OCCLUSION EVALUATION IN HIERARCHICAL RADIOSITY

Yann Dupuy, Mathias Paulin and René Caubet

I.R.I.T.[1]
118, route de Narbonne
31062 Toulouse cedex 4
France

{ydupuy|paulin|caubet}@irit.fr http://www.irit.fr/SYNTHIM

ABSTRACT

In any hierarchical radiosity method, the most expensive part is the evaluation of the visibility. Many methods use sampling and ray casting to determine this term. Space partitioning considerably speeds up the computation process. Partitioning with shafts, leads to a quite precise subdivision of 3D space, as far as interactions between pair of objects are concerned.
The use of bounding boxes allows to speed-up many computations, such as collision or intersection detection. Those intersections can profitably be used to determine visibility between objects. Axis aligned bounding boxes allow very fast evaluation of intersection, but are not that precise, whereas oriented bounding boxes, much closer to the 3D object achieve more accurate visibility evaluation.
We present here a method that allow to quickly and accurately determine the relative position of an object and a shaft (inside, outside, occluding), and how to implement it in a hierarchical radiosity algorithm, in order to limit the hierarchy construction where not necessary.

Keywords: Radiosity, shaft, visibility, occlusion, bounding boxes.

1. INTRODUCTION

Visibility is probably the most expensive task in any image synthesis method. Determining whether a light is visible from a given point in ray tracing, or giving the part of an emitter patch, seen from a receiver point or patch in radiosity model is really CPU-time consuming.
In the hierarchical radiosity method [Hanra91] (in wavelet radiosity [Gortl93] as well), it is possible, given a few hypotheses, to distinguish the form factor computation from the visibility term in the double integral. One of those restrictions is the consideration of an average visibility factor between to patches. To compute that term, there are many possibilities, such as, for instance, ray casting or using the hardware of workstation (SGI pbuffer) [Alons99]. Whatever the chosen solution is, we need to use the other objects of the scene, and the fewer they are, the faster it is. The commonly used acceleration methods are space partitioning [Glass84][Haine91][Shen89], to reduce the number of objects to be considered.
We chose a partition of space using shafts, which has already been successfully used [Drett97][Shaw97] [Stamm97], as it is probably one of the most cost-effective partition methods. We will briefly explain that partitioning method, and present its implementation in a hierarchical radiosity algorithm. Then, we will discuss about the efficiency of adding full occlusion testing.

2. A POWERFUL TOOL: THE SHAFT

DEFINITION AND CONSTRUCTION

The shaft, introduced by E. Haines [Haine91], allows an important time saving, as far as visibility culling is concerned. It represents the portion of space concerned by any interaction between two objects, as presented on Figure 1. They initially intended to accelerate ray tracing, but they can really be used at

[1] Institut de Recherche en Informatique de Toulouse, Computer Science Research Institute, Toulouse

any visibility purpose. The shaft is a set of planes bounding the interaction volume. Those planes are based on the axis aligned bounding boxes of the objects, leading to a maximum of fourteen planes. The planes are built following the objects respective position in every of the three axes. The construction process is detailed in Haines's paper.

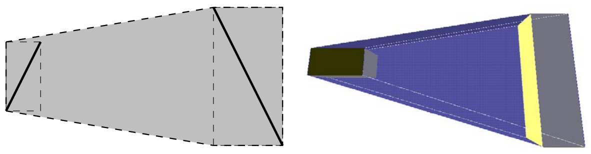


Figure 1. Shaft based on two objects.

In order to perform some of the particular computations we will need later, we must add something to the explained construction. We want to get the planes ordered in a cyclic way. Instead of sorting the planes, thanks to their normal for instance, we preferred to exhaustively determine every possible case, each determined by the objects respective positions. Note that the order is senseless for some positions, as for example one object crossing the other. In such case, any visibility consideration is also senseless, so that the shaft will not be used as is (we will split it, building a hierarchy, as explained a bit later).

OBJECTS SORTING

The main idea of shaft is a fast determination of objects that are implied in the interaction of the two objects. As far as visibility is concerned, an object intersecting the shaft might partly occlude, and sometimes completely hide one of the base objects from the other. We can define these objects as the set of potential occluders. Determining whether an axis aligned bounding box intersects the shaft or not, is a quite easy thing and a cheap computation. Afterwards, when determining the energy transfer between the objects (essentially the visibility value), we just have to deal with the set of potential occluders, the other objects being simply culled out. The shaft is thus also an easy way to sort objects for further computations needs, particularly the creation of a hierarchy.

SHAFT HIERARCHY

Let us consider a single room, with a light near the roof and a few more furniture in it. The lighting of the floor comes from the top light. The energy transfer between the light and the floor are all contained in the shaft built with these objects. Any piece of furniture, lying on the floor, creates umbra on the floor. As a matter of fact, the bounding box of that piece of furniture intersects the shaft volume.

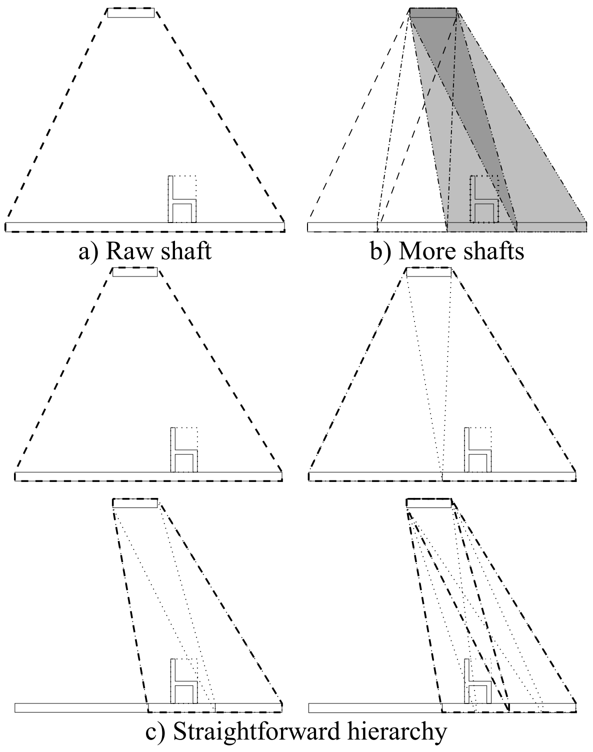


Figure 2. Shafts and accuracy: more shafts give better visibility approximation. Going down the hierarchy refines the visibility approximations

We could therefore conclude there is only partial visibility from the light to the floor Figure 2a. As it is by far inaccurate, we can refine the result: instead of considering the whole floor, we could divide it into parts and build the shafts from the light to each part of the floor. Thus only a few of them would be concerned by the piece of furniture, as we can see on Figure 2b. We have to notice that the bounding box of a given object obviously contains any part of the object, and as far as axis aligned bounding boxes are concerned, the bounding box of each part as well. The shafts built from parts of the floor are all included in the shaft built using the floor as a single object. We can thus subdivide a shaft if it gives unsatisfying results. We have to notice that the set of potential occluders for a “sub-shaft» is a sub-set of the potential occluders of the whole shaft. We can then straightforward create a hierarchy of shafts by iterating the subdivision process (Figure 2c) on the objects.

3. USING SHAFTS WITHIN HIERARCHICAL RADIOSITY

Hierarchical radiosity, as introduced by P. Hanrahan [Hanra91], is a link-based algorithm. A shaft can be considered as the materialization of a link, and then it seemed natural to use a hierarchical algorithm to implement our method. In order to get the best performances as possible, we must use a “good” subdivision oracle, which is an oracle fast enough, while accurate.

SUBDIVISION ORACLE

There are different oracles to decide whether to refine the links or not. They mostly depend on the needed accuracy and the implementation of the hierarchical algorithm. Human eye is sensitive to contrast, rather than light intensity. This is why a non-sense umbra is very confusing, whereas an artificial light source might not be a problem (see the flash for photography). But light intensity is much easier to quantify, as far as we want to control the error on the solution.

Given the hypothesis of constant radiosity among a whole patch, it is possible to take the visibility value out of the energy transfer integral, compute it separately and use it as a scalar coefficient applied to the form factor. We chose to separate the computations of visibility and energy transfer, because it's easier to control and use in our method. The visibility is evaluated using some of the shaft properties, as we will see in paragraph. The possible values are either 0, 0.5 or 1.

In order to control the accuracy of the solution, we compute the form factor gradient among the receiver patch: if there is too much variation, we will subdivide and go deeper in the hierarchy.

If the visibility is not full (visibility strictly inferior to 1), the visibility factor modifies a bit that oracle. If we detect a full occlusion, we will not subdivide, as there is in fact no energy transfer between the objects. If the visibility is somehow partial, we subdivide unless the energy transfer is inferior to a certain threshold. We then provide a complete control on the solution accuracy, which was a need. Here is, in pseudo-code, the algorithm we use:

```
compute the shaft between the objects
for each object in the parent shaft
        look for an intersector.
                is it an occluder?
if the shaft is occluded
        energy transfer is inexistent
        quit
else
        if there is an intersector
                the minimum form factor is set to zero
        compute the error on the form factor (max-min)
        if the error is greater than a threshold
                set the visibility to 0.5
                subdivide current shaft
        else
                compute “real” visibility
                quit
```

VISIBILITY EVALUATION

In order to determine the visibility between the objects, there are several ways. We could cast some rays and compute the ratio of them intersected by any objects between the source and the receiver. But this could turn inaccurate for big objects unless we use a huge amount of rays. In order to save computing time, we intensively use the shaft hierarchy. Most of the time we can tell whether the shaft is fully, partially or not occluded at all. We just need to compute more precisely the visibility factor (with ray casting) when we stop subdividing a link.

We previously said that using axis aligned bounding boxes was fast and easy. But it might also be very interesting to determine if an object totally occludes the shaft: this could avoid a lot of vain subdivisions though there is no energy transfer. The problem is not to be too optimistic in deciding whether a shaft is occluded or not, as it could result in counterfeit dark region. All the more than "missing" a total occluder would most of the time result in excess computation (excess ray casting for instance), but the visibility computation would still be correct. When considering a hierarchy of shafts, marking a shaft as being completely occluded avoid any more subdivision of that shaft, and consequently any more energy transfer computation. That is why the very attractive "occlusion" criterion must be wisely used.

Axis Aligned Bounding Boxes

The existence of the intersection of a shaft and an AABB is straightforward and very fast. But we must admit that an AABB is a very bad approximation of a convex object (we only deal with convex objects as most of complex concave objects can be divided into many convex ones). So we can decide that an object is a potential occluder, though it is not. The problem is even worse with occlusion: an AABB might completely occlude a shaft, whereas the object hardly intersects it. Figure 3 is a trivial 2D example that points out this problem.

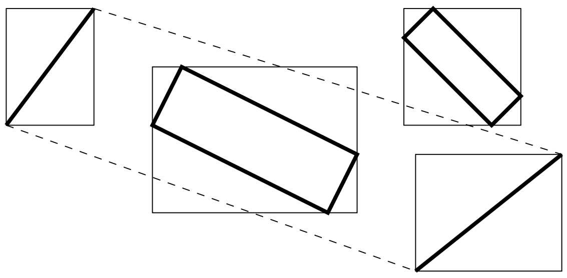


Figure 3. False intersection or occlusion with the AABB, whereas the OBB is more accurate.

It seems then really absurd to trust any occlusion consideration with AABB. And even though the ray casting will perfectly work to determine the final visibility, we will create unnecessary hierarchy parts. And the ray casting will also be a bit too much time consuming as we will try rays against objects that are not really in the set of potential occluders.

Oriented Bounding Boxes

We define the oriented box of an object, as the smallest box containing the object. It is always much more accurate than an AABB (see Figure 3), though it can hardly represent some objects (a sphere, a pyramid). It would be ideal to work with the convex hulls of convex objects only, but the bounding volume might happen to be as complex as the object itself. So let us consider that the oriented bounding box is close enough to the objects. Thus, an oriented bounding box fully occluding a shaft means that the object is actually obstructing the visibility between the considered objects.

First of all, an object that occludes a shaft, is obviously intersecting it, and therefore in the set of potential occluders. The occlusion test will not be performed on every object, but only on those intersecting the shaft. Doing the contrary would be a waste of time.

Then we can say that a box occludes a shaft if it contains a full slice of it. We just have to care about "side planes" of the shaft: these are the planes relying on both objects. When dealing with AABB, this is equivalent to find a point outside each plane.

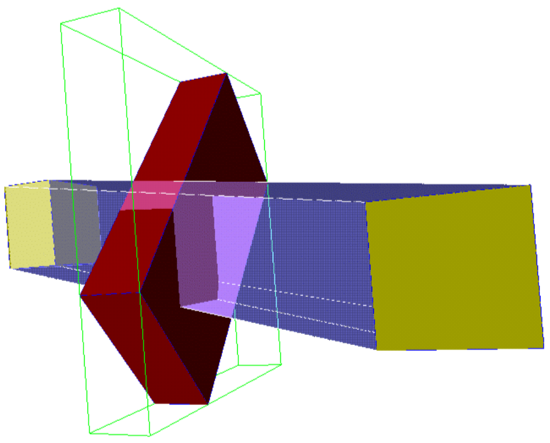
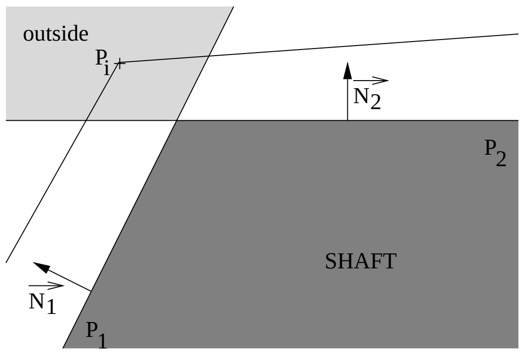
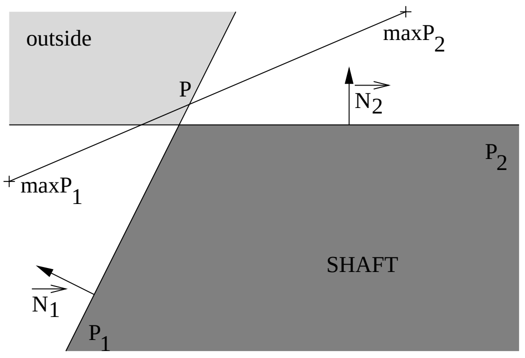


Figure 4. The AABB of the object includes a full slice of the shaft and thus occludes it. The OBB gives a different result, despite there is a vertex outside each plane.

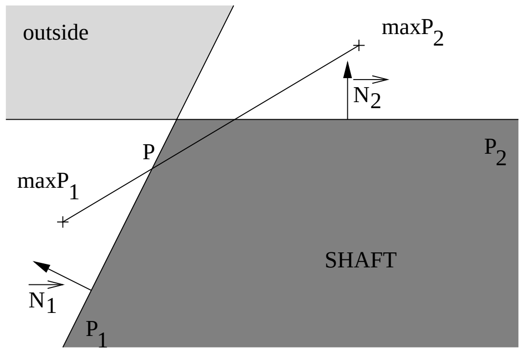
But with oriented boxes, this is obviously wrong (see Figure 4). If we consider two planes, finding a vertex outside each plane does not necessarily implies that the edge is also outside the shaft. It might cross it as well. We must test the box against two planes at a time, not a single one. Figure 5 shows the different possible space configurations, and consequently, the different cases we have to handle.



a) Point outside the planes



b) Edge outside the planes



c) Edge inside the planes

Figure 5. Close up of what happens considering a pair of planes. The point P, intersection of the out-most edge and the plane P1, might either be inside or outside plane P2.

4. CONCRETE IMPLEMENTATIONS

Before going any further in the algorithms, here a few useful properties.

We represent a shaft by a set of plane, each defined by its normal vector $\vec{N}(a,b,c)$, and its distance to origin *d*. A plane splits 3D space into two regions: we chose the normal to aim at the exterior. We can tell whether a point $\vec{P}(x,y,z)$ is inside or outside a plane by computing $\vec{N}\cdot\vec{P}+d$. A positive result indicates that the point is outside the plane, whereas a negative one implies $\vec{P}$ is inside the plane.

An AABB is represented thanks to three intervals, one for each world coordinate. An OBB is defined thanks to one of its vertices, the origin, and the three edges of the box containing that vertex, the axes.
An object is outside of a volume defined by a set of planes, if one of the planes satisfies all the vertex of the object are outside of that plane. In other words, an object might intersect a set of planes if there is at least one vertex inside each plane (one vertex per plane, not a common vertex for all planes). This is just a necessary condition as we can see Figure 6.

INTERSECTION

To avoid testing all eight vertices of a box against each plane, we use arithmetic for intervals [Snyde92]. If we consider the plane analytical expression, the low bound of the function interval must be negative. So we just need to compute that minimum value. The plane expression is linear in every three dimensions, so to compute the minimum value for an AABB, we must take either the minimum or maximum value of an axis, depending on the sign of the normal vector coordinate.
The way to obtain the minimum for OBB is slightly different. The minimum is reached for one of the vertices. We can obtain that minimum by computing the value for the origin of the box, and adding the difference value ($\vec{N} \cdot \overrightarrow{axis}$) for each axis of the box (if negative, of course). We then just have to verify that the minimum is negative.
We must notice that this algorithm is optimistic, as it might find some intersections where there are not. This is not a real problem concerning intersection. In case of partial visibility we use ray casting as final evaluation so that the result will still be correct.

OCCLUSION

As we previously mentioned, we must be very careful with occlusion. This means that we must absolutely not decide that there is an occlusion if there are any doubts. So the necessary condition we mentioned at the beginning of this section must be wisely used.

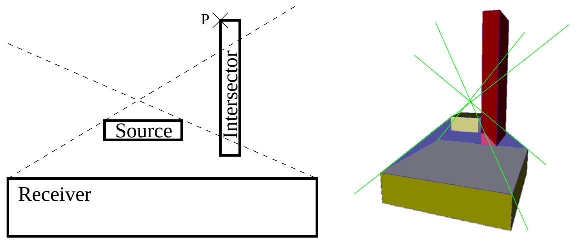


Figure 6. Incorrect occlusion consideration

Figure 6 perfectly explains this. There is a point outside each plane (each pair for 3D space), but there is no occlusion.
2D case is quite easy to explain, considering one case at a time. If an object includes both the source and the receiver of a shaft, then this object should not be considered as an occluder as it does not interfere in the visibility between the objects. Else, if an object includes only one of the objects, there is no visibility among the shaft. And if an object partially intersects another one, we must split the object, creating a shaft hierarchy. We can thus consider we have only none overlapping objects. In such conditions, if one point is outside the planes (Figure 6, left), there can be no occlusion. 2D case is simple as there are only two planes.
We could not find any theoretical method for 3D (Figure 6, right), so we tried and guess a little trick to cope with that annoying case. We can consider the three orthogonal projections of the shaft following the three reference axes. With each of these projections, we can define up to four pairs of opposite side planes. Back to 3D, we experimentally determined that if there were N pairs of opposite planes, there could not be an occlusion if we could find one point outside both planes of a pair, for N-1 pairs.
We have already seen that we had to test the box against a pair of planes at a time. There are three possibilities, shown on Figure 5:

- There is a point outside both planes
- There is no vertex outside both planes, but an edge crosses it
- There is neither a vertex nor an edge outside the planes

Given a pair of planes, if we can find a point satisfying the first case, we do not need to compute any further for that pair. On the contrary, if we fail to find a point outside both planes, we need to determine whether the "most external" edge crosses the common exterior or not. To do so, we can determine the intersection point of one of the plane and the candidate edge, and apply the obtained point to the other plane's equation in order to know if this point is outside or inside the plane.
We note $P_i\left(\vec{P}\right)$ the value of the function of plane P_i applied to the point $\vec{P}$. To find the most external edge, we must find the two vertices $\overrightarrow{\mathrm{maxP_1}}$ and $\overrightarrow{\mathrm{maxP_2}}$ satisfying for $i=1,2$ and $j=2-i$:

$$P_i\left(\overrightarrow{\mathrm{maxP_i}}\right)+P_j\left(\overrightarrow{\mathrm{maxP_i}}\right)=\max_{\vec{P}}\left\{\begin{matrix} P_i\left(\vec{P}\right)+P_j\left(\vec{P}\right) \\ with\ P_i\left(\vec{P}\right)\geq 0\end{matrix}\right\}$$

Let P_1 and P_2 be the two studied planes, as shown on Figure 5. Both of them are defined with their normal $\overrightarrow{N_i}$ and their distance to origin d_i. Let $\overrightarrow{\mathrm{maxP_1}}$ and

$\overrightarrow{\mathrm{maxP}_2}$ be the obtained vertices of the out-most edge. The intersection point $\vec{P}$, between the edge and the plane P_1, satisfies $\vec{P} = t \times \overrightarrow{\mathrm{maxP}_1\mathrm{maxP}_2} + \overrightarrow{\mathrm{maxP}_1}$ (as it is a point of the edge) and $\vec{N}_1 \cdot \vec{P} + d_1 = 0$ (as it must be on the plane P_1). This implies $t = -\dfrac{P_1\left(\overrightarrow{\mathrm{maxP}_1}\right)}{\overrightarrow{N_1} \cdot \overrightarrow{\mathrm{maxP}_1\mathrm{maxP}_2}}$.

We wonder about the position of the point $\vec{P}$ respectively to the plane P_2. Keeping the same implicit functions considerations, we want to determine the sign of the function plane P_2 applied to the point $\vec{P}$, $P_2\left(\vec{P}\right)$. Introducing the previous expression of t within $\vec{P}$, results in the equivalent formulation:

$$P_2\left(\vec{P}\right) = P_2\left(\overrightarrow{\mathrm{maxP}_1}\right) - \frac{\overrightarrow{N_2} \cdot \overrightarrow{\mathrm{maxP}_1\mathrm{maxP}_2}}{\overrightarrow{N_1} \cdot \overrightarrow{\mathrm{maxP}_1\mathrm{maxP}_2}} P_1\left(\overrightarrow{\mathrm{maxP}_1}\right)$$

If the evaluation of the function results negative, the edge is inside the planes, and there is no occlusion. Otherwise, we must iterate the process with the next two planes.

Note that every plane might be treated once as P_1 and another time as P_2. To avoid computing twice the same things, specifically the values of $P_i\left(\vec{P}\right)$, we can use a cache to store the results once computed. That way we ensure a worst case of n dot products computation if n is the number of vertices of the bounding box. This algorithm also applies if we deal with the vertices of the object itself (considered convex) or of any other of its convex bounding polyhedron, and not just its OBB. So we can use it even if the OBB turns to be a bad approximation of objects, we will just need to define a better convex bounding volume.

5. BENCHMARKING AND RESULTS

To compare the use of AABB and OBB, we worked on different scenes (Figure 8 and Figure 9 on the last page). Some of them are taken from Smits and Jensen report [Smits00], as they aim at being a wide spread reference.

Most of these are not “real world” scenes, but intend to test specific problems of the algorithms. The duplex scene is a bit more complex, and maybe also more realistic.

The AABB algorithm does not use the occlusion testing, and therefore should create more links (and shafts). On the contrary, the OBB method computes the occlusions: it should result in fewer links, but longer to compute.

All the following numbers are the results we obtained on some of the test scenes, running our test application on a 600 MHz Intel Pentium III. The value called “time” represents the CPU time in seconds, and “mem” is the memory usage, in megabytes. The workstation has enough memory to prevent swap, and we started the application from fresh between each test, to avoid any undesirable cache acceleration.

VRML scene	AABB		OBB	
	time	mem	time	mem
Cornell Box	8.89	21.7	8.97	20.9
Duplex	61.13	44.5	52.15	36.9
Geometry	12.36	23.1	11.39	21.1
Shadow	9.18	22.4	9.87	21.3

These results confirmed are first guess as far as memory is concerned. But looking at the different CPU times, things are not so clear. In fact, for simple scenes, using AABB is sometimes faster than OBB. This is easy to explain: there are very few objects, and consequently, the time needed for ray casting is quite negligible. When the scene becomes a bit more complex (like in the Duplex for instance) the method performs better. Mainly for two reasons: on the one hand there are more objects to deal with when casting rays, and on the other hand there more occlusions in the scenes. Of course both reasons are strongly linked. But there are also many occlusions because the objects are from various size and shape.

The Cornell Box is probably one of the worst scene as far as occlusion is concerned. A very small amount of little surface elements are really occluded. The occlusion method would probably outperform the raw AABB intersection - ray-casting algorithm, in architectural scenes, such as the infamous Soda Hall, which provides a huge number of walls, and then a huge number of occlusion. But this is obviously unfair and it would be more interesting to compare our method against well-proven BSP trees or portals. We will come back on that a bit later on.

Anyway there is a very annoying point we must absolutely not skip. We noticed on some pictures some abnormally dark elements (See the close up of the stairs in the Duplex scene computed with rather strict parameters: pretty small minimum area and reduced transfer accuracy margin Figure 7).

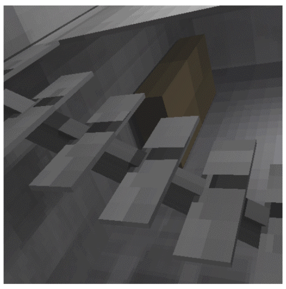


Figure 7. Occlusion error

We mention that the intersection determination was not fully correct: the algorithm might find false intersections. As we said, this really not a problem for intersection, and most of the time (partial visibility) there are no artifact on the pictures. But this turns much more dangerous when this incorrect intersection lead to completely wrong occlusion, as it is the case on the stairs. In fact, as we only consider the side planes to test for an occlusion, we can find an occlusion "behind" one of the objects. In the Duplex, the beam occludes the stair, though the stair is nearer the light than the beam. We can see mainly two ways to avoid such problems: compute correct intersection or assure that the occlusion is in between the two objects. As these cases seem rather not frequent, we think that it will not affect a lot the results we obtained with our implementation.

6. CONCLUSION AND FUTURE WORK

We show in this paper an efficient way to improve (mainly in terms of memory, slightly for CPU time) the radiosity rendering process. The gain in time is more satisfying as the scene becomes more complex. In the very near future, we must compute real intersections, to avoid the problems we encountered. And we must verify that it does not change the results a lot.

One can think that an oriented bounding box is not accurate enough. Regarding our algorithm, this is not a real problem. It works for any convex polyhedral bounding volume, and consequently we can use better volume estimation for more complex objects. All the more than it is possible to reduce the possibilities of erroneous occlusion consideration: by setting a threshold, we can ensure the bounding volume is more or less occluding the shaft. Instead of saying a vertex *V* is outside a plane *P* by comparing *P(V)* and *0*, we can compare *P(V)* to the threshold. For instance, the threshold could be computed for each object so that we try to determine whether the object occludes a shaft 5% "wider" than the real one. The threshold can also take into account the ratio of the object volume and its bounding polyhedron volume. It is then possible to reduce the risk of incorrect occlusion, which is really important for good shadow quality.

Of course, the occlusion only works for convex objects. It would be really interesting to apply these algorithms to maximal convex parts of non-convex objects. But determining such parts seems hardly possible without heavy computation [Szilv86].

Another interesting idea to explore is the fusion of occluders, in order to increase the number of possible occlusions. We can merge the different intersectors into a bigger equivalent one, in order to determine occlusions due to many objects and not a single one, a bit like in [Schau00].

We mentioned portals and BSP trees to manage big set of objects, specifically huge architectural buildings. It would be interesting to compare our algorithms (with clustering [Silli94][Smits94], to handle huge amount of objects), to these wide spread methods. We also think to the shaft as a tool to efficiently build portals. Last but not least, will be to try our method with dynamic environments, and see if it can be interesting in implementations similar to Drettakis Line-Space Hierarchy [Drett97], which also uses shafts. We are rather optimist, as we showed no negligible gain in memory. We could thus store a bit more information and compute the modifications in interactive time.

REFERENCES

[Alons99] Laurent Alonso and Nicolas Holzschuch: Using graphics hardware to speed-up your visibility queries. *Accepted in Journal of Graphics Tools*, 1999.

[Drett97] George Drettakis and François X. Sillion: Interactive update of global illumination using a line-space hierarchy. In *SIGGRAPH '97 Conference Proceedings*, Annual Conference Series, pages 57-64.

[Glass84] Andrew S. Glassner: Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15-22, October 1984.

[Gortl93] Steven J. Gortler, Peter Schroder, Michael F. Cohen and Pat Hanrahan: Wavelet radiosity. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 221-230, 1993.

[Haine91] Eric A. Haines and John R. Wallace: Shaft culling for efficient ray-cast radiosity. In *Photorealistic Rendering in Computer Graphics*, Eurographics, pages 122-138. Springer-Verlag Berlin Heidelberg New York, 1991.

[Hanra91] Pat Hanrahan, David Salzman and Larry Aupperle: A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH'91 Proceedings)*, 25(4):197-206, July 1991.

[Schau00] Gernot Schauffler, Julie Dorsey, Xavier Decoret and François X. Sillion: Conservative volumetric visibility with occluder fusion. In *SIGGRAPH'00 Conference Proceedings (New Orleans, LO, July 23-28, 2000).*

[Shaw97] Erin Shaw: Hierarchical radiosity for dynamic environments. *Computer Graphics Forum*, 16(2):107-118, 1997. ISSN 0167-7055.

[Shen89] L. S. Shen, E. Deprettere and P. Dewilde: A new space partition technique to support a highly pipelined parallel architecture for the radiosity method. In *Fifth Eurographics Workshop on Graphics Hardware*, 1989.

[Silli94] François X. Sillion: Clustering and Volume Scattering for Hierarchical Radiosity Calculations. In *Fifth Eurographics Workshop on Rendering*, pages 105-117, Darmstadt, Germany, June 1994.

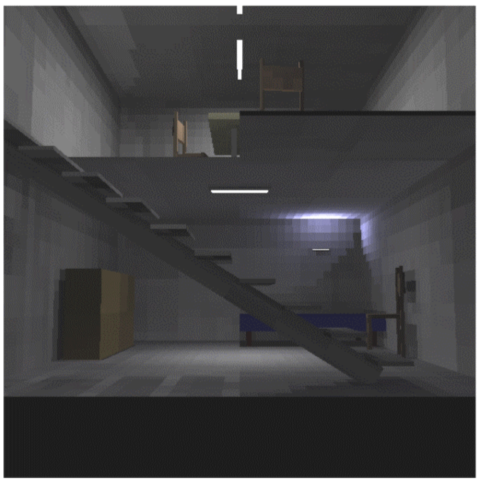
[Smits00] Brian Smits and Henrik Wann Jensen: Global illumination test scenes. Technical Report UUCS-00-013, Computer Science Department, University of Utah, June 2000. http://www2.cs.utah.edu/~bes/papers/scenes.

[Smits94] Brian Smits, James Arvo and Donald P. Greenberg: A clustering algorithm for radiosity in complex environments. In *Proceedings of SIGGRAPH'94 (Orlando, Florida, July 24-29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 435-442.

[Snyde92] John M. Snyder: Interval analysis for computer graphics. Computer Graphics (SIGGRAPH'92 Proceedings), 26(2):121-130, July 1992.

[Stamm98] M. Stamminger, Ph. Slusallek and H.-P. Seidel: Three point clustering for radiance computations. In *Rendering Techniques'98 (Proc. Eurographics Workshop on Rendering '98)*, pages 211-222, Springer, 1998.

[Szilv86] M. Szilvasi-Nagy: Two algorithms for decomposing a polyhedron into convex parts. *Computer Graphics Forum*, 5(3):197-201, September 1986.


Downstairs view

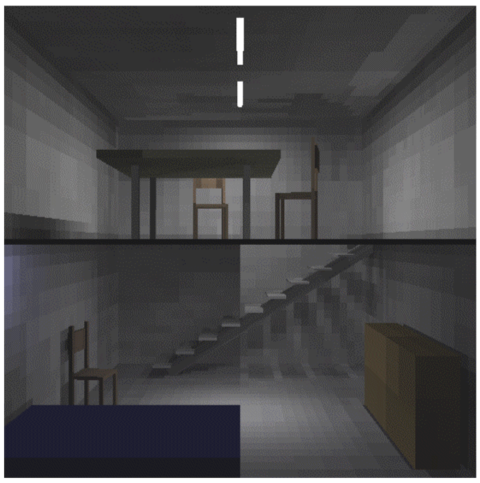
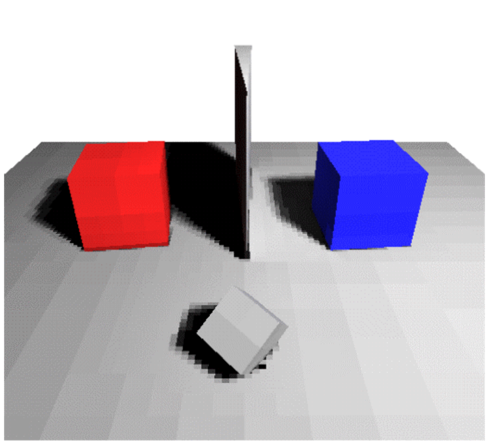

Reverse view

Figure 8. Duplex test scene


Geometry scene

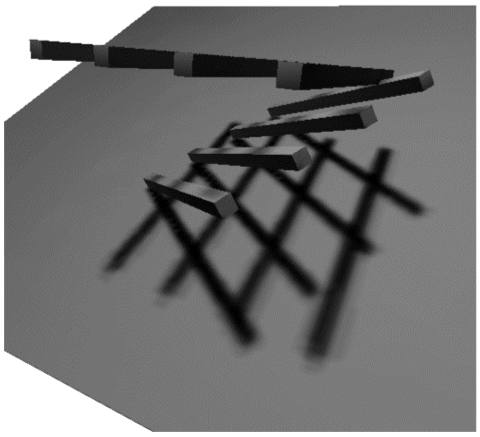

Shadow scene (smooth, no t-vertex removing)

Figure 9. Specific visibility test scenes [Smits00]
(Available at http://www.cs.utah.edu/~bes/graphics/scenes/)