Adaptive Acceleration Structures in Perspective Space

Warren Hunt[*]
University of Texas at Austin

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

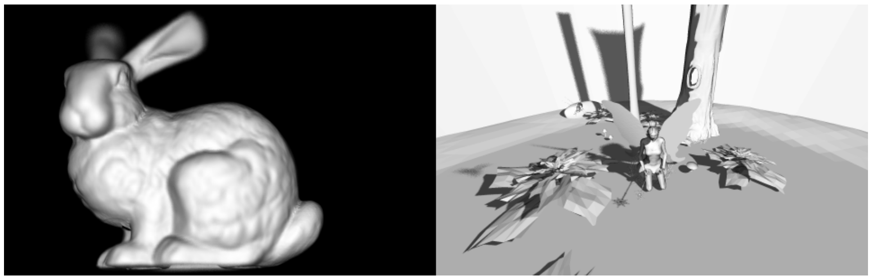
Figure 1: The Stanford Bunny rendered with depth of field and Fairy Forest rendered with soft-shadows both using a perspective kd-tree.

Abstract

Traversal efficiency of ray tracing acceleration structures can be improved by specializing them, each frame, for the rays that are traced in that frame. A companion paper to this one demonstrates that extremely high traversal performance for eye and hard shadow rays can be obtained by transforming rays and geometry with a perspective transform, then using a grid acceleration structure in the perspective space. However, the performance of this perspective grid acceleration structure suffers for off-axis rays such as those used for soft shadows or depth of field. In this paper we address this shortcoming by exploring the use of the perspective transform with adaptive acceleration structures such as kd-trees. The problem of choosing optimal split planes for an acceleration structure is different in perspective space than it is in world space, so we introduce a new cost metric for estimating traversal cost of acceleration structures in perspective space. This metric is related to the traditional surface area metric but defined in perspective space. We implement a ray-tracing system with both traditional and perspective acceleration structures and demonstrate significant performance improvements using the perspective space structures even when build time is included. Additionally we evaluate the effectiveness of our new cost metric compared to traditional cost metrics. A key insight demonstrated by these results is that end-to-end rendering performance can be improved by building a *different* specialized acceleration structure for each light and camera in the scene instead of using a single non-specialized acceleration structure for all rays.

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

1 Introduction

Ongoing advances in graphics hardware will soon provide enough computing power to make ray-tracing a viable rendering solution for interactive and real-time applications. However, algorithmic advances will also be required to make ray-tracing competitive in performance with alternative approaches.

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

A companion paper to this one introduces the perspective transform for ray-tracing acceleration structures and introduces a new acceleration structure, the perspective transformed uniform grid [6]. This transform aligns rays closely to the spatial axes and provides high traversal performance for eye and shadow rays. Although this acceleration structure is extremely fast to build, its traversal performance suffers significantly when used to trace off-axis rays, such as those required for soft shadows and depth-of-field effects. The goal of this paper is to extend the performance benefits of perspective transformed acceleration structures to these off-axis rays.

Our experimentation has shown that the performance degradation for off-axis rays is primarily due to the well-known inability of a uniform acceleration structure to adapt to the scene geometry. Adaptive acceleration structures such as bsp-trees and BVHs are known to address these issues [11]. In this paper, we extend the idea of tracing rays in perspective transformed space to adaptive acceleration structures, and in particular to kd-trees. To support this change, we will define a new metric for estimating traversal cost during acceleration-structure build. This new cost metric is designed for perspective space acceleration structures and incorporates more accurate assumptions about the origins and directions of rays. We provide results comparing the execution time and algorithmic operation counts of perspective kd-trees with those of traditional world-space kd-trees. These results show that the perspective kd-tree performs better than a perspective grid or a regular kd-tree for soft shadow and depth-of-field eye rays. For shadow and depth of field images, the improvement is a 24%-41% reduction in total time to image as compared to a regular kd-tree built using a scan-based fast build.

Although adaptive acceleration structures are more costly to build than uniform acceleration structures, this additional cost can be justified when tracing large numbers of rays, as is the case for soft shadow and depth of field effects. For example, using modern build techniques the Razor [2] and Manta [1] ray tracers use a very small fraction of overall runtime building acceleration structures. Offline ray-tracers typically use even a smaller fraction of time building acceleration structures. This implies that more complicated but more efficient (from a traversal point of view) acceleration structures could provide overall performance gains. A key insight demonstrated by this paper is that it can be adventageous to build an acceleration structure for each light and camera, every frame, even for complicated adaptive acceleration structures such as surface area heuristic based kd-trees.

2 Background: The SAM

Adaptive acceleration structures are built top-down using a recursive partitioning scheme [11]. In the case of kd-trees a (parent) voxel is partitioned using a split plane into two (child) voxels which are then left as leaves or recursively subdivided. The recursion terminates when a heuristic determines that further partitioning will no longer provide any benefit. In the case of a BVH, objects in a (parent) bounding box are partitioned into two collections which are then bounded by (child) bounding boxes. This process terminates when a box only contains one object. We will generically refer to a kd-tree voxel or a bounding box in a BVH as a *node*.

The effectiveness of an adaptive acceleration structure hinges greatly upon the quality of the heuristic used to choose partitions. Simple heuristics or poor cost metrics that choose bad partitions can produce acceleration structures that provide significantly poorer performance for ray-tracing. A classic example of a simple partition heuristic is the spatial mean split heuristic, which always partitions a kd-tree voxel in half. This heuristic produces usable acceleration structures quickly, but they are often several times less efficient than more carefully constructed structures. Given the classic motivation that acceleration structure build was a pre-processing step and essentially free, more advanced heuristics have been developed to improve ray-tracing performance.

The most common cost metric for building high quality acceleration structures is known as the surface area metric (or SAM) [3, 9, 5]. The SAM estimates the expected cost of traversing a ray through an acceleration structure or sub-structure. This metric can be used to evaluate the potential effectiveness of many different partitions in order to choose the best one. The process of building acceleration structures greedily using the SAM is known as using the surface area heuristic (or SAH). This heuristic produces acceleration structures that are significantly more efficient than simple heuristics such as the median spatial split heuristic [11]. Additionally, the SAM provides an automatic way to terminate the recursion during the build process. Build can be terminated when the cost (according to the SAM) of traversing a partitioned node is greater than the cost of traversing the non-partitioned node.

The SAM estimates the cost of traversing a partition by estimating the cost of each child in the partition and then scaling those costs by the conditional probability that a ray strikes each child. The cost of the children is estimated simply by the number of objects that fall into that partition. The conditional probability of intersecting a child is estimated by assuming a uniform distribution of ray-directions and computing the ratio of child surface area to parent surface area. Neither of these assumptions usually very accurate but the heuristic is simple to evaluate and works reasonably well in most cases. For reference, we provide the metric here.

$$cost_{traversal} = c_{node} + \sum_{children} P_{child} * C_{child} \tag{1}$$

- 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.

The assumption of uniform ray direction is, in some cases, a reasonable assumption. In the past, acceleration structures build was an offline process. Acceleration structures were built once and then potentially re-used for many different camera positions. This scenario provided a wide range of potential rays to be traced in a given acceleration structure. More accurate assumptions however, would provide a higher quality structure. Havran studied this problem in detail, providing many modifications to the standard SAM in order to address these issues [4]. However, at the time, per-frame rebuild was considered to be too slow to be practical and many of the proposed improvements have not been used for general purpose ray-tracing.

Recently, in order to support dynamic scenes, researchers have successfully made acceleration structure build an online process [16, 8, 7, 12]. Dynamic ray-tracing systems re-build or refit an acceleration structure (or in some cases many structures) each frame [17]. This per-frame rebuild allows for restrictions to be placed on the acceleration structure such as a known camera or light location, and we make use of this capability in this paper.

3 Adaptive Perspective Space Acceleration Structures

With these motivations in mind, we propose using one adaptive perspective space acceleration structure for each camera or light in the scene. Because the perspective grid works so well for eye and hard-shadow rays, we will focus on depth of field and soft-shadow rays here. The goal (which we demonstrate to be achievable) is for the improvement in traversal efficiency to outweigh the extra cost of building the extra acceleration structures.

As explained in our companion paper [6], building and traversing acceleration structures in perspective space is no more complicated than building and traversing acceleration structures in regular space. The perspective transform has the important feature that it maps lines to lines, thus mapping rays to rays and triangles to triangles. Because of theses properties, an acceleration structure built in perspective space can be traversed by rays in perspective space using all of the same algorithms that one would use in regular space. Even ray-triangle intersection can be performed in perspective space, although barycentric coordinates must be corrected prior to shading.

In this paper, we use a kd-tree in perspective space as our adaptive perspective space acceleration structure. In order to construct a perspective space kd-tree, we first transform the scene geometry into perspective space using the perspective transform:

$$x' = x/z \tag{2}$$
$$y' = y/z \tag{3}$$
$$z' = -1/z \tag{4}$$

In this case we use $z' = -1/z$ instead of $z' = 1/z$ because it preserves ordering along the z axis and more importantly, because it maintains the handedness of the coordinate system. Once the geometry is in perspective space, an acceleration structure may be constructed using traditional means. In our case we construct a kd-tree but the perspective transform may be used with any kind of acceleration structure.

In order to traverse a ray through the perspective space acceleration structure, the ray must first be transformed into perspective space using the equations above. Once it has been properly transformed, the ray may be traversed using any traditional means. This includes the use of packets [18], frustum culling [14, 15], vertex culling [13], etc.

In short, using the perspective transform to build specialized acceleration structures is conceptually no more complicated than using an object space transform for object-local acceleration structures. Geometry must be transformed properly before the structure is built and rays must be transformed properly before being traversed. In the case of using perspective space, the benefit comes from aligning the acceleration structure with the rays rather than with the geometry.

One additional benefit of using specialized per camera/light/frame acceleration structures is the ability to cull back-facing polygons before inserting them into the acceleration structure. Given an area light or a camera, it is easy to determine which faces are back-facing (or front-facing in the case of shadows [19]) to *all* of the rays from that light or camera. (Note: With an area light or camera, this is a smaller set than the faces that are back-facing to a single point.) Specifically, we cull all faces for which a sphere bounding the light/camera aperture lies completely behind the plane of the face. These faces will not be intersected by rays and may be completely ignored by the build process. This reduces build times in our results by approximately 50%.

4 The Perspective Singularity

Although the perspective transform has many useful properties such as mapping lines to lines, it has one challenging property that must be dealt with. The perspective transform divides by z and thus has a singularity at $z = 0$. This problem is well known in raster graphics systems, and we use one of the standard solutions: a near-clip plane. As will be evident from our build time results, this clipping process is not expensive.

One should note that because of this singularity, perspective space structures can only deal with half-spaces. To use perspective space structures for lights or cameras with a viewing angle greater than π radians, it would be necessary to use multiple perspective space structures. A solution to this problem would be to use six back to back structures. In raster graphics, this is known as cube mapping [10]. In the case of perspective space acceleration structures for ray-tracing, this increase in the number of structures should not affect build performance significantly because each structure, on average, will contain only one sixth of the total geometry.

5 The PSAM

Although the traditional SAM is commonly used to build acceleration structures in "regular" space, several factors make it less appropriate for use in the construction of perspective space adaptive acceleration structures.

The assumption of uniform incoming ray direction made by the SAM allows the probability of intersecting a child node to be computed as a ratio of node surface areas. These node surfaces are usually axis aligned boxes, whose areas are inexpensive to compute. To use this metric in perspective space, the node faces must be transformed back to regular space before computing their surfaces areas. These transformed faces are in general not axis aligned and their areas are thus more expensive to compute. Alternatively, if we changed the assumption to be: "incoming ray directions are uniformly distributed in *perspective* space", then we could use surface areas computed in perspective space from axis-aligned rectangles. However, the assumption of uniform incoming ray direction in perspective spaces is likely to be even less accurate than the assumption of uniform incoming ray direction in regular space. Either way, the assumption of uniform incoming ray distribution in any space is unreasonable when the acceleration structure is used for just one frame with one light or camera.

We opt for a first principles re-derivation of a heuristic for perspective space acceleration structures. First, we abandon the assumption of a directionally uniform incoming ray distribution in favor of a distribution more appropriate for cameras and area lights. Second, we compute the probability of hitting a node in perspective space.

Our new metric, which we will refer to as the perspective surface area metric (PSAM) more accurately models the distribution of rays that use a perspective space acceleration structure. The metric assumes that ray origins (or destinations) have a uniform distribution on the surface of an axis aligned quad (which will be referred to as the **aperture**). The metric also assumes a uniform distribution of ray directions leaving one side of the aperture. More specifically we define a uniform directional distribution as meaning that if we place a plane at a certain distance from the origin, the spacing of ray intersections on the plane will be uniform. This distribution is equivalent to the equal spacing of pixel centers on an image plane. Figure 2 illustrates this distribution.

The assumption that ray origins are uniformly distributed is almost always accurate given common sampling patterns associated with Monte-Carlo integration. The assumption that outgoing ray directions are uniformly distributed is somewhat less accurate, but we use it anyway for two reasons. First, it is impossible (or at least very difficult) to know the distribution of secondary rays within a scene without first firing primary rays and collecting statistics on the actual distribution. We do not propose solving this problem here. The second, more comforting reason is that the probabilities computed by the PSAM using this assumption are ratios. Given that we are computing ratios, as long as the distribution of rays in the direction of the parent box is locally close to uniform the effects of ray density will cancel and the computed ratios will be close to accurate.

Before we begin the derivation of the PSAM, we would like to outline some properties that it intuitively should have. First, if the area of the light is zero (i.e. the light is a point light) then the probability of a ray striking a box should be proportional to the projected area of the front rectangle of the box. That is, the probability should be proportional to the *area* of the box when projected into two-space. Second, moving twice as far away from a light should have the same effect as shrinking the light by one half in each dimension (following the rule that asymptotically, light falls off quadratically with increasing distance). In other words, area lights should appear smaller the farther away from them you get. In the limit, partitions that are far from an area light should be the same as partitions that are from a point light.

6 Derivation of the PSAH

For our new cost metric, we will use the same form as the traditional SAM:

$$cost_{traversal} \quad = \quad c_{node} + \sum_{children} P_{child} C_{child} \tag{5}$$

The cost estimates will remain the same, simply the number of objects overlapping each child. This section will be dedicated to deriving probability terms using the assumptions described earlier. The probability of a ray striking a child box given that it struck the parent box is the ratio of the number of rays (out of all possible rays) that strike the child over the number of rays that strike the parent. The number of rays that strike a box may be formulated as an integral over all rays of the Boolean intersection function for a ray/box pair. The probability of striking a child box is the ratio of these integrals.

$$p(child) = \frac{\int_{Rays} hit(child, ray) dray}{\int_{Rays} hit(parent, ray) dray} \tag{6}$$

In order to more accurately define these integrals, we will make formal several of our assumptions about the distribution of rays that use this acceleration structure. Rays are assumed to launch from a rectangular aperture A and are assumed to have a uniform distribution of slope. Rays will take the following form:

$$\begin{aligned} ray \quad &:= \quad (o, d) && (7)\\ o \quad &\in \quad [-A_x, A_x] \times [-A_y, A_y] \times [0] && (8)\\ d \quad &\in \quad (-\infty, \infty) \times (-\infty, \infty) \times [1] && (9) \end{aligned}$$

Where o is the ray origin taken from a uniform distribution on a rectangle at $z = 0$ and d is the ray direction, taken from a uniform distribution of intersections with a plane at $z = 1$. See figure 2 for a two dimensional cross section of this distribution in direction.

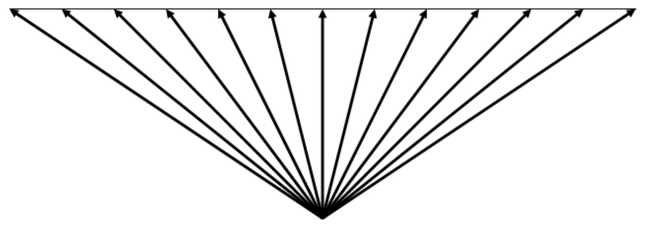

Figure 2: A cross section of rays with a "uniform" distribution in direction (by our assumptions). These rays are **not** uniformly distributed in angle, but this is the common distribution for eye rays. This style of uniformity also translates into perspective space nicely.

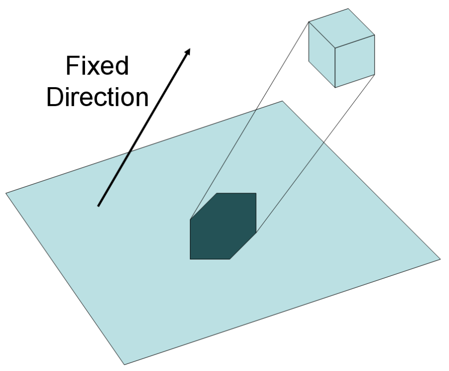

Figure 3: Given a fixed direction (indicated) the shadowed region is the set of origins that will intersect the box. The area of the shadowed region is computed in (26) - (28).

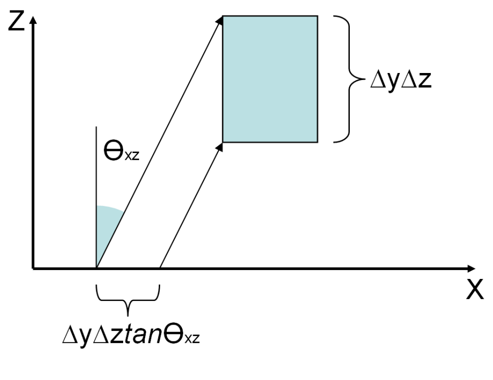

Figure 4: A cross section of a box being projected onto a plane. This figure is designed to give intuition into equation (27). The planes of the box perpendicular to the $z = 0$ plane have areas scaled by the tangent of the direction vector.

However, since we are computing our cost metric in perspective space, we must transform our rays into perspective space. First, we take the equations defining a ray in world space:

$$\begin{aligned} x &= d_x t + o_x && (10) \\ y &= d_x t + o_y && (11) \\ z &= t && (12) \end{aligned}$$

Then we transform these equations into perspective space:

$$\begin{aligned} x' = x/z &= (d_x t + o_x)/t && (13) \\ y' = y/z &= (d_y t + o_y)/t && (14) \\ z' = -1/z &= -1/t && (15) \end{aligned}$$

By defining a new parametric variable for the ray, $t' = 1/t$, we can rewrite the perspective equations as:

$$\begin{aligned} x' &= o_x t' + d_x && (16) \\ y' &= o_y t' + d_y && (17) \\ z' &= -t' && (18) \end{aligned}$$

Notice that the roles of values for o and d have been reversed in perspective space. By introducing new variables $o' = d$ and $d' = o$ representing the ray origin and direction in perspective space we get:

$$\begin{aligned} x' &= d'_{x'} t' + o'_{x'} && (19) \\ y' &= d'_{y'} t' + o'_{y'} && (20) \\ z' &= -t' && (21) \end{aligned}$$

And from the swap of o and d and their initial distributions we also obtain the following distribution for o' and d':

$$\begin{aligned} ray' &:= (o', d') && (22) \\ o' &\in (-\infty, \infty) \times (-\infty, \infty) \times [0] && (23) \\ d' &\in [-A_x, A_x] \times [-A_y, A_y] \times [-1] && (24) \end{aligned}$$

A common confusion regarding the above transform is that many readers think about transforming an origin and a direction separately. This leads to questions regarding limits or infinity because the origin in "normal" space has $o_z = 0$. Rather, we would recommend thinking of this as the transform of a line-equation when following our math. We may now more accurately specify the intersection integral over all rays in perspective space:

$$\int_{-A_x}^{A_x} \int_{-A_y}^{A_y} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} hit(box', ray') do'_{y'} do'_{x'} dd'_{y'} dd'_{x'} \quad (25)$$

In perspective space, if we fix a ray direction (from $[-A_x, A_x] \times [-A_y, A_y] \times [1]$) and assume a uniform distribution of origins, the number of rays that strike a box is equal to the area of that box projected onto the $z' = 0$ plane in the direction of the ray. See figure 3. This area may be computed by summing the projected areas of the three visible faces from the fixed direction. See Figure 4 for reference with regard to the following equations. Given a fixed ray direction direction we have:

$$\begin{aligned} area &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} hit(box', ray') do'_{y'} do'_{x'} && (26) \\ &= \Delta x' \Delta y' + |\tan \theta_{y'z'}| \Delta x' \Delta z' + |\tan \theta_{x'z'}| \Delta y' \Delta z' && (27) \\ &= \Delta x' \Delta y' + |d'_{y'}| \Delta x' \Delta z' + |d'_{x'}| \Delta y' \Delta z' && (28) \end{aligned}$$

Where $\Delta x'$, $\Delta y'$ and $\Delta z'$ are the dimensions of an axis aligned box in perspective space. Substituting this formula into the original integral we get:

$$\int_{-A_x}^{A_x}\int_{-A_y}^{A_y}\left(\Delta x'\Delta y' + |d'_{y'}|\Delta x'\Delta z' + |d'_{x'}|\Delta y'\Delta z'\right)dd'_{y'}dd'_{x'} \tag{29}$$

$$\begin{aligned}
&= 4\int_0^{A_x}\int_0^{A_y}\left(\Delta x'\Delta y' + d'_{y'}\Delta x'\Delta z' + d'_{x'}\Delta y'\Delta z'\right)dd'_{y'}dd'_{x'} && (30)\\
&= 4\int_0^{A_x}\left(A_y\Delta x'\Delta y' + \frac{A_y^2}{2}\Delta x'\Delta z' + A_y d'_{x'}\Delta y'\Delta z'\right)dd'_{x'} && (31)\\
&= 4\left(A_xA_y\Delta x'\Delta y' + \frac{A_xA_y^2}{2}\Delta x'\Delta z' + \frac{A_x^2A_y}{2}\Delta y'\Delta z'\right) && (32)\\
&= A_xA_y(\Delta x'\Delta y' + \frac{A_y}{2}\Delta x'\Delta z' + \frac{A_x}{2}\Delta y'\Delta z') && (33)
\end{aligned}$$

Because probability is a ratio of these integrals, the division will cause common factors to cancel. Thus we may scale the computed integral to:

$$G(box,A) \quad = \quad \Delta x'\Delta y' + \frac{A_y}{2}\Delta x'\Delta z' + \frac{A_x}{2}\Delta y'\Delta z' \tag{34}$$

This expression is similar to the traditional SAM except that it contains scaled aperture terms. With it we may compute the probability of striking each child box and thus the cost of a partition:

$$cost_{traversal} \quad = \quad c_{node} + \sum_{children}\frac{G(child,A)}{G(parent,A)}C_{child} \tag{35}$$

With a solution in hand, let's look back at the intuitive properties we required in an appropriate cost. First, if the area of the aperture is zero (e.g. a light is a point light) than the probability of striking a box should be proportional to the projected area of the front rectangle of the box. Second, if we move twice as far away from a light, it should have the same effect as shrinking the light by one half in each dimension. Both properties are clearly provided by the solution formula. Setting $A = (0,0)$ removes the $\Delta z'$ terms and yields $\Delta x'\Delta y'$. Increasing z_{near} and z_{far} by a factor of two decreases z'_{near} and z'_{far} by a factor of two and thus $\Delta z'$ by a factor of two. Since $\Delta z'$ always occurs multiplied with A_x or A_y, this has the same effect as dividing A by two. This implies that parts of the scene with large z values will not choose splits in z' nearly as often as those in x and y because those terms will dominate the cost function.

This cost function has the useful practical properties that it is simple to compute, is linear in x', y' and z' and requires very little algorithmic change from original SAM to adopt.

7 Implementation and Results

With an appropriate cost metric in hand, we implemented a ray-tracing system to evaluate the effectiveness of perspective space acceleration structures. The ray tracer is a simple, SIMD(4)-wide packet ray-tracer [18] using a kd-tree as the acceleration structure. The primary goal of the implementation is to determine the effectiveness of building and using perspective space acceleration structures versus "regular"-space structures. Great care was taken to ensure that identical build code (except for heuristic evaluation), traversal code and intersection code was used in both regular and perspective space for the fairest comparison.

Geometry is transformed into perspective space and clipped during the frame-setup phase, before being handed to the build algorithm for the acceleration structure. This frame setup cost *is* counted in the **Build Time** column in Table 1. Rays are transformed into perspective space before being traversed. Shadow rays that use a different perspective space than primary rays are transformed between the perspective spaces in a similar method to that used in Z-buffer shadow mapping.

Our results in Table 1 and Table 2 compare the performance of the regular kd-tree versus a perspective kd-tree in two common scenes, bunny69k and fairy-forest from the viewpoints illustrated in Figure 1. Results are provided for eye rays, hard shadow rays, soft shadow rays, and depth-of-field eye rays. Figure 5 provides details about the soft shadow and depth-of-field images. Back-face culling in the perspective space structure reduces build time by approximately 50% in each perspective case. On average, the perspective space acceleration structures, along with the PSAH reduce render times by 20-40% over the traditional SAH and regular space structures. The results demonstrate that even for such a simple case as eye rays, the improvement in trace performance makes up for the additional build time proving end-to-end runtime performance in perspective space similar to render time (excluding build) in regular space. More noticeably, the perspective space acceleration structures reduce the number of intersection tests by 40% on average. This result demonstrates that the perspective-space acceleration structure is of much higher "quality", i.e. it reduces the number of intersection tests performed.

When the number of rays traced is large, perspective space acceleration structures more than compensate for their build cost by improving trace performance. Thus, each area light in a scene should use its own perspective based acceleration structure in order to maximize overall system performance. Memory concerns related to having multiple acceleration structures and large scenes may be addressed by only maintaining one such structure at a time as is described in [6].

In the second table, we compare the performance of different build heuristics in perspective space. The table demonstrates that the PSAM is a significantly more effective cost-estimation metric that either the traditional SAM in perspective space or median split of longest axis in perspective space.

8 Conclusions and Future Work

We have presented an approach that accelerates the tracing of near-common-origin rays (such soft shadows and depth of field effects). The approach uses an adaptive acceleration structure defined in a perspective-transformed space. We have shown that the cost of building multiple perspective-transformed acceleration structures is more than offset by reduced traversal and intersection costs. This tradeoff will be especially evident in extremely high quality renderings with vastly more time spend in traversal. Additionally, we have derived a novel cost metric for building heuristic based acceleration structures in perspective space and have shown this metric to be more effective in practice.

A key feature of the perspective transform is that allows us to continue using all of our current technology. Most other ray-tracing optimizations are agnostic to the space that the rays are being traversed in, and since the perspective transform maps lines to lines, all of the standard high-performance techniques still apply. Future work includes implementing BVHs in perspective space, wider packets and interval arithmetic, frustum culling, mail-boxing and a host of other ray-tracing optimizations in perspective space.

An additional interesting piece of future work is to incorporate focal depth into the PSAM. At the moment, the assumptions about the distribution of ray directions are more accurate for area lights than for depth of field rays, which converge at some depth.

Acknowledgements

This work was supported by the Intel Foundation Fellowship.

Fairy Forest Scene

Acceleration structure	Build Time	Intersections	kd-tree steps	Render Time	Total Time
Eye Rays					
(1) traditional / SAH sort	7.21 s	9.369 M	23.15 M	1.16 s	8.37 s
(2) traditional / SAH scan	0.76 s	9.530 M	24.77 M	1.22 s	1.99 s
(3) perspective / PSAM scan	0.58 s	5.482 M	14.49 M	0.71 s	1.29 s
ratio: (3)/(2)	75%	58%	58%	58%	65%
Depth of Field x16					
(1) traditional / SAH sort	7.24 s	148.0 M	369.6 M	18.5 s	25.8 s
(2) traditional / SAH scan	0.78 s	150.6 M	395.5 M	19.1 s	19.9 s
(3) perspective / PSAM scan	0.58 s	92.60 M	314.7 M	14.5 s	15.1 s
ratio: (3)/(2)	74%	61%	80%	76%	76%
Hard Shadows					
(1) traditional / SAH sort	7.28 s	21.05 M	46.13 M	2.13 s	9.41 s
(2) traditional / SAH scan	0.80 s	21.39 M	49.78 M	2.34 s	3.15 s
(3) perspective / PSAM scan	1.15 s	12.29 M	39.33 M	1.86 s	3.01 s
ratio: (3)/(2)	143%	57%	80%	79%	96%
Soft Shadows x16					
(1) traditional / SAH sort	7.83 s	346.4 M	558.9 M	35.5 s	43.3 s
(2) traditional / SAH scan	0.79 s	347.7 M	601.6 M	34.4 s	35.3 s
(3) perspective / PSAM scan	1.19 s	204.7 M	540.4 M	24.3 s	25.5 s
ratio: (3)/(2)	138%	59%	90%	71%	72%

Bunny69K Scene

Acceleration structure	Build Time	Intersections	kd-tree steps	Render Time	Total Time
Eye Rays					
(1) traditional / SAH sort	2.17 s	2.838 M	10.43 M	0.492 s	2.62 s
(2) traditional / SAH scan	0.26 s	2.888 M	11.10 M	0.532 s	0.79 s
(3) perspective / PSAM scan	0.15 s	1.515 M	5.981 M	0.272 s	0.43 s
ratio (3)/(2)	59%	52%	54%	51%	54%
Depth of Field x16					
(1) traditional / SAH sort	2.09 s	81.19 M	255.1 M	12.7 s	14.8 s
(2) traditional / SAH scan	0.25 s	83.15 M	272.4 M	13.3 s	13.5 s
(3) perspective / PSAM scan	0.15 s	46.29 M	141.1 M	7.8 s	8.0 s
ratio: (3)/(2)	62%	56%	52%	66%	59%

Table 1: Comparison of performance: perspective-space kd-tree vs. traditional world-space kd-tree. **"traditional / SAH sort"** is a traditional kd-tree built using the SAH with sort-based selection at every step. **"traditional / SAH scan"** is a traditional kd-tree built using the SAH with a scan-based selection at every step [7]. **"perspective / PSAM scan"** is a perspective-space kd-tree built using this paper's perspective-space cost metric with a scan-based selection at every step. All results are at 1920x1200 resolution on a single core of a mobile 2.2 Ghz Intel Core 2 Duo (Merom). **ratio** is the ratio of the perspective-space results to the traditional scan-based results. Build time results include all build times (if more than one structure is used). All acceleration structures specialized to a light or camera use face culling.

Fairy Forest Scene

Build heuristic	Build Time	Intersections	kd-tree steps	Render Time	Total Time
Eye Rays					
Median split	0.209 s	63.85 M	48.76 M	4.16 s	4.38 s
SAH	0.449 s	11.46 M	31.64 M	1.43 s	1.88 s
PSAM	0.581 s	5.482 M	14.49 M	0.71 s	1.29 s
Depth of Field x16					
Median split	0.208 s	999.1 M	794.4 M	65.8 s	66.0 s
SAH	0.472 s	177.2 M	519.5 M	23.3 s	23.8 s
PSAM	0.580 s	92.60 M	314.7 M	14.5 s	15.1 s

Table 2: Comparison of different build heuristics for a perspective-space kd-tree. **Median-split** uses the median split of the longest axis. **SAH** uses the traditional SAM, but in perspective space. **PSAM** uses this paper's new perspective-space PSAM. All results are at 1920x1200 resolution on a single core of a mobile 2.2 Ghz Intel Core 2 Duo (Merom).

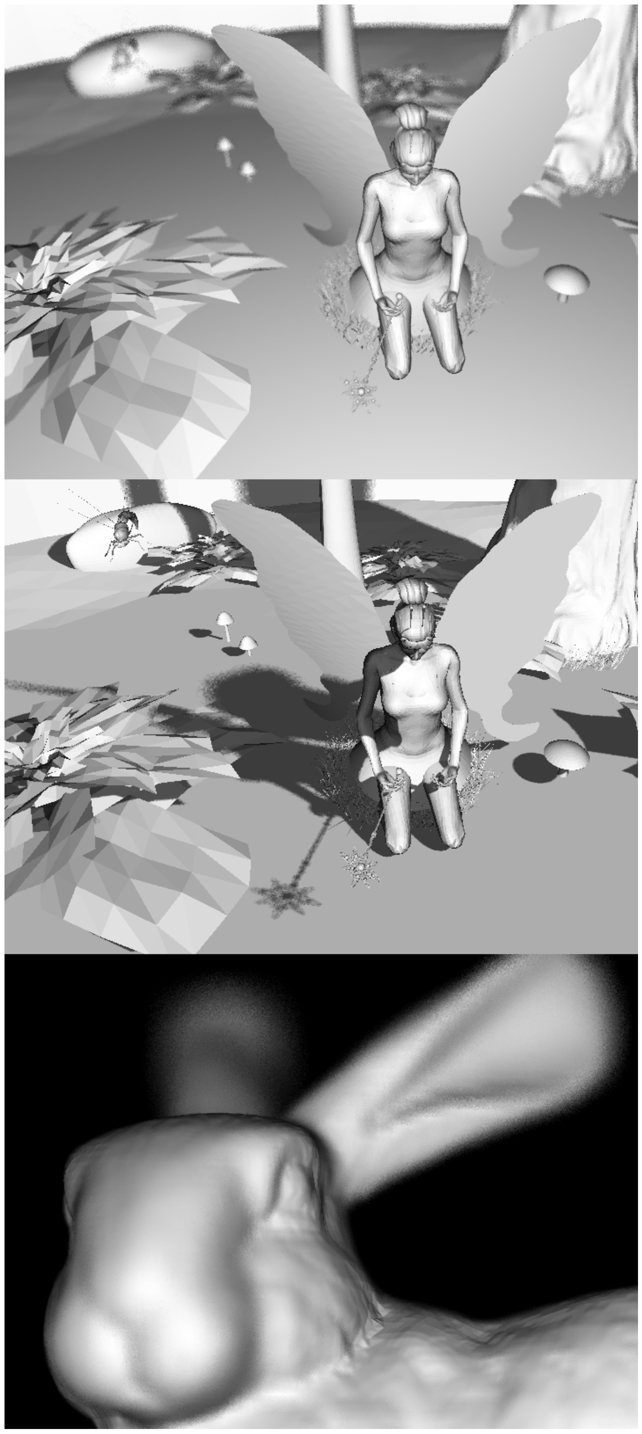

Figure 5: Close-ups of depth of field and soft-shadow images. Top: Fairy Forest scene with depth-of-field. Middle: Fairy Forest scene with soft shadows. Bottom: Bunny scene with depth-of-field. Details of the soft shadow and depth-of-field rays are as follows. First, bunny is approximately 0.15 x 0.15 x 0.12 units in size and Fairy Forest is approximately 6.3 x 1.6 x 6.3 units in size. Fairy Forest with depth-of-field uses a focal depth of 0.91 and an aperture “radius” of 0.01. Fairy Forest with soft shadows has a light with “radius” 0.1. Bunny with depth-of-field uses a focal depth of 1.98 and an aperture “radius” of 0.1 (the bunny is viewed at a distance of 2.0). Since all apertures are square, a “radius” of x is in fact a quad of size $2x \times 2x$.

References

[1] J. Bigler, A. Stephens, and S. G. Parker. Design for parallel interactive ray tracing systems. In *IEEE Symp. on Interactive Ray Tracing 2006*, pages 187–196, Sept. 2006.

[2] P. Djeu, W. Hunt, R. Wang, I. Elhassan, G. Stoll, and W. R. Mark. Razor: An architecture for dynamic multiresolution ray tracing. Technical report, University of Texas at Austin Dep. of Comp. Science, 2007. Conditionally accepted to ACM Transactions on Graphics.

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

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

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

[6] W. Hunt and B. Mark. Ray-specialized acceleration structures for ray tracing. In *IEEE Symposium on Interactive Ray Tracing 2008*. IEEE, 2008.

[7] W. Hunt, W. Mark, and G. Stoll. Fast kd-tree construction with an adaptive error-bounded heuristic. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 81–88, 2006.

[8] C. Lauterbach, S.-E. Yoon, D. Tuft, and D. Manocha. RT-DEFORM: Interactive ray tracing of dynamic scenes using BVHs. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 39–45, 2006.

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

[10] G. S. Miller and C. R. Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments, 1984.

[11] M. Pharr and G. Humpreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004.

[12] S. Popov, J. Günther, H.-P. Seidel, and P. Slusallek. Experiences with streaming construction of SAH kd-trees. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 2006.

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

[14] A. Reshetov, A. Soupikov, and J. Hurley. Multi-level ray tracing algorithm. In *SIGGRAPH ’05: ACM SIGGRAPH 2005 Papers*, pages 1176–1185, New York, NY, USA, 2005. ACM.

[15] 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.

[16] 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).

[17] 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.

[18] I. Wald, P. Slusallek, C. Benthin, and M. Wagner. Interactive rendering with coherent ray tracing. In *Proc. of Eurographics 2001*, 2001.

[19] Y. Wang and S. Molnar. Second-depth shadow mapping. Technical report, Chapel Hill, NC, USA, 1994.