construction of acceleration structures from scene hierarchies [14], so as to efficiently support scenes with high depth complexity. In a companion paper [13], we apply the perspective transform to an adaptive acceleration structure (the kd-tree) in order to provide better performance for soft shadows, which have a high ratio of traversal cost to build cost.

## ACKNOWLEDGEMENTS

We thank Peter Djeu for helping us with tools and models and Ingo Wald and the anonymous reviewers for providing feedback on the paper. This work was funded primarily by an Intel Fellowship and Intel Research grant, for which we thank Jim Hurley and Stephen Junkins. Additional funding was provided by an NSF CAREER grant (CCF-0546236).

## REFERENCES

[1] M. Abrash. Optimizing Pixomatic for x86 processsors: Part I. *Dr. Dobbs Journal*, Aug. 2004.

[2] M. Agrawala, R. Ramamoorthi, A. Heirich, and L. Moll. Efficient image-based methods for rendering soft shadows. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 375–384, July 2000.

[3] T. Aila and S. Laine. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 161–166. Eurographics Association, 2004.

[4] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A. S. Glassner, editor, *An Introduction to Ray Tracing*. Academic Press, San Diego, CA, 1989.

[5] R. L. Cook, L. Carpenter, and E. Catmull. The REYES image rendering architecture. *Computer Graphics (Proc. of SIGGRAPH 87)*, 21(4):95–102, July 1987.

[6] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, and L. Israel. A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics (Proc. of SIGGRAPH '89)*, 23(3):79–88, 1989.

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

[8] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 231–238. ACM, 1993.

[9] E. Haines and D. P. Greenberg. The light buffer: A shadow-testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, Sept 1986.

[10] A. Hashimoto, T. Akimoto, K. Mase, and Y. Suenaga. Vista raytracing: High speed ray tracing using perspective projection image. In *New Advances in Computer Graphics (Proc. of CG International '89*, pages 549–562. Springer-Verlag, 1989.

[11] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University, Nov. 2000.

[12] V. Havran and J. Bittner. On improving KD trees for ray shooting. In *Proceedings of WSCG*, pages 209–216, 2002.

[13] W. Hunt and W. R. Mark. Adaptive acceleration structures in perspective space. In *2008 IEEE Symposium on Interactive Ray Tracing*. IEEE, Aug. 2008.

[14] W. Hunt, W. R. Mark, and D. Fussell. Fast and lazy build of acceleration structures from scene hierarchies. In *2007 IEEE Symposium on Interactive Ray Tracing*, pages 47–54. IEEE, Sept. 2007.

[15] T. Ize, I. Wald, C. Robertson, and S. G. Parker. An evaluation of parallel grid construction for ray tracing dynamic scenes. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 47–55, 2006.

[16] G. S. Johnson, J. Lee, C. A. Burns, and W. R. Mark. The irregular Z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics*, 24(4):1462–1482, 2005.

[17] S. Kim, S. Kim, and K.-H. Yoon. A study on the ray-tracing acceleration technique based on the ZF-buffer algorithm. In *Proc. IEEE International Conference on Information Visualization*, July 2000.

[18] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *Visual Computer*, 6(6):153–65, 1990.

[19] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[20] RAD Game Tools. Pixomatic SDK Features. http://www.radgametools.com/pixofeat.htm, Jan 18, 2008.

[21] A. Reshetov. Faster ray packets - triangle intersection through vertex culling. In *2007 IEEE Symposium on Interactive Ray Tracing*, pages 105–112. IEEE, Sept. 2007.

[22] A. Reshetov, A. Soupikov, and J. Hurley. Multi-level ray tracing algorithm. In *ACM Transactions on Graphics (Proc. SIGGRAPH 2005)*, pages 1176–1185. ACM, 2005.

[23] S. Rubin and T. Whitted. A 3D representation for fast rendering of complex scenes. In *Proceedings of SIGGRAPH*, pages 110–116, 1980.

[24] D. Salesin and J. Stolfi. The ZZ-buffer: A simple and efficient rendering algorithm with reliable antialiasing. In *Proceedings of the PIXIM '89 Conference*, pages 451–66, Hermes Editions, Paris, France, 1989.

[25] D. Salesin and J. Stolfi. Rendering CSG models with a ZZ-buffer. *Computer Graphics (Proc. of SIGGRAPH '90)*, 24(4):67–76, 1990.

[26] J. Shade, S. Gortler, L. wei He, and R. Szeliski. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, New York, NY, USA, 1998. ACM.

[27] I. Wald, S. Boulos, and P. Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1):1–18, 2007.

[28] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker. Ray tracing animated scenes using coherent grid traversal. *ACM Transactions on Graphics*, 25(3):485–493, 2006. (Proceedings of ACM SIGGRAPH).

[29] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, and P. Shirley. State of the art in ray tracing animated scenes. In *Eurographics 2007 State of the Art Reports*. Eurographics Association, 2007.

[30] Y. Wang and S. Molnar. Second-depth shadow mapping. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1994.

[31] H. Weghorst, G. Hooper, and D. P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, 1984.

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

[33] L. Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)*, pages 270–274, Aug. 1978.

[34] A. Woo. The shadow depth map revisited. In *Graphics Gems III*, pages 338–342. Academic Press Professional, Inc., San Diego, CA, USA, 1992.

[35] A. Woo, A. Pearce, and M. Ouellette. It's really not a rendering bug, you see... *IEEE Computer Graphics and Applications*, 16(4):21–25, Sept. 1996.