TIME COSTS OF RAY TRACING WITH AMALGAMS

Kelvin Thompson* and Donald Fussell

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-91-01 January 1991

ABSTRACT

The use of renderable bounding volumes—or amalgams—is potentially a useful extension to traditional ray tracing techniques. Analysis indicates that simple, highly detailed scenes containing amalgams should render more quickly than scenes modeled with traditional techniques. Ray tracers operating on such scenes arguably have better asymptotic time complexity—in both the best case and the expected case—than traditional rendering systems.

Keywords: Photorealistic Image Generation, Display Algorithms, Object Hierarchies, Ray Tracing.

* Department of Electrical and Computer Engineering.

Introduction

Most ray tracers use bounding volumes — invisible scene objects that enclose other objects — as a means of accelerating rendering [Glas89]. Some kinds of bounding volumes partition the world space of the scene or some related space [Glas84, NaTh86]. Other schemes group clusters of objects [Kay86, GoSa87]. The fundamental purpose of any bounding volume implementation is to reduce the number of primitives the renderer must test for intersection with a given ray.

We propose extending the definition of a bounding volume so that it has visual attributes (e.g. color, reflectance, opacity) in addition to the usual geometric attributes (e.g. position, orientation, size). We call these enhanced bounding volumes *amalgams*. When a ray intersects an amalgam, the renderer may choose to treat the amalgam either as a primitive or as a traditional bounding volume. That is, the ray tracer may *render* the amalgam using conventional shading techniques, or it may *traverse* the amalgam — recursively test the objects inside for intersection. Since a ray tracer can use amalgams to choose the sizes of objects it renders, we call ray tracers that use amalgams *scale-sensitive renderers*.

When the ray tracer renders the amalgam, it ignores the objects inside the amalgam; but with a traditional bounding volume the renderer would have to test many of these children for intersection. Below we analyze whether this amalgam-by-amalgam reduction in intersection tests translates into improved overall performance.

In many cases amalgams are translucent. We study two ways a ray tracer can model translucency (or, conversely, opacity) to obtain the shaded color of an amalgam. With the first model — *deterministic opacity* — the renderer blends the shaded color of the amalgam itself with the color of light passing through the amalgam. With *stochastic opacity*, however, the renderer randomly decides (based on opacity) either that the amalgam is completely opaque or completely transparent. When a ray intersects a translucent amalgam modeled with deterministic opacity, the renderer must perform shading operations on both the amalgam and any objects behind it. However, if the renderer uses the stochastic model and decides the amalgam is opaque, then objects behind the amalgam can be ignored.

Problem Statement

Ray tracing involves many operations: deriving ray parameters, blending light colors and intensities, following links between scene objects. However, the costs of performing ray/object intersection tests overwhelms the costs of other operations. Since the earliest implementations of ray tracing, experience has shown that the primary factor affecting ray tracing performance is the number of intersection tests performed to generate an image

[Whit80]. So the analysis below will measure time costs in the number of intersection tests between rays and scene objects (primitives, bounding volumes, and amalgams).

Also, to keep costs independent of image resolution, the analysis will consider only the costs associated with a single primary ray R_0 and the ray tree it spawns.

It is extremely difficult to characterize all possible 3D scenes a ray tracer might encounter, and it is even more difficult to analyze the time cost of ray tracing all possible scenes. Accordingly, we assume a simple, stylized 3D scene for cost analysis. The simplified scene is a generalization of many reasonable 3D scenes, so, despite the simplification, the cost analysis should still be useful.

We assume the 3D scene is composed of a uniform hierarchy of amalgams, with primitives at the bottom of the hierarchy, and a single amalgam bounding the scene at the top of the hierarchy. Levels of the hierarchy are numbered from 0 to l_{max} (for the top and bottom levels, respectively). When a ray intersects an amalgam in the scene, the ray intersects n child amalgams; when a ray intersects an amalgam at level l_{max}-1 (just above the bottom, primitive level), the renderer must perform intersection tests on n objects inside the amalgam. Figure 1 shows a stylized view of objects the ray encounters at different levels of the hierarchy (n=3 in the figure). There are n^i objects on level i, and $p \equiv n^{l_{max}}$ primitives on the bottom level. Much of the discussion below implicitly refers to this figure.

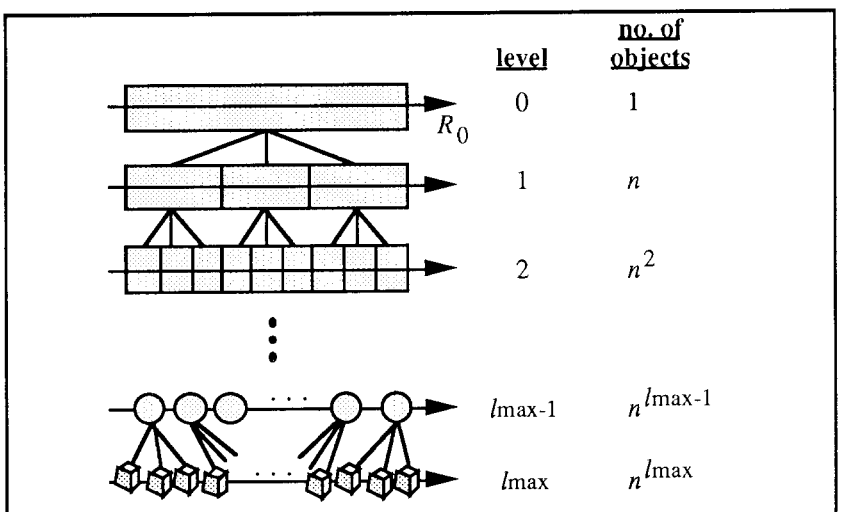

Figure 1 — Scene Hierarchy

Figure 2 shows a 3D scene that produces this kind of behavior. This scene is composed of several nested levels of rectangular prism amalgams. Note that the actual number of objects on each level is not necessarily the same as the number of objects a given ray intersects. Each amalgam in Figure 2 contains eight child amalgams, but if the renderer samples the scene with a ray that is orthogonal to an amalgam face, then the ray will intersect

only two children of each amalgam. If each amalgam at the bottom of the hierarchy contains two primitives, then the ray encounters a uniform hierarchy like the one in Figure 1, but with n=2.

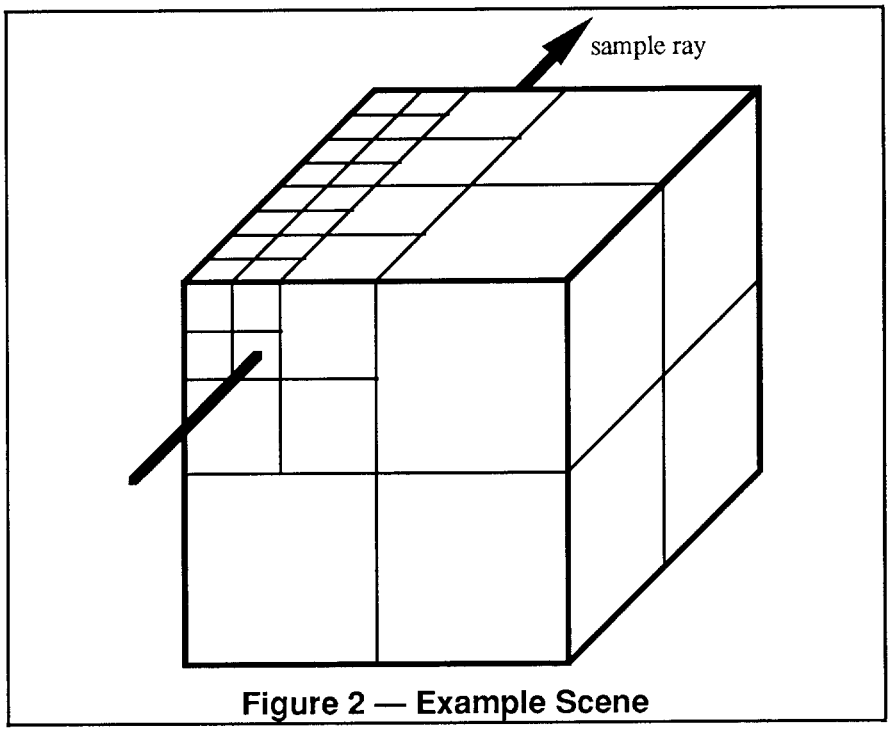

Figure 2 — Example Scene

We consider p, the number of primitives the ray may examine, to be the primary variable characterizing the size of the input data. The branching factor n of the hierarchy is a fixed, constant attribute of the hierarchy; the height of the hierarchy varies logarithmically with the basic size of the database, i.e. $l_{max} = \log_n p$.

We would like to express the cost in terms of the number of primitives in the scene. However, in most scenes, p is not the number of primitives; instead, p is some function of the number of primitives. For example, in the sampling situation in Figure 2, p varies as the square root of the number of primitives (just as the volume of intersected amalgams with a given size varies with the square root of the overall volume). Still, Figure 3 shows a pathological database where p is exactly the number of primitives. In this scene, all bounding volumes and light sources lie along the line of the primary ray, so the renderer must examine all scene objects for both primary and secondary rays.

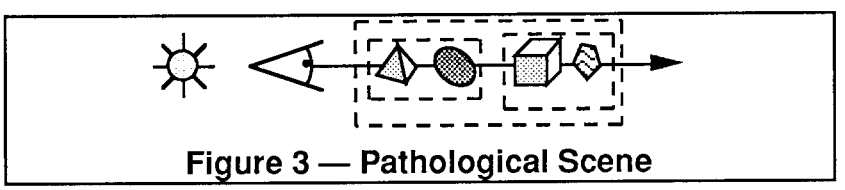

Figure 3 — Pathological Scene

Next, we assume that primitives do not have refraction or mirror reflection shading attributes, so renderers spawn only shadow rays as secondary rays. We have found that amalgams seldom need to spawn reflection or refraction rays, so placing the same restriction on primitives keeps traditional and scale-sensitive renderers on a more even footing.

Finally, we assume there is a single light source in the scene. Further below we show that all results are scaled by the number of light sources, so the number of light sources is not important for relative comparisons.

Worst Case Analysis

For worst case analysis we assume the primary ray R_0 intersects only the last primitive it tests for intersection. This means the renderer may have to perform intersection tests on — or *examine* — all of the objects in Figure 1.

When the traditional ray tracer spawns a primary ray, the renderer must examine all objects in Figure 1. The cost for the primary ray is the number of objects in the hierarchy:

$$C_{t0} = \sum_{i=0}^{l_{max}} n^i = \frac{n^{l_{max}+1} - 1}{n - 1} = O(n^{l_{max}}) = O(p) .$$

The renderer also will spawn a single shadow ray from the last primitive it examines. At worst, this shadow ray will have the same asymptotic cost as the primary ray, so the overall cost is $O(p)$.

The cost for the scale-sensitive renderer depends on which amalgams it can render. If the renderer chooses to render to the deepest levels of the hierarchy, it will exhibit the same asymptotic complexity as the traditional renderer. However, if the ray tracer renders amalgams at level $l_1 < l_{max}$ — and hence does not have to traverse any deeper than level l_1 — then the cost of processing the primary ray is

$$C_{s0} = \sum_{i=0}^{l_1} n^i = \frac{n^{l_1+1} - 1}{n - 1} = O(n^{l_1}) . \quad (1)$$

In the worst case, all n^{l_1} amalgams on level l_1 will be translucent. If the renderer uses deterministic opacity, it must spawn a shadow ray from each of them. Assuming each of these shadow rays also has cost C_{s0}, the cost of processing all shadow rays is

$$C_{sd1} = n^{l_1} C_{s0} = \frac{n^{2l_1+1} - n^{l_1}}{n - 1} = O(n^{2l_1}) . \quad (2)$$

The total cost for deterministic opacity is therefore $C_{s0}+C_{sd1}=O(n^{2l_1})$.

Effectively, the scale-sensitive renderer is more efficient than the traditional renderer only when the scale-sensitive renderer traverses less than halfway down the object hierarchy

(i.e. when $2l_1<l_{max}$). If the scale-sensitive renderer stays above this threshold in the hierarchy, it will be faster. Otherwise the traditional renderer will be faster.

What fraction of the database is above the threshold? If we set $l_1=l_{max}/2$, then the "efficient" part of the database (for the scale-sensitive renderer) contains $O(n^{l_{max}/2})$ objects. Since the entire database contains $O(n^{l_{max}})$ objects, the "efficient" fraction of the database is

$$\frac{O(n^{l_{max}/2})}{O(n^{l_{max}})} = O(n^{-l_{max}/2}) = O(\sqrt{1/p}) .$$

So as the number of primitives p grows large, the fraction of the hierarchy that the scale-sensitive ray tracer can render efficiently grows small.

Clearly, deterministic opacity will not be efficient in the general case: the scale-sensitive renderer provides lower cost per ray than traditional renderers, but the use of deterministic opacity can spawn so many secondary rays that the savings are overwhelmed.

With stochastic opacity, however, the renderer spawns at most one shadow ray, which has the same, low cost as the primary ray (C_0). The grand total cost for the stochastic opacity is therefore

$$2\,C_{s0} = O(n^{l_1}) . \tag{3}$$

This cost is independent of the number of primitives — no matter how much data there is below level l_1, the renderer will take the same time to execute. In contrast, the execution time for the traditional renderer will grow with the number of primitives in the scene.

For example, suppose a scale-sensitive ray tracer renders the scene in Figure 2 to a certain level in the hierarchy. Now suppose we redefine the database so that there are several additional levels of amalgams, and therefore many additional primitives. If the ray tracer still renders the same amalgams as before, then the execution time should be the same — the cost has not changed even though there are many times more primitives in the scene. In contrast, a traditional renderer would have to examine many of the new amalgams and primitives, so it would take longer to render the second scene.

Expected Case Analysis

For the expected case analysis we assume a ray may intersect a primitive within any amalgam...not just the last. If a ray does intersect a primitive, the renderer may be able to ignore (i.e. not test for intersection) all subsequent amalgams and primitives on the ray. We estimate the probabilities that the renderer will examine varying numbers of objects on a ray, then average the weighted counts to get the expected cost.

The expected case analysis assumes that the renderer can examine the bounding volumes in sorted order — nearest to farthest from the ray origin — along the ray*. Many modelers that use space-partitioning schemes [Glas84, NaTh86] give amalgams to the renderer in sorted order. However, object-grouping schemes [Kay86, GoSa87] usually do not. Modelers based on these schemes must sort groups of amalgams from nearest to farthest. In practice, however, the cost of this sorting is insignificant compared to the cost of performing intersection tests. Also, in the database in Figure 1, the renderer sorts exactly n objects at a time. The renderer can use a linear-time bucket sort for this operation, which means the sort has the same asymptotic cost as examining the objects in the first place.

Let us first calculate an expression for expected time that might apply to many rendering situations. We will then evaluate the general expression in terms of the specific behavior of traditional and scale-sensitive ray tracers. We will concentrate on the expected time for a single primary ray R_0, since the expected times for secondary rays should have the same asymptotic behavior. Also, since expected case analysis requires different stochastic rendering outcomes, the analysis below will not consider deterministic opacity.

Let us calculate the cost of traversing as far as level $l_1 < l_{max}$ in the hierarchy. Figure 4 shows several possible traversals through level l_1. The traditional renderer always traverses to the bottom of hierarchy, so $l_1 = l_{max}-1$; the scale-sensitive renderer may traverse to some level $l_1 < l_{max}-1$. Both renderers stop traversal after examining i bounding volumes on the level (i.e. the renderer detects intersection with a primitive inside the i-th bound).

All traversals in Figure 4 involve l_1 objects on the "left edge"† of the hierarchy and i objects on level l_1. Accordingly, a lower bound for the cost of any of the traversals in Figure 4 is

$$l_1 + i = \log_n b + i , \tag{4}$$

where $b \equiv n^{l_1}$ is the number of bounding volumes on level l_1.

---

* The worst case analysis did not require this assumption; instead, the analysis assumed that only the last primitive intersected the ray, so the ordering of amalgams was irrelevant.

† The "left edge" consists of the first object on each level above l_1; the renderer must traverse these objects to reach the first bound on l_1.

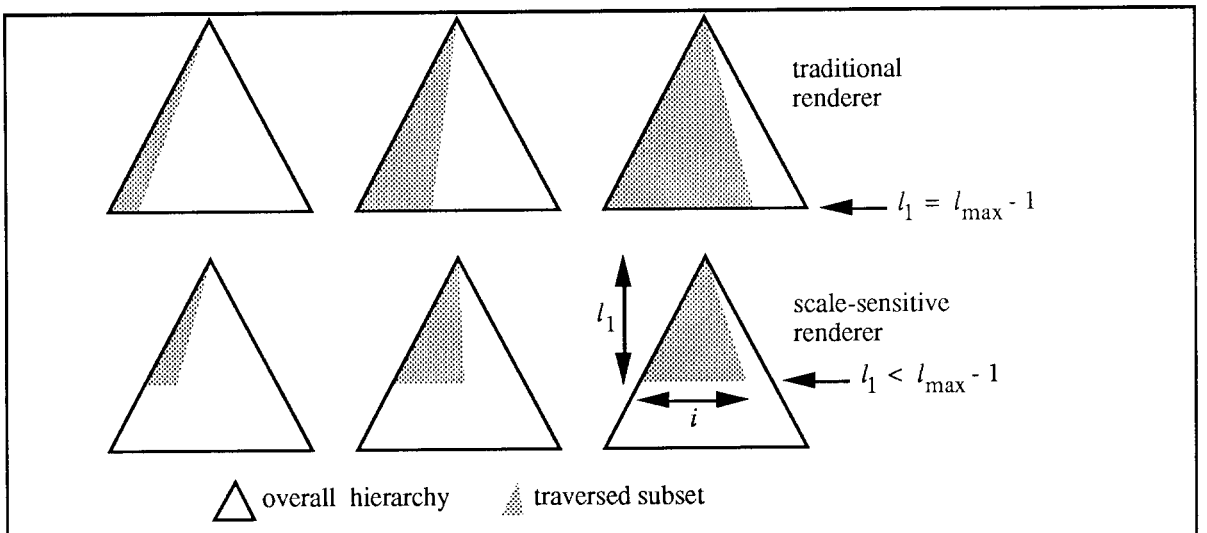

Figure 4 — Expected Case Outcomes

However, Equation 4 does not account for "interior" objects in the traversal — objects above l_1 and to the right of the left edge. To account for these objects, we first estimate the total number of traversed and untraversed objects above and within level l_1:

$$C_{l_1} = \sum_{i=0}^{l_1} n^i = \frac{n^{l_1+1} - 1}{n - 1} \approx \frac{n^{l_1+1}}{n - 1} = \frac{n\,b}{n - 1}. \tag{5}$$

If we distribute this cost equally among all of the b amalgams on the level, the average cost associated with reaching each amalgam is

$$C_b \approx \left(\frac{1}{b}\right)\left(\frac{n\,b}{n - 1}\right) = \frac{n}{n - 1}. \tag{6}$$

So we may estimate the total cost of reaching the first i objects on level $l_1$as

$$i \cdot C_b . \tag{7}$$

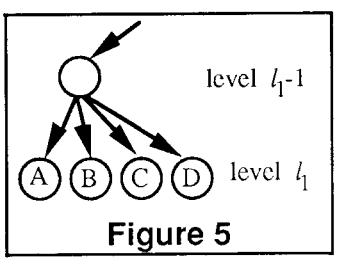

Figure 5

However, we must be careful to regard C_b as only an average cost associated with each object when the renderer traverses many objects. For example, Figure 5 shows a few objects at the left and bottom portion of a hierarchy. Equation 4 tells us the cost of traversing through object A is l_1+1, which this is usually greater than the cost $1 \cdot C_b$ that Equation 7 estimates. However, traversing through each of the subsequent objects B, C, and D only adds one additional unit of cost to the overall cost; this reduces the average cost per object. As the renderer traverses through more objects on l_1, the estimate in Equation 7 becomes still more accurate.

We can also see that C_b should indeed be independent of l_1: the number of objects on level l_1 is $b=n^{l_1}$, but the number of all objects in the hierarchy through l_1 is also $O(n^{l_1})$

(Equation 1). Hence the cost *per object* on any row is constant. However, lower levels of the hierarchy will contain more bounding regions, so the *overall* traversal cost still will grow for deeper levels.

Equation 4 serves as a lower bound on the cost of a given traversal, but it does not account for traversal of internal nodes of the hierarchy for large i. Conversely, Equation 7 is a good approximation of the traversal cost for only large i. Hence we write the overall cost for traversing through i amalgams on on level l_1 as

$$\max(\log_n b + i,\ i \cdot C_b) . \tag{8}$$

To calculate the expected traversal cost, we must weight the cost of each possible traversal by the probability that the traversal will occur. Both terms of the overall cost in Equation 8 depend asymptotically on i, so the expected overall cost will depend on the expected value of i. Accordingly, we now turn our attention to calculating the expected value of i.

To obtain the probability of traversing any number of amalgams on l_1, we model intersections of R_0 with primitives as a Poisson process. So if we expect x intersections along some portion of R_0, then the probability that there will be no intersections along that portion is e^{-x}. Along the whole ray we expect p intersections, and within each bounding volume on level l_1 we expect p/b intersections. With a traditional renderer, the probability that the renderer will continue traversal through a single bounding volume on l_1 is $e^{-p/b}$, the probability of no intersections with primitives in the bounding volume. With the scale-sensitive renderer, the probability that R_0 will stochastically "miss" an amalgam is also $e^{-p/b}$.

On any given traversal, the probability that the renderer will visit only the first i amalgams on level l_1 is

$$\begin{aligned} (e^{(-p/b)(i-1)})\,(1-e^{(-p/b)}) \;&=\; (e^{p/b}-1)\,(e^{-p/b})^i && \text{for } i<b \\ 1 - \sum_{i=1}^{b-1} \left(e^{p/b}-1\right)\left(e^{-p/b}\right)^i \;&&& \text{for } i=b. \end{aligned}$$

So the expected number of amalgams is

$$\sum_{i=1}^{b-1} i \left(e^{p/b}-1\right)\left(e^{-p/b}\right)^i + b\left(1 - \sum_{i=1}^{b-1} \left(e^{p/b}-1\right)\left(e^{-p/b}\right)^i\right)$$

$$= \left(e^{p/b}-1\right)\sum_{i=1}^{b-1} i \left(e^{-p/b}\right)^i + b\left(1 - \left(e^{p/b}-1\right)\sum_{i=1}^{b-1} \left(e^{-p/b}\right)^i\right).$$

We can simplify the sums above with the relations

$$\sum_{i=1}^{m} x^i = \frac{x^{m+1} - x}{x - 1} \quad \text{and} \quad \sum_{i=1}^{m} i\,x^i = \frac{(xm - m - 1)\,x^{m+1} + x}{(x - 1)^2}$$

to get

$$\begin{aligned}
&\left(e^{p/b}-1\right)\left(e^{-p/b}-1\right)^{-2}\left[\left(\left(e^{-p/b}\right)(b-1) - (b-1) - 1\right)\left(e^{-p/b}\right)^{b} + e^{-p/b}\right] \\
&\quad + b\left(1 - \left(e^{p/b}-1\right)\left(e^{-p/b}-1\right)^{-1}\left[\left(e^{-p/b}\right)^{b} - e^{-p/b}\right]\right) \\
&= \left(e^{p/b}-1\right)\left(e^{-p/b}-1\right)^{-2}\left[\left(b\,e^{-p/b} - e^{-p/b} - b\right)e^{-p} + e^{-p/b}\right] \\
&\quad - b\left(e^{p/b}-1\right)\left(e^{-p/b}-1\right)^{-1} e^{-p} \quad . \qquad (9)
\end{aligned}$$

Traditional Renderer. Now let us consider Equation 9 in light of the behavior of the traditional renderer. This renderer traverses to the deepest level that contains bounding volumes, so $l_1=l_{\max}-1$. (On the lowest level, $l_{\max}$, the renderer traverses "clumps" of unordered primitives, where each "clump" is delimited by a bounding region at level $l_{\max}-1$. This means the behavior of the renderer at level $l_{\max}-1$ best matches the modeled behavior that led to Equation 9.) Also, $p/b=n$ is constant, and hence b is a function of the number of primitives, i.e. $b=p/n$. Applying these substitutions to Equation 9, we get

$$\begin{aligned}
&\left(e^{n}-1\right)\left(e^{-n}-1\right)^{-2}\left[\left(\frac{p}{n}e^{-n} - e^{-n} - \frac{p}{n}\right)e^{-p} + e^{-n}\right] \\
&\quad - \frac{p}{n}\left(e^{n}-1\right)\left(e^{-n}-1\right)^{-1} e^{-p} \quad . \qquad (10)
\end{aligned}$$

We know that as p grows larger, e^{-p} grows small very rapidly compared to any constant $k>0$. So we apply the following initial simplification to Equation 10

$$a\,e^{-cp} + k \;\rightarrow\; k, \qquad (11)$$

for constants $c>0$ and $a,k\neq 0$. This allows us to remove the $-e^{-n}$ term nested most deeply in the upper line of Equation 10. With this modification, the terms involving pe^{-p} cancel, and we are left with

$$e^{-n}\,(e^{n}-1)\,(e^{-n}-1)^{-2} = (1-e^{-n}) = O(1)\,. \qquad (12)$$

So we expect the traditional renderer to traverse a constant number of amalgams on level $l_{\max}-1$ of the hierarchy. However, the logarithmic component of Equation 8 dominates the overall complexity, so the expected cost for the renderer is $O(\log p)$.

Scale-Sensitive Renderer. Now let us calculate the expected number of amalgams the scale-sensitive renderer will traverse on level l_1. If we consider l_1 to be a fixed number of levels

above the bottom level l_{max}, then $p/b = n^{l_{max}-l_1}$ is constant and the time cost closely resembles Equations 3 and 12 (with n in most expressions replaced by $n^{l_{max}-l_1}$).

Alternatively, we may consider l_1 to be independent of l_{max}, which implies that b is constant and independent of p. In this case we must factor expressions involving $e^{p/b}$ out of Equation 9. Applying the substitution

$$e^{p/b}\, e^{-p} = e^{-p(1-1/b)}$$

to Equation 9 we get

$$\left(e^{-p(1-1/b)} - e^{-p}\right)\left(e^{-p/b}-1\right)^{-2}\left(b\, e^{-p/b} - e^{-p/b} - b\right) - \left(e^{-p/b}-1\right)^{-1} - b\left(e^{-p(1-1/b)} - e^{-p}\right)\left(e^{-p/b}-1\right)^{-1} \quad . \tag{13}$$

And when we apply the approximation in Equation 11 we again get $O(1)$. However, the logarithmic term of Equation 8 still dominates, so the overall expected cost is $O(\log b)$.

In summary, each type of renderer (traditional vs. scale-sensitive) has an expected asymptotic cost of the number of levels it traverses in the hierarchy. Since the scale-sensitive renderer often traverses fewer levels than the traditional renderer (and never more), it should perform better in the expected case as well.

A Second Look

This section reexamines some of the simplifications made for the analysis above. The earlier analysis covers several combinations of worst case vs. expected case, traditional vs. scale-sensitive, and determinisitic vs. stochastic implementations. However, most overall costs are expressed in terms of the cost for a single primary ray — only deterministic opacity implementations have an additional dependency: the number of amalgams that intersect the primary ray on a given level. So most of the reevaluation below concerns the cost of a single primary ray.

"Failed" Intersection Tests. Usually, when the modeler uses an object-grouping scheme for the scene hierarchy, the renderer examines more objects than the previous analysis suggests. With such schemes, the renderer must test for intersection with *all* children of any amalgam it traverses — the n children that we discuss above may be a small subset of all the children of an amalgam. For example, for the ray in Figure 2, the renderer will test for intersection with eight children of each amalgam, but the analysis above only accounts for the n=2 of these that actually intersect the ray. We call tests on objects that do not intersect the ray *"failed" intersection tests.*

Let us revise the analysis above under the assumption that the renderer examines an additional m "failed" objects inside each amalgam. So the renderer examines a total of $n+m$ objects inside each amalgam, but it still continues traversal through only n of them. This traversal hierarchy is much the same as the one discussed earlier, except at each level $l_i>0$ we have an additional m objects for each amalgam in the next highest level — level l_i-1. In all, there are an additional $m{\cdot}n^{l_i-1}$ amalgams at each level l_i of the hierarchy.

In the worst case, the number of objects examined through level l_1 for a primary ray R_0 is

$$n^0 + \sum_{i=1}^{l_1} \left(n^{l_1} + m\, n^{l_1-1}\right) = 1 + \left(1 + \frac{m}{n}\right) \sum_{i=1}^{l_1} n^{l_1} = O(n^{l_1}) .$$

In absolute terms, the renderer examines more objects than the previous analysis indicated. However, the asymptotic number of objects is the same, so the overall results for the primary ray are still valid.

Costs for worst case secondary rays also will be $O(n^{l_1})$, so the earlier results for traditional and stochastic opacity implementations are still correct. The deterministic opacity implementation also depends on the number of amalgams on l_1 that actually intersect the primary ray. This value does not change with our more general assumptions, so the deterministic result is still correct.

"Failed" intersection tests essentially scale the cost of examining each amalgam, but scaling does not affect asymptotic results. It is the structure of *branching* in the algorithm (and hierarchy) that determines the asymptotic results. So the asymptotic costs for the expected case are also valid.

Multiple Light Sources. Let us suppose several point or directional light sources exist in the scene. Most of the implementations analyzed above spawn one secondary ray for every light source in the scene. Each of these secondary rays has the same cost as the primary ray, so the overall cost is effectively scaled by the number of light sources. If we consider the number of light sources to be a constant property of the 3D scene, asymptotic results are not affected by the number of light sources.

The determinisitic opacity implementation is slightly different. In this case, the renderer spawns a secondary ray for each light source *and* for each rendered amalgam. However, this again only scales the results in Equation 2, so results remain the same.

Linear Cost. Some researchers have asserted that object hierarchies like the ones described above should have logarithmic worst case asymptotic costs, i.e. $O(\log p)$. For example, Clark

proposes a scanline renderer resembling our ray tracing renderer. He notes that traversal for his speculative renderer "resembles a logarithmic search." [Clark76] However, the worst case results above do not have a logarithmic component.

Let us consider the canonical, balanced-tree search algorithm that Clark mentions — we call the algorithm *FIND*. Figure 6 shows a typical tree structure that such a search algorithm might use. The lowest nodes of the tree (the *leaf nodes*) contain groups information (*records*) we wish to search for, and each record is identified by a number (a *key*). The upper nodes of the tree describe which ranges of keys exist in lower parts of the tree. Given a key, *FIND* locates the matching record in this type of structure by following the links describing ranges of numbers that contain the key. For example, in the figure, to locate record "3," *FIND* would start at the root of the tree and traverse the highlighted nodes.

There are many implementations of the general idea sketched in Figure 6. However, for all of them, the worst case asymptotic time required to find a record in the structure is $O(\log p)$ — where p is the number of records at the bottom of the structure — for both worst case and expected case. Intuitively, we can understand this result because *FIND* must examine only one node on each level in the tree — the algorithm's execution time will be proportional to the height of the tree, which is $O(\log p)$.

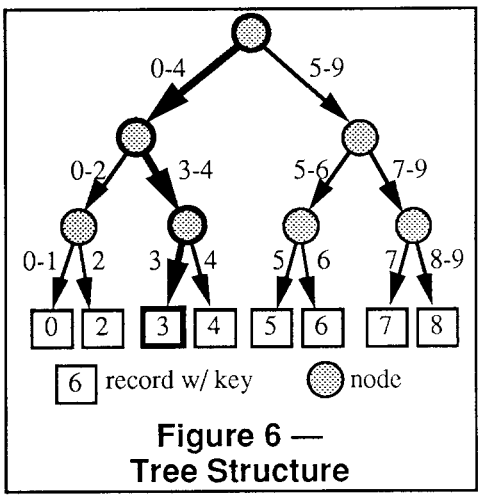

Figure 6 — Tree Structure

The ray tracer's traversal of the hierarchical database in Figure 1 appears to be quite similar to *FIND*'s traversal of the tree in Figure 6, so we might expect that the ray tracer, too, should have $O(\log p)$ cost. However, as sketched in Figure 2, the ray tracer usually will have to traverse more than one child of each amalgam in the hierarchy. This means the set of nodes the ray tracer examines is itself a branching, treelike hierarchy — the examined hierarchy is smaller than the overall database, but it still has a branching structure. Such a structure will always have $O(p)$ objects, where p is the number of objects at the bottom of the structure. Usually p is a non-logarithmic function of the actual number P of primitives in the scene — e.g. $p=O(P)$ or $p=O(\sqrt{P})$ — so the cost for the ray tracer is non-logarithmic.

FIND and the ray tracer are both search algorithms in a general sense — both attempt to locate objects that match a set of criteria. The crucial difference between the algorithms is that *FIND* can identify exactly one child node it must traverse for each parent node, while the ray tracer must explore several children of each parent. This difference exists because each

level of the hierarchy for *FIND* exactly partitions the space of record keys, but the 3D scene hierarchy does not partition the space of the ray tracer's "key" — the ray.

To better understand this partitioning problem, let us consider a search algorithm *PT2D* that is given a point in a 2D "scene" and locates primitives that intersect the point. Figure 7 shows two bounding volume hierarchies that *PT2D* might operate on. In each "scene," thicker and solid lines denote bounds that are higher in the hierarchy (no objects are shown). In Figure 7a, each bounding region has four child regions that exactly tile the parent. In Figure 7b, each bounding region has overlapping child regions.

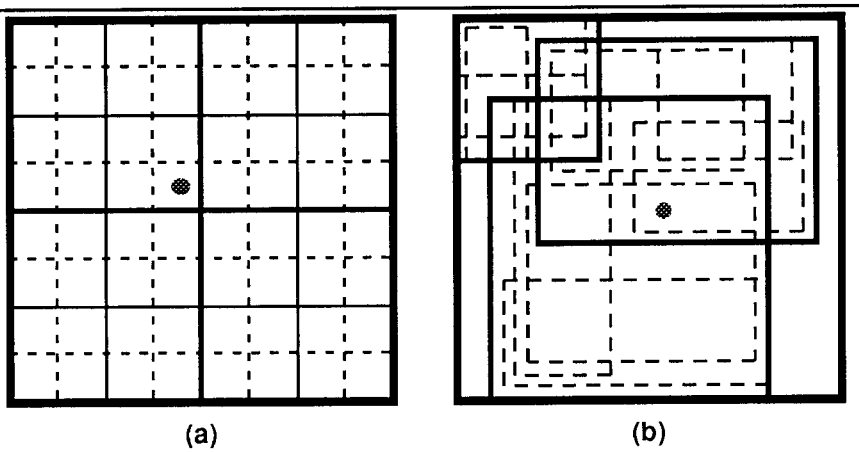

Figure 7 — 2D Scene Hierarchies

Suppose we give *PT2D* the points (small darkened circles) shown in Figure 7. When the algorithm traverses a bound containing the point in hierarchy (a), the algorithm will continue traversal with only one child of the bound. However, when *PT2D* traverses a given bound in hierarchy (b), the algorithm may have to continue traversal with several children of the bound. This difference makes *PT2D* $O(\log P)$ for hierarchy (a), and $O(f(P))$ for (b), where P is the number of objects in the scene and $f(P)$ is a non-logarithmic function of P — $\sqrt{P}$, or perhaps P itself.

Now let us consider another search algorithm *RAY2D* that casts a 2D ray into the scene and attempts to find objects that intersect the ray. Generally, for the hierarchies in Figure 7, *RAY2D* will have to traverse more than one child of each bounding region. This happens because the scenes are not partitioned in the *space* of the "key" that *RAY2D* uses. The "key" that *PT2D* uses is a (x,y) coordinate in the space $\mathbf{R}^2$; the hierarchy in Figure 7a is partitioned with respect to this space, so *PT2D* has complexity $O(\log P)$ with this hierarchy. However, the "key" that *RAY2D* uses is a ray in a 3D space (two dimensions for ray origin, and one for direction), and the hierarchies in Figure 7 are not partitioned with respect to this space.

Arvo and Kirk address this problem with a unique partitioning scheme [ArKi87]. For a full-fledged 3D ray tracer they define a ray space in $\mathbf{R}^5$ (three dimensions for ray origin, and two for direction). They then partition the 3D scene into a hierarchy of regions in this space; a ray tracer can traverse this hierarchy much like *FIND* and *PT2D* above. They make no assertions about the asymptotic complexity of their renderer, but it potentially might have $O(\log P)$ worst case cost.

Unfortunately, Arvo and Kirk's *5D ray tracing* is complicated and non-intuitive, so it has not been widely used. Also, the implementation they describe in [ArKi87] does not allow for $O(\log P)$ performance: Since 5D ray space can be extremely large, they do not construct the hierarchy before rendering begins. Instead, their implementation builds the hierarchy in the regions of 5D space where the renderer actually spawns rays. This scheme of *lazy evaluation* essentially embeds a sorting process — traditionally $O(P \log P)$ — into the renderer.

Noise from Stochastic Opacity. The use of stochastic opacity introduces noise into the image. To maintain image stability despite the noise, the renderer must perform extra sampling of the scene, which effectively raises the cost of stochastic opacity. However, a scale-sensitive renderer using stochastic opacity should still have better performance than a traditional renderer.

Consider a ray that has a given direction and touches an amalgam. With stochastic opacity, the amalgam has the same shading attributes regardless of where the ray intersects it. However, a traditional bounding volume can have widely varying attributes at different intersection points — the ray can intersect completely different objects inside the bound. In the image signal on the image plane, the amalgam has a lower variance than the corresponding bounding volume, so the amalgam requires less sampling than the bounding volume.

Conclusions

Asymptotic time costs of ray tracing with amalgams compare favorably with the costs of traditional ray tracing techniques. Expected asymptotic costs of the scale-sensitive and traditional techniques are the same, but under many circumstances the scale-sensitive renderer should be faster (in an absolute sense). Also, under some circumstances, we may consider the scale-sensitive technique to have better worst case asymptotic costs than traditional techniques. In any case, the scale-sensitive technique is no slower.

We can even say that scale-sensitive ray tracing has better worst case asymptotic time costs than conventional *scanline* rendering techniques. Equation 3 indicates that the

rendering time for the scale-sensitive ray tracer can be independent of the number p of primitives in a scene, while conventional scanline techniques are at best linear with p. (Of course, scanline renderers that use scale-sensitive techniques can achieve the same asymptotic speedups as ray tracers that use amalgams.)

References

[ArKi87] Arvo, James, and Kirk, David, "Fast Ray Tracing by Ray Classification," *Computer Graphics*, **21**, 4, July 1987, p. 55-64.

[Glas84] Glassner, Andrew S., "Space Subdivision for Fast Ray Tracing," *IEEE CG&A*, **4**, 10, October 1984, p. 15-22.

[Glas89] Glassner, Andrew S., editor, *An Introduction to Ray Tracing*, Academic Press, 1989.

[GoSa87] Goldsmith, Jeffrey and Salmon, John, "Automatic Creation of Object Hierarchies for Ray Tracing," *IEEE CG&A*, **7**, 5, May 1987, p. 14-20.

[Kay86] Kay, Timothy L. and Kajiya, James T., "Ray Tracing Complex Scenes," *Computer Graphics*, **20**, 4, August 1986, p. 269-78.

[NaTh86] Naylor, Bruce F., and Thibault, William C. "An Application of BSP Trees to Ray Tracing and CSG Evaluation." Technical Report GIT-ICS-86/03, School of Information and Computer Science, Georgia Institute of Technology, February 1986.

[Whit80] Whitted, Turner J., "An Improved Illumination Model for Shaded Display," *Comm. ACM*, **23**, 6, June 1980, p. 343-9.