Shaft Culling for Efficient Ray-Cast Radiosity

Eric A. Haines & John R. Wallace
3D/Eye Inc.
2359 North Triphammer Road
Ithaca, NY 14850
USA

email: erich@eye.com, johnw@eye.com

Abstract

In radiosity algorithms, much time is spent computing visibility between two surfaces. One approach to approximating this visibility is to use ray casting methods. A new algorithm is presented which takes advantage of object coherency when using ray casting for radiosity. An efficient method is presented to form a volume between the emitter and receiver, and then generate a candidate list of items partially or wholly within the volume. Using this list, ray casting is performed to determine the amount of visibility between surfaces. Statistics are presented showing the decrease in overall computation time compared to a traditional ray casting technique.

Introduction

Within radiosity implementations the most time consuming procedure is determining the visibility between surfaces. Determining the amount of light that one surface transfers to another is made difficult by other objects which can obscure some of this light. An exact solution of this problem is usually not attempted; rather, some point sampling of the two surfaces is performed and the visibility approximated.

In [Wallace et al., 1989] the authors presented a procedure of ray casting to perform progressive radiosity computations. Ray casting between surface vertices and samples on the emitter was used to determine visibility. The number of sample points on the emitter could vary (i.e. be anything between 1 and 25, typically 4), and the surfaces could be tessellated to generate more samples. The amount of shadowing for each sample point on each surface was computed using a ray caster with a hierarchical bounding volume efficiency scheme.

Ray tracing efficiency schemes have exploited a number of forms of coherence to decrease the amount of time spent testing rays against the environment. One set of rays which have a large degree of coherence are those starting at the eye; such rays have a common origin and a predictable set of directions. This coherence has been used in a variety

of ways, such as the item buffer [Weghorst et al., 1984] and hybrid hidden-surface/ray-tracing algorithms [Nakamae et al., 1989; Salesin & Stolfi, 1989]. However, not as much is known in advance about all other rays, i.e. those for shadow testing, reflection, and refraction.

In shadow testing we know the rays will end at some light source. This fact has been used in efficiency schemes such as the light buffer [Haines & Greenberg, 1986]. In the light buffer algorithm, a preprocess computes lists of candidates for all directions from the light. The appropriate list is retrieved for a given direction from the light and used for ray tracing. The lists are much shorter to test than the full environment. However, because there is no knowledge of where the shadow test rays originate, much effort is wasted in creating visibility lists for volumes of space which are never accessed. Also, the light buffer has a problem of creating candidate lists at a uniform resolution, that is, space is partitioned into volumes of approximately the same size. In practice there are volumes with a large degree of complexity that could use further subdivision, while other volumes contain few objects. The light buffer suffers from not knowing in advance which directions rays will come towards each light source.

Ray classification [Arvo & Kirk 1987] uses 5D volumes to exploit ray location and direction coherency. However, when forming a 5D volume it is not known how many times this volume will be accessed, so a caching scheme is used to save candidate sets. One problem is that the cost of building the volume may not be offset by the savings from using it. Caching schemes usually result in the candidate list for a particular volume being regenerated a number of times.

In [Marks et al., 1990] are explorations of ray casting efficiency for radiosity visibility computations. For environments made entirely of quadrilaterals, they create a decahedral volume between the two test patches. If no objects are found to be inside this volume, the patches must be fully visible; otherwise, the larger patch is subdivided via a quadtree and the intervisibility between sub-patches is checked. At some level standard ray casting would finally be performed as needed. They found that this technique provided acceleration for scenes with much intervisibility coherence, but in others cost a little more time.

In this paper we present a method of ray casting that takes advantage of similar forms of coherence. One source of coherence is that in radiosity algorithms a single object (e.g. a light emitter, such as a fluorescent ceiling panel) is tested for visibility with a large number of other objects. Note that this is true for both progressive radiosity [Cohen et al., 1988] and full matrix solution radiosity [Cohen & Greenberg, 1985]. Another form of coherence is that rays between two objects are limited to a particular volume of space. We also know in advance approximately how many rays will be cast between these two objects. These properties of radiosity algorithms provide considerable opportunities for increasing efficiency.

Algorithm

We define an *object* to be some single renderable primitive, e.g. a polygon, a torus, etc. An *item* is defined as an object or a bounding volume containing one or more other items. Some examples of items include: a spline surface, a bounding sphere containing three objects, and a root bounding box of a bounding volume hierarchy.

At the start we are given a set of objects, and have built a bounding volume hierarchy by

some method [Goldsmith & Salmon, 1987; Kay & Kajiya, 1986]. We wish to find some set of items which can potentially block the visibility between the emitter and the receiver (we call these two items *reference items*). This set of items will be called the *candidate list*. Each time we wish to determine the visibility of some sample point to some other sample point, we use a ray caster to test all the items on the candidate list until an intersection is found or the list is exhausted.

One trivial solution would be to form a candidate list of all objects in the scene. This is equivalent to what a ray tracer without any efficiency scheme does - all the objects are tested against the shadow ray. Another trivial solution would be to give the root node of the bounding volume hierarchy as the only candidate on the list. Standard traversal of the bounding volume hierarchy then ensues for shadow testing.

Imagine we define a volume in space which contains the two reference items and all points between them. That is, any ray starting on one reference item and ending at the other is fully contained inside this volume we have defined. We can form a better candidate list by comparing this volume with the various items in the hierarchy; if an item is outside of the volume, it cannot be hit by any rays between the two reference items and so cannot occlude visibility between them. In fact, we could find that there are no objects at all between the reference items, in which case we know that the two reference items are fully visible to one another without any ray casting being necessary.

An important consideration is the amount of time spent creating this candidate list versus the time saved by using it. In radiosity algorithms we typically will sample each emitter a number of places on its surface, and each object receiving energy will typically have a set of points (e.g. a mesh) on its surface from which we wish to determine the emitter's visibility. For example, if we sample our light source 4 times per test point, and have an object with a 5 x 5 grid of test points, there are potentially 4 x 5 x 5 rays we need to cast between the surface and the light. The fact that we can know in advance about how many tests we will be doing means that we can much better approximate how much time to put into forming a candidate list.

This fact points toward a simple test to determine whether to create a candidate list. First, if either or both items are polygons, culling tests can be performed: if the emitter faces away from the receiver, or vice versa, then no energy is transferred. Else, find the approximate number of potential rays between the two reference items. If it is greater than some preset value, then form the candidate list and use this. If less, then don't bother, as the time saved per ray may not add up to be more than the time lost forming the candidate list. This value is best determined empirically for the system used; we think 10 rays is a reasonable break-even point. In fact, in most cases we test four points on the emitter against each object, so even an untessellated triangular receiver will generate 4 x 3 potential rays. This means that in almost all cases we will want to form a candidate list, so can dispense with this test altogether since we will almost always pass it.

The candidate list algorithm itself consists of three parts: forming the testing volume, creating the candidate list by using this volume, and accessing the candidate list for visibility determination between samples.

Forming the Testing Structure

The testing structure is formed by doing the following:

1. Obtain the bounding boxes for the reference items.
2. Compute the extent bounding box containing both reference items.
3. Create the plane set between the two reference items' boxes.

The first step is to form a bounding box for each reference item. Since we are using a bounding volume hierarchy, these boxes will be available or easily derivable. Each box is axis aligned and so consists of a minimum and maximum corner. These will be referred to by *lo.x*, *lo.y*, *lo.z* for the minimum corner, *hi.x*, *hi.y*, *hi.z* for the maximum.

In the second step an *extent box* is formed which contains both reference items. The extent box is formed by comparing the low and high corners of the reference boxes and taking the minimum and maximum, respectively. In this step we also wish to identify all edges of this box which are not a part of either reference box; call these *culled* edges.

The third step is to form a set of planes which connect the two reference boxes. The idea is to connect the edges of the reference boxes so that a minimal bounding volume is formed. Another way of thinking of this process is that each plane added whittles away the space between the reference boxes and one of the culled extent box edges identified in step two. This volume will be referred to as a *shaft*. These three steps are shown in Figure 1.

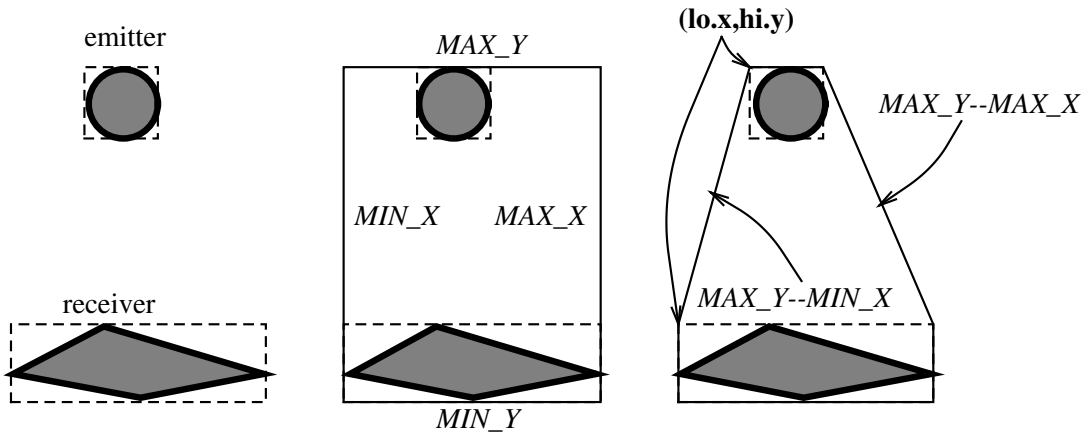


Figure 1. Shaft Formation Process

When forming the extent box we also keep track of which coordinates of which box become the minimum or maximum. These lists will be used to identify culled edges. Two lists, one for each reference item, are needed. Each list can have a maximum of six entries,

and each entry records a coordinate type (minimum or maximum) and direction (*X*, *Y*, or *Z*), e.g. *MIN_Z*. The minimums of each reference box are compared: whichever is smaller is noted on the list for that reference item. If the minimums are exactly equal, nothing is put on either list. Likewise, the maximums are compared and the higher coordinate type is stored.

We will walk through what lists are formed for Figure 1. We first compare the lower extents (i.e. *lo.x* and *lo.y*) of the emitter and receiver. The receiver's lower extent is less than the emitter's in both cases, so the receiver list has *MIN_X* and *MIN_Y* placed on it. The higher extents of the two boxes are now compared, and since the emitter's *hi.y* is greater than the receiver's, *MAX_Y* is placed on the emitter's list. Similarly, *MAX_X* is added to the receiver's list. The *Z* values are not shown for this 2D example, but would normally be tested and listed.

The third step is to generate all the planes which connect the edges of the two reference boxes. This turns out to be analogous to forming all combinations of elements on one reference list with those on the other, discarding combinations with matching directions. Each combination of elements corresponds to a plane connecting the edges of the reference boxes.

For example, in Figure 1 the *MAX_Y* of the emitter can be combined with the *MIN_X*, *MIN_Y*, and *MAX_X* of the receiver. *MAX_Y--MIN_Y* is discarded as a combination since the directions match. *MAX_Y--MIN_X* means that we need to form a plane between the edge on each box which has *hi.y* and *lo.x* constant for that box. In other words, two coordinates of the edge are fixed, and so the third, unmentioned, direction (in this case, *Z*) varies. Explicitly, we wish to connect the edges formed by the two vertices (*lo.x,hi.y,lo.z*) and (*lo.x,hi.y,hi.z*) on each reference box with a plane. Another way to think of this plane is that it cuts away the space in the extent box associated with the culled edge *MAX_Y--MIN_X*.

Forming such a plane is straightforward. The plane's normal has three coordinates. The "unmentioned" coordinate is always 0. The other two coordinates of the normal consist of the differences between the other axis' specified coordinates for each reference box. Carrying on with our example of *MAX_Y--MIN_X* (which means that the emitter's *hi.y* value is the maximum *Y* of the extent box, paired with the receiver's *lo.x* value being the minimum *X* of the extent box), the plane in this case is:

X.direction = Receiver.hi.y - Emitter.hi.y
Y.direction = Emitter.lo.x - Receiver.lo.x
Z.direction = 0

Each pair of extents has its own set of equations and its own culled edge associated with it. Care should be taken so that each normal formed faces outward from the shaft.

One last coordinate is needed for the plane formed: a relative distance from the origin. This is the negative of the dot product of the direction vector just computed and any point on either edge forming the plane. Knowing in advance that one of the plane's coordinates is always 0, this calculation can be streamlined to two multiplies and an addition. Our Figure 1 example yields:

P.distance = -(X.direction * lo.x + Y.direction * hi.y)

using the *lo.x* and *hi.y* of either reference box.

Each plane formed is stored in a list, which is called the *plane set*. Note that it is possible for no planes to be formed. For example, this occurs when one reference box is entirely inside the other box. A maximum of eight planes can be formed between any two boxes; see Figure 2 for some configurations where eight planes are formed between two reference boxes.

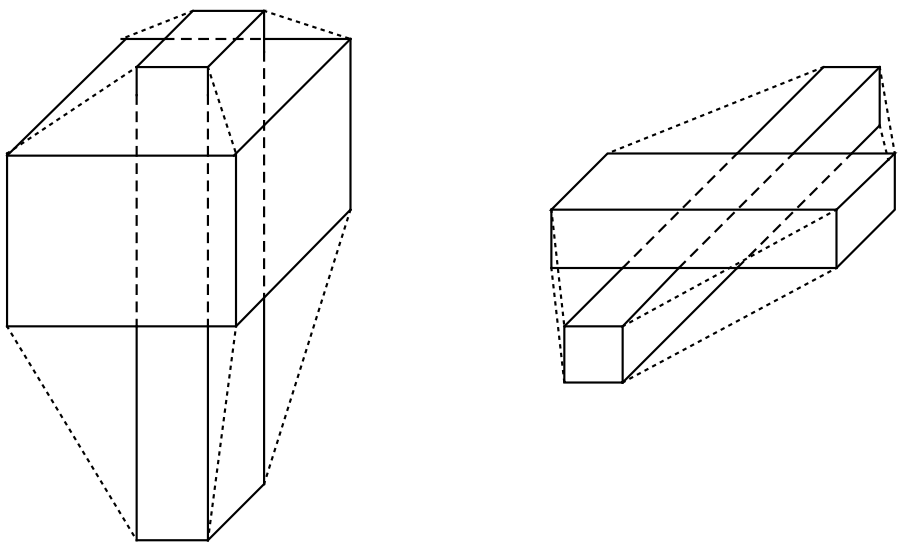


Figure 2. Eight Plane Shafts

Creating the Candidate List

Now that the extent box and plane set have formed a shaft, we will use them to determine if items overlap it. To begin, the root item of the bounding volume hierarchy is compared to the shaft. Three possibilities can occur:

1. The item is entirely outside of the shaft.
2. The item is entirely inside the shaft.
3. Else, the item overlaps the shaft.

If the item is entirely outside of the shaft, then the item cannot obscure the visibility of the reference items and so can be ignored.

If entirely inside the shaft or overlapping, one strategy, called "always open", is to always open the item if it is a bounding volume and then recursively test its children against the shaft. The candidate list will then consist entirely of objects (no bounding volumes) inside or overlapping the shaft at the end of testing.

For all other strategies, if the item is fully inside the shaft, it is put on the candidate list. Bounding volumes save time in ray tracing whenever they are *missed* by rays; such a ray/bounding volume test is then cheaper than all the items inside the bounding volume being tested and missed. The philosophy here is that a bounding volume fully inside the shaft is more likely to be missed than an overlapping bounding volume, and so more likely to save time.

The overlap case is where things become interesting. If the overlapping item is an object, then it is put on the candidate list since it can obscure visibility. Many strategies are possible for dealing with an overlapping bounding volume. One is to always put these bounding volumes on the candidate list, on the theory that we want to minimize time spent forming the list, and the bounding volume is meant to help ray casting efficiency. In this case, overlapping items are treated the same as those inside the shaft. This is called the "keep closed" strategy.

The converse, called "overlap open", is to always open up overlapping bounding volumes and test all the children against the shaft, recursively. The idea here is that there are generally only a few objects between any two reference items, so opening the boxes now will save many intersection tests of them during ray casting.

However, some time can be wasted opening bounding boxes when unnecessary. For example, imagine a bounding box has four objects in it, three of which are barely in the shaft. Opening the box culls out the fourth object not in the shaft, but forces us to test the other three objects against each ray. If we had left the box intact, and the box overlapped the shaft only a little bit, each ray would be tested (and would most likely miss) against the box, thereby saving us testing the three objects.

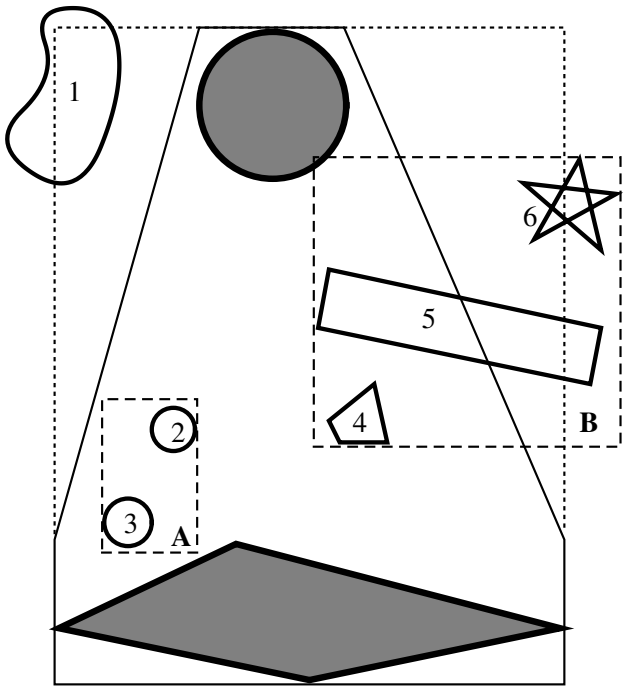
The "ratio open" strategy addresses this situation by providing a ratio and testing to see if it is worth opening a bounding box. For example, we might give the ratio 0.4. If more than 40% of the items in a bounding box are found to be in or overlap the shaft, don't open the bounding box; rather, put it on the candidate list. It should be noted that, regardless of the ratio, if only one child of the bounding volume overlaps the shaft, then the bounding volume should probably be discarded. About the only reason to keep such a bounding volume around is if the child item is much more difficult to test for a miss than the bounding volume.

In Figure 3 a set of six objects (1-6) and two bounding volumes (A and B) are compared using the various strategies, with the candidate lists shown for each. Object 1 is eliminated by all strategies. Bounding box *A* is opened by the "always open" strategy, otherwise left closed. Bounding box *B* depends on the strategy; for example, the number of objects within or overlapping the shaft in the "ratio open" strategy is greater than 40%, so the bounding box is not opened.

Further enhancements of these strategies are presented later. The performance of the various strategies is explored in the Statistics section.

Shaft Cull Testing

Regardless of which strategy is used for traversing the hierarchy and forming the candidate list, the algorithm which remains the same is the comparison of a shaft and a bounding volume. This test is called the *cull test*, and the bounding volume tested is called the *test*



Always Open
Object 2
Object 3
Object 4
Object 5

Keep Closed
Bounding Box A
Bounding Box B

Overlap Open
Bounding Box A
Object 4
Object 5

Ratio Open (0.4)
Bounding Box A
Bounding Box B

Figure 3. Shaft Strategies

volume. The two bounding volume types we will consider are the sphere and the axis aligned box. The process for the sphere is presented first, and then a referencing scheme is described which makes the box about as fast to test.

The steps in the test are as follows:

1. See if the test volume is inside, outside, or overlapping the extent box.
2. If outside the extent box, testing is done. Else, check if the test volume overlaps either reference item.
3. If it overlaps, testing is done. Else, compare the test volume to the plane set.
4. If outside the plane set, testing is done. Else (depending on the strategy), either test if the item is fully inside the shaft or perform the "always open" strategy.

The first step is a simple comparison of the extent box with the test sphere. We use Arvo's algorithm for box/sphere comparison [Arvo, 90]. In this test each coordinate axis is treated separately. If the sphere's center coordinate for an axis is outside the box's extent along that axis, then take the difference between the sphere's coordinate and the closer extent value and square it. Sum and check these squared values (if any) for the axes; if greater than the radius squared, then the sphere is outside the box.

In the second step we do a similar test as the first step for the sphere and each of the reference item boxes. If the test sphere overlaps either reference item, then it must overlap

the shaft and testing is done.

Testing the sphere against the plane set is straightforward. To simplify computations, each plane equation's normal should be normalized and the plane distance value scaled accordingly. The distance from the sphere's center to the plane is defined as:

T = dot_product(Plane.normal, Sphere.center) + Plane.distance

For each plane, the distance of the sphere's center from the plane is compared to the sphere's radius:

If T > Sphere.radius, then the sphere is outside the plane

If the sphere is found to be outside of any plane, then it must be outside the shaft. If the sphere survives all these tests, it is probably overlapping or inside the shaft. Note that a sphere can overlap the extent box and one or more planes and yet still be outside of the shaft. We call these *false positives*, as they are falsely identified as overlapping the shaft. Such false categorization does not causes any errors but can cost speed, as some bounding spheres which could actually be ignored may instead be placed on the candidate list and wastefully tested against rays. If a shaft is being formed for a particularly large set of rays, it may be worth fully checking for these false positives. Figure 4 shows a portion of a bounding sphere which is a false positive.

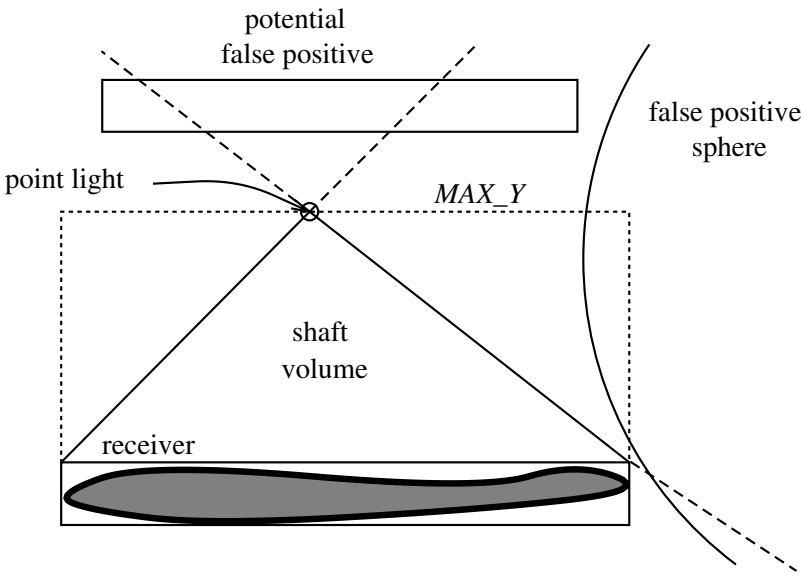


Figure 4. False Positives

Depending on the strategy, we may want to determine whether the sphere is fully inside the shaft. If any of the sphere's extents are outside the shaft's extent box, then the sphere cannot be fully inside the shaft. Otherwise, the sphere is again tested against the plane set. The test is similar to the above:

If T > -Sphere.radius, then the sphere is fully inside the plane

Note that the comparison computed is simply 2 * Sphere.radius more than that calculated for the outside test, so the outside test's results can be used to quickly test this condition. If the sphere is found to be inside all planes in the plane set, then it is fully inside the shaft; else, as soon as it fails to be fully inside any plane, it is categorized as overlapping the shaft.

Boxes can be tested in a similar fashion to spheres. We do not need to normalize the plane set normals if only boxes are tested against them. In the sphere test the radius was used for comparison, so normalization was important. In the box test we will find the distance between various corner points and each plane and test the sign of these distances, so the scale of the distance is irrelevant.

An insight into the nature of testing any box against a plane is important in understanding the box and shaft test. First, we define the distance of a point from the plane as:

T = dot_product(Plane.normal, Point) + Plane.distance

The plane defines a half space. Similar to the sphere test, if T > 0 then the point is outside the half-space, if T = 0 it is on the plane itself, and T < 0 means inside. When referring to the distance from the half-space, note that we mean this signed distance, i.e. a point with T = 2 is termed to be farther away from the half-space than a point with T = -9.3. Points inside the half-space are considered to be closer to it.

Given a particular plane, there is at least one corner which is always going to be the farthest away from the half-space, and at least one that is always nearest. The farthest corner is a function of the normal of the plane: if the sign of a particular coordinate is negative, use the *lo* value of the box, if the sign is positive use the *hi*, if 0 either value can be used. The nearest corner is just the opposite of these conditions: *lo* for positive, etc.

For example, given a plane with a normal [-3 8 4], the farthest box corner is always [*lo.x*,*hi.y*,*hi.z*] and the nearest always [*hi.x*,*lo.y*,*lo.z*]. Each plane points into one octant of the *XYZ* reference frame, which corresponds to one corner of all boxes tested against it. A fast way to obtain the box's relevant corner during testing is to store a little extra data for each plane as it is formed. Store the offsets to the coordinates needed for the far corner and near corner with each plane, e.g. if *lo.x* = 0, *lo.y* = 1, etc, then the farthest box corner's offsets in our example would be [0,4,5] and the nearest [3,1,2]. Note that near corner offsets equal ((far offsets + 3) modulo 6). These offsets can then be used directly to obtain the desired corner, as opposed to wasting time with comparison tests to access it.

Using this observation, the outside/inside/overlap test for boxes is done similarly to the sphere test. First check the test box against the extent box: if there is no overlap, the box is definitely outside and testing ends. Else check the test box against the reference item boxes: if there is any overlap, the test box must overlap the shaft and testing is done. Else test each plane in the plane set against the box's nearest corner for that plane:

```
If dot_product( Plane.normal, Box.near_corner ) + Plane.distance > 0,
    box is outside the plane (and the shaft)
```

If the box survives all of these tests, then it must overlap or be inside the shaft. In Figure 3 object 1 overlaps the extent box, but is outside the *MAX_Y--MIN_X* plane so it is discarded. Bounding boxes *A* and *B* survive the plane set test.

From our earlier extent box testing we know if the test box was fully inside the extent box; if it is not then it must overlap the shaft. For example, in Figure 3 bounding box *B* is not fully in the extent box, so we know that it must overlap the shaft. Else, if needed, we can find the exact state of the box by testing planes in the plane set against the corresponding farthest corners:

```
If dot_product( Plane.normal, Box.far_corner ) + Plane.distance > 0,
    box overlaps shaft
```

If the box does not overlap any plane in the set, then it is fully inside the shaft. Bounding box *A* in Figure 3 needs to be tested in this fashion, as it was fully inside the extent box but it was not known if it was fully inside the plane set.

The box test uses a small number of operations, which is made possible by accessing only the relevant corner for testing. The traditional method for testing overlap is to perform a full culling test of the box against shaft's extent box and plane set and see if anything was left inside, which is much more than is needed to classify the test box. Using interval analysis a faster classification test can be devised, but this type of testing is conservative and inexact: such an algorithm errs on the side of classifying some boxes which are outside as being overlapping, for example.

Candidate List Access

For each combination of sample points on the receiver's surface and on the emitter's surface a ray is cast, if both surfaces face each other. The ray is tested against the candidate list, with bounding volumes hit being opened and their children tested. As soon as an opaque object is hit the shadow testing is over, with the receiver's point in shadow.

One implementation of this process is to copy the candidate list to a stack. Pop an item and test it for intersection, and push the children of any bounding volume intersected on the stack. A hit or an empty stack ends testing.

Enhancements

In this specific use of shafts to test rays traveling from an emitter to a receiver there are a number of important enhancements we can make to the basic shaft testing algorithm. We know that rays cast between these two reference items must originate in one box and end in the other. Therefore, if our test volume fully encompasses either reference box, then the item must *always* be tested for intersection by any ray cast. So, when testing overlap with the

extent box, we can also check if the test volume encompasses either reference box. If so, and the item is a bounding volume, we should then always open the volume and test its children, since we know the bounding volume will always be hit. If the item is an object it should be placed on the candidate list. Box/box comparison is just extent overlap testing. Sphere/box comparison can be done by using Arvo's method [Arvo, 1990] Essentially, sum the squares of the distances between each axes' sphere center coordinate and most distant box extent coordinate; if less than the sphere's radius squared, the box is fully inside the sphere.

The realization that all volumes which contain a reference box must be hit, combined with the fact that in radiosity we compute the effect of one emitter against a large number of receivers, leads to another useful preprocess. In fact, this preprocess can be used to prune the hierarchy of useless bounding volumes and so increase the efficiency of a traditional ray caster. When we first begin testing the emitter with the set of receivers, we can traverse the bounding volume hierarchy once and find a candidate list (called the *emitter list*) of those items which do not enclose the emitter's box. We then can use this list when testing the hierarchy against any shaft, since all shafts for this step have the emitter as a reference box. Instead of starting at the root node and testing the whole hierarchy against each new shaft (and finding that the same items enclose the emitter's box and so must be opened again and again), we now test items on the emitter list against the shaft. For each shaft, only the receiver's box then needs to be tested for enclosure within the test item, versus the earlier method in which both reference items were so tested.

One other advantage to building an emitter list is that all the items placed on it can be categorized as either overlapping or outside the emitter. When it comes time to check items on this list against the shaft, we know that an item which overlaps the emitter cannot be categorized as outside.

One efficient way to form the emitter list is to start at the emitter and travel up the hierarchy. The parent of the emitter must contain the emitter, so can be opened. The other children of the parent are then tested to see if they enclose the emitter, and if not are placed on the emitter list, else they are opened in turn and their children tested. When all of the parent's other children have been categorized, we move to the parent of this parent (which again must contain the emitter) and repeat the process, ending when the root item's children are categorized. Note: the emitter itself may or may not be placed on the emitter list, depending on whether it can occlude itself.

If the emitter casts energy in a limited set of directions (e.g. a one-sided polygonal light will not illuminate anything behind it), then all items behind it can be eliminated once at the beginning of the step. Another technique is useful if surfaces have only one side, but none of these non-existent backfaces are visible to any frontface. For example, in a solids modelling system the insides (backfaces) of the objects can not be seen and so do not have to be rendered. In such cases, the light reference object can be used to cull away all polygons which face away from the light. This can be done by the same methods used to compare a plane and a bounding volume (the emitter, in this case).

Another idea is to mark within the hierarchy when various sons are outside the shaft. When creating the candidate list, the "ratio open" strategy finds which children of a bounding volume are outside the shaft. If the bounding box is put on the candidate list, mark the children of this box which are outside the shaft by using a mailbox method [Arnaldi et al., 1987], with a new identifier for each new shaft. Now when the ray caster is called, it also

checks if a child is marked as outside; if so, the child is not tested for intersection.

One problem with using the receiver's bounding box is that this volume often slightly overlaps adjacent polygons which do not have any possibility of shadowing the receiver. Using a small tolerance to cull out such polygons can eliminate a few of these from the candidate list (though the shaft will still catch a few). If such adjacent, non-occluding polygons can be identified by other means, they can all be eliminated from testing.

An enhancement which is worth using is shadow coherency testing [Haines & Greenberg, 1986]. This method keeps track of what object (if any) the previous ray hit, and tests the current ray against this object. The theory is that whatever shadowed the last ray will probably shadow the current ray, since the current ray is usually similar to the previous ray. This technique is useful for all ray tracers and casters, and the shaft tester is no exception. In fact, when a new receiver is tested, we should not build a shaft between the reference items until the shadow coherency object is missed. We may find that the previous shadowing object occludes all rays between the new receiver and the light, too.

Other Uses

Though presented as an algorithm for speeding ray casting testing between two surfaces, the shaft culling technique is applicable to other computer graphics problems. For example, it is a fast way of determining the candidates in a 5D classification scheme ray tracer [Arvo & Kirk, 1987]. In their method, all vertices of the bounding polyhedron for an object are tested against each plane. The shaft method is much faster than their scheme when testing boxes for inclusion, as only one corner of any box needs to be tested against each plane.

We also find shaft culling to be a useful tool in minimizing the work done by other algorithms. For example, if we want to perform an analytic solution for point to patch visibility, shaft culling gives us a list of candidates to send to the analytic algorithm.

Statistics

Each object in the scene has its own bounding box, and the bounding boxes were clustered using Goldsmith and Salmon's hierarchy building algorithm, as it was found to yield faster intersection timings than Kay and Kajiya's scheme. The regular ray caster has been optimized over a number of years, so provided a tough standard of comparison.

A simple environment was used to compare the performances of the various strategies. The office scene in [Wallace et al., 1989] was used in two different forms: standard and premeshed. The standard scene was meshed by the radiosity software, then shafts were formed using the original polygons. This means that shafts generated for the walls and floor have a large number of rays generated (due to the internal mesh points in them), but the shafts are proportionally wider and so include more candidates for testing. The premeshed database was identical, except that the floor and walls were tessellated into polygons before input. This causes more shafts to be formed, with less rays per shaft, but makes for tighter shaft for those rays. Both scenes have 10 area light sources. The standard database has 906 polygons before meshing, the premeshed scene has 2563 polygons before further meshing. These two

versions of the same scene were used to examine the effect of shaft width on the various strategies.

The code to test shafts had various inefficiencies due to the nature of the calling routines. For example, shafts were formed using the entire emitter, though the samples on the emitter could have been bounded by a smaller bounding box. Shafts were always formed, even if the shaft had only one uncached ray left to shoot.

The first testing done was to see the effect of shadow coherency, a proven algorithmic improvement. The standard and premeshed scenes have similar results. In the simplest test case of one sample per emitter and 20 steps of progressive radiosity, about 54% of the vertices were shadowed. Of these shadowed vertices, about 78% were found occluded by the shadow cache; that is, whatever had previously shadowed the last ray also shadowed this one. The total number of intersection tests for the standard scene was 3262K intersection tests of bounding boxes and polygons without and 2252K tests with shadow caching, a 31% decrease.

We then tested the effect of forming a candidate list for the emitter at the beginning of each step of progressive radiosity. We also culled away any bounding boxes which were entirely behind the area light source. This candidate list was then used for regular ray casting with shadow caching. For the simplest test case this provided a 17% decrease in number of intersection tests for the standard scene, 12% for the premeshed scene. Because the cost of this preprocess was so minimal and always saved time overall, it and shadow caching are used in all the tests that follow. The baseline ray caster used for comparison also uses both of these algorithmic improvements.

Two sets of tests were done on each of the databases for each strategy. We varied the number of samples on each emitter for these tests to see the effect on overall performance. Shadow caching quickly found the visibility of some of the rays cast. Since we are trying to understand the effect of the shaft, we did not include the rays which were shadow cached (each ray has one intersection) in the statistics. In the standard scene, this meant averages of 8.1 and 34.0 uncached rays per shaft; the premeshed scene has less samples per polygon on the average, so averages were lower at 3.9 and 18.5 rays/shaft.

Statistics for the low and high sampling rates for each scene and strategy are shown in Table 1. Tests were done on a Hewlett-Packard Series 9000 model 433 workstation with unoptimized, debuggable, profiled code. The "tests/ray" is the total of bounding box and object tests made per ray: recall that each object has its own bounding box. The times are the average costs in microseconds for shooting an uncached ray, with the cost of building and testing the shaft included. In general, the number of intersection tests per ray were similar at the two sampling rates, and so are presented as an average of these two.

TABLE 1. Statistics for Office Databases

Strategy	standard			premesh		
	tests/ray	low time	high time	test/ray	low time	high time
baseline	44.6	787	762	79.2	1487	1355
keep closed	28.4	642	534	47.8	1094	919
always open	132.9	2268	1785	27.5	1289	694
overlap open	66.7	1382	1008	25.8	1283	680
ratio open 0.2	22.1	670	482	27.1	1104	685
ratio open 0.4	21.9	681	480	26.7	1135	682
ratio open 0.6	22.1	679	478	25.0	1154	654
ratio open 0.8	24.1	753	524	21.1	1327	631

Oddly enough, the percentage of rays which hit occluders was lower at the higher sampling rate. This also had the peculiar effect of lowering the time cost per ray for the higher sampling rate by 3-9% for the baseline ray caster.

Which beam culling strategy to use is an interesting question, as there is no strategy which has dominance over the others. The "keep closed" strategy minimizes the time spent doing the shaft culling preprocess, as most items are quickly classified as outside the shaft or overlapping. Shaft culling eliminates those items which definitely have no chance of being hit, saving the costs of intersecting (and missing) these items repeatedly by rays in the shaft. Because this strategy does spend such a small amount of time building the candidate list and is so conservative (bounding volumes are not opened unless definitely hit), it always yields some speed up. Time saved ranged from 21% to 32%.

The "always open" strategy failed for the standard office scene and the low sampling rate premesh, but performed quite well for the high sampling rate premesh scene. We believe this is because of two factors. In the standard scene, the wall and floor shafts have many samples and shafts formed using them encompass much of the office. This strategy removes all the bounding boxes for items in these shafts. Many of these bounding boxes would have been more profitable to keep around and use for the standard scene. This leads to excessive intersection rates for the standard scene. The premesh scene has shafts with smaller volumes, and so less likelihood that bounding boxes would help much.

Opening all these boxes during the preprocess also costs time when forming the shaft's candidate list. For the premesh scene, it is not until this cost is amortized over a large number of rays that the time per ray drops to acceptable levels. However, when the shafts are mostly narrow (as in the premesh scene) this strategy can yield greater savings than the "keep closed" strategy.

The "overlap open" strategy is not much different than "always open". The preprocess cost was higher due to having to do extra tests, and some savings were garnered by keeping around bounding volumes which were fully in the shaft. Looking more closely at the performance of this strategy, we found that there were very few boxes with more than one item in them which were fully inside shafts: 0.71 boxes per standard scene shaft and 0.36 boxes per premeshed shaft. Such complex boxes contained an average of 33.1 objects as descendants for the standard scene and only 4.3 children for the premeshed scene. Such boxes, then, proved useful in improving standard scene rates, but were of little use in the premesh scene.

The "ratio open" strategy gave the best results in terms of number of intersection tests. At

the best ratio, they cut the number of intersection tests performed by a factor of 2.0 for the standard scene and 3.8 for the premeshed scene. With the preprocess time included, speedup ratios of 1.6 and 2.2 were obtained for the maximum sampling rates. Note that for the standard scene in particular the lower ratios caused less boxes to be opened, so that less time was spent on shaft culling itself. Even though this led to slightly higher numbers of tests per ray, the overall cost was lower.

It is interesting to examine how the ratio is related to the database type. A low ratio corresponds to keeping bounding volumes closed, while a high ratio opens them more frequently. With the large, wide shafts in the standard scene, it seems to be profitable to keep bounding volumes closed, while the premesh scene's narrow, tight shafts benefit little from bounding boxes. However, the 0.8 ratio does perform better than the 1.0 ratio, which is exactly equivalent to the "overlap open" strategy, so it is still worth keeping around a bounding volume in which most or all of its children overlap the shaft.

The statistics shown for "ratio open" use the mailbox marking enhancement for boxes found outside the shaft (i.e. such boxes are tagged so that during ray casting they are not tested, even though in the hierarchy). This enhancement helps more for lower ratios, as more boxes could be outside the shaft but still a part of the hierarchy. For example, for the premeshed scene, tagging with a ratio of 0.2 saved 2.6 bounding box tests per ray, at 0.6 saved 1.4, while tagging at 0.8 saved less than 0.01.

Factoring in the speedup due to "ratio open" with the savings from emitter list preprocessing, we obtain a maximum speedup of 2.44 for the new algorithms presented when compared to a standard ray caster with shadow coherency.

Future Work

What follows are some areas of research which could be done. We present them in hopes of sparking interest in pursuing further research in this area.

One improvement suggested by the statistics is gathering clumps of rays for shaft formation. For example, the floor is a receiver that may have a thousand grid points on it which need to be tested, but since the floor is so large, a shaft between it and a light source may cull out very little of the whole environment. However, limiting shaft creation to single mesh elements may create efficient shafts which then do not get used by many rays. Our standard and premesh scenes reflect these extremes. A happy medium might be to collect a set of rays from a receiving object until a certain number is reached; at this point build a shaft around the set of rays and test each ray in turn. This method has the advantages of a high number of rays to amortize the shaft culling costs, while hopefully creating a set of smaller shafts than if the whole surface was tested using one shaft.

An important property of such beams is that each cluster can use the parent object's candidate list as a starting point. That is, each sub-shaft formed from a set of rays on the object must have a candidate list which is encompassed by the object's candidate list. Using the object's candidate list will give us a head start on forming a sub-shaft's list. In fact, if we had a large enough number of samples on the emitter, it might be worth forming a separate shaft for each individual sample point on the receiver (or vice versa).

Another use of this nesting of shafts idea is superstructuring. Imagine you have stored for

each bounding volume the number of sample points on objects contained in the hierarchy below it. The number of points could be used to decide when to create a shaft: if within some given range (i.e. not too many and not too few), a shaft could be formed for the bounding volume containing receivers instead of a particular receiver. In this way all objects will have access to a candidate list of some sort, and this list will always be better than or equal to (i.e. have the same number of items or less) our emitter's original list. Other ideas are to use the number of samples per unit of surface area of the bounding volume (which should be related to the flux or density of rays between the reference items) to determine when to make a shaft.

Candidate list sorting is a simple idea worth exploring. Given a candidate list and a set of rays, there exists an optimal ordering of the list in order to minimize ray/item intersection time. What criteria to use to sort this list (and whether the list is even worth sorting) is a subject that has never been researched. One important variable to note is what purpose the ray has: the optimal shadow test list (where the first hit is desired) is almost certainly different that the optimal shading ray test list (where the closest object is desired). Such analysis could prove fruitful in other areas of graphics.

The basic shaft culling algorithm always forms the full plane set. However, some of these planes may whittle very little space away from the extent box's volume. A quick approximation of how much space a plane trims from the extent box volume could be used to determine whether it appears profitable to use that plane for testing. Sorting the planes by their approximate contribution is another speedup; we want to find if an object is outside the shaft as soon as possible, so using the plane which chops the most space first will minimize shaft culling time. Figure 2 shows some eight-plane shafts; all planes make a fair contribution to the shaft formed at the right, but the left shaft's top four planes chop off little volume of the extent box compared to the bottom four.

New strategies are an area open for research. The "ratio open" strategy is effective at high sampling rates, "keep closed" for lower rates; this suggests making strategy selection dependent upon the number of rays to be tested. Strategies based upon other factors (e.g. approximate solid angle or relative shaft volume) are also worth investigating.

The extension of shafts to other efficiency schemes is also of interest. As has been shown, 5D classification can immediately benefit. Grid and octree efficiency algorithms could be used with plane sets; however, the rules for finding items to test would have to be reworked. Boxes which are not axis aligned, slab sets [Kay & Kajiya, 1986], and ellipsoids are bounding volumes which can fit into shaft testing with some work. There may be some interesting insights to be gained by exploring how to test these efficiently.

The mathematical nature of shafts is an area of research in its own right. Bounding boxes which are tested against a shaft are always correctly categorized (though of course their contents may not be). We strongly believe that shaft formation yields no boxes which are falsely categorized (i.e. false positives). However, an arbitrary set of planes forming a polyhedron cannot be used to test bounding boxes without the risk of false positives. In Figure 4 is a case in which the *MAX_Y* plane seemingly adds nothing to the volume, since all the other planes fully demark the volume's bounds. However, a box above the point light is within the bottom three extent box planes and overlaps the two planes in the plane set - it is the *MAX_Y* plane which is necessary to correctly categorize the box as being outside the shaft. We have constructed an informal proof showing that shafts do not give false positives

for box testing, but have not formally studied what are necessary and sufficient conditions to avoid miscategorization.

Summary

We have presented methods to improve the use of ray casting in determining visibility for radiosity. Taking advantage of object coherency, we form candidate lists for ray casting from a hierarchical bounding volume structure by a variety of means. One operation presented is the building and use of the volume between two items to cull out those items in the environment which cannot occlude visibility and to form an efficient list of items for intersection testing. A number of efficiencies are explained which make this shaft forming and testing algorithm more profitable to use. Some future research directions are identified.

Acknowledgements

Kells Elmquist first posed the problem of how to find the set of objects between two given objects. The algorithm presented was tested using the ARCore radiosity and ray tracing software developed by a large number of people at 3D/Eye and Hewlett-Packard. Thanks to Juhana Kouhia for careful proofreading of the text.

References

Arnaldi, B., Priol, T. & Bouatouch, K. (1987) *The Visual Computer*, **3**,3 98-108, "A New Space Subdivision Method for Ray Tracing CSG Modelled Scenes."

Arvo, J. & Kirk, D. (1987) *Computer Graphics* (SIGGRAPH '87 Proceedings), **21**,4 55-64, "Fast Ray Tracing by Ray Classification."

Arvo, J. (1990) **in:** Glassner, A.S. (ed) *Graphics Gems*, Academic Press, London, 335-339, "A Simple Method for Box-Sphere Intersection Testing."

Cohen, M. & Greenberg, D.P. (1985) *Computer Graphics* (SIGGRAPH '85 Proceedings), **19**,3 31-40, "The Hemi-Cube: A Radiosity Solution for Complex Environments."

Cohen, M., Chen, S.E., Wallace, J.R. & Greenberg, D.P. (1988) *Computer Graphics* (SIGGRAPH '88 Proceedings), **22**,4 75-84, "A Progressive Refinement Approach to Fast Radiosity Image Generation."

Goldsmith, J. & Salmon, J. (1987) *IEEE Computer Graphics and Applications*, **7**,5 14-20, "Automatic Creation of Object Hierarchies for Ray Tracing."

Haines, E.A. & Greenberg, D.P. (1986) *IEEE Computer Graphics and Applications*, **6**,9 6-16, "The Light Buffer: A Ray Tracer Shadow Testing Accelerator."

Kay, T.L. & Kajiya, J.T. (1986) *Computer Graphics* (SIGGRAPH '86 Proceedings), **20**,4 269-278, "Ray Tracing Complex Scenes."

Marks, J., Walsh, R., Christensen, J. & Friedell, M. (1990) *Proceedings of Graphics Interface '90*, Canadian Information Processing Society, Toronto, Ontario, 17-30, "Image and Intervisibility Coherence in Rendering."

Nakamae, E., Ishizaki, T., Nishita, T. & Takita, S. (1989) *IEEE Computer Graphics and Applications*, **9**,2 21-29, "Compositing 3D Images with Antialiasing and Various Shading Effects."

Salesin, D. & Stolfi, J. (1989) *Proceedings of the PIXIM '89 Conference*, Hermes Editions, Paris, France, 451-466, "The ZZ-Buffer: a Simple and Efficient Rendering Algorithm with Reliable Antialiasing."

Wallace, J.R., Elmquist, K.A. & Haines, E.A. (1989) *Computer Graphics* (SIGGRAPH '89 Proceedings), **23**,3 315-324, "A Ray Tracing Algorithm for Progressive Radiosity."

Weghorst, H., Hooper, G. & Greenberg, D.P. (1984) *ACM Transactions on Graphics*, **3**,1 52-69, "Improved Computational Methods for Ray Tracing."