Different approaches to reduce shaft hierarchy in dynamic radiosity

Yann Dupuy, Mathias Paulin and René Caubet
IRIT - Computer Science Research Institute
Toulouse, France

Abstract

Hierarchical radiosity provides a good model for the lighting of static scenes. But when an object moves, this model requires an update of all the lighting. In fact, thanks to space-time coherence, we just have to compute some of the geometry parameters (form factors). The problem is to determine which part of the hierarchy of links must change or not.

A shaft is a practical mean to represent the interactions of light between two objects. A hierarchy of shaft describes all the interactions in a 3D scene. So the intersection of the moving object and the hierarchy of shafts directly gives the part of the hierarchy that are concerned by the displacement.

The time needed to compute the new required form factors is somehow proportional to the total number of shafts of the hierarchy. Therefore, in order to decrease this computation time, we present a method based on the Oriented Bounding Box of the objects and compare it to a more standard Axis Aligned Bounding Box approach.

Keywords: Dynamic Radiosity, OBB, AABB, Visibility, Shaft.

1. INTRODUCTION

Radiosity is a perfectly suited model for walk-through class applications. The lighting of the scene is computed only once, and afterward, today's hardware is able to cope with the high number of polygons needed by the rendering of the solution. But as soon as the scene turns dynamic, things are much more complicated.

As for any global model, the smallest change implies a different solution for the whole scene. Some works ([1], [2], [10], [9]) have already shown that it was possible to profit by the space and time coherence to avoid computing everything from scratch.

In fact, we can characterize two kinds of “dynamic” environments. On the one hand, in scenes where every move is known at the beginning, we can compute all the changes in lighting before displaying the first frame. This can take a while, but it is then possible to display the dynamic scene in “real-time”. This would typically be the case when creating movies, instead of computing a frame at a time.

On the other hand, many other applications require a different approach, as we might neither have the necessary long pre-processing time, nor know what will append. For instance, in a virtual visit of a building, the visitor might want to interact with the furniture, or even do simple things like opening or closing a door, switching on or off a light. This implies we are able to compute the new lighting situation on the fly. And we must absolutely compute it in a very short time.

We can define this constraint as “interactive time”: the acceptable delay for rendering a new correctly light scene after the user action. Longer than real-time, but short enough to provide interaction between the user and the application. As we only have little time, we must reduce the computations.

Considering the radiosity model, many form factors will not be affected by the displacement of an object. Knowing that the computation of the form factors is the longest part of the resolution process, this is a rather good piece of news. In the case of hierarchical radiosity, only a few links will change. And we need a quick identification of these.

Convex objects have very interesting features, such as providing real light occlusion: a chair cannot lead to plain shadow, whereas the convex seat itself does. This is why, from now on, we will only consider convex objects, as most of complex concave objects can be divided into many convex ones.

2. WHAT'S TO BE MODIFIED?

The use of shaft, as defined by Eric Haines, provides an excellent subdivision of space: each shaft bounds the interaction between two objects, as far as light transfer is concerned. Considering a shaft as the volumic representation of a link, it is possible to merge the hierarchy of links (from the hierarchical radiosity method) and a hierarchy of shafts.

Now if we consider an object moving from a position to another, we can split the set of shafts into three parts:

- the shafts based on the moving object
- the shafts being intersected by any position of that object
- the shafts not being intersected

The third part will not need any new computation of form factor and/or visibility factor. On the contrary, both first and second parts require work.

When an object moves, so does its bounding box. Remember a shaft relies on axis aligned bounding boxes of two objects. We must then calculate again every shaft based on the moving object: in a scene of n objects, this involves the computation of a hierarchy coming from n-1 shafts (about twice that amount if we differentiate transfer direction). This results in approximately o(n) complexity.

If a shaft is intersected by the starting or finishing position (or both), this will most of the time result in a change of the visibility between the sides (the objects) of the shaft. In a hierarchical radiosity method [7] (as well as in wavelet radiosity [5]), we can compute the form factors without any visibility consideration, and apply a visibility factor to the energy transfer. Doing so, we will only need do compute the changes in visibility. Here again, there are several possible cases. The new position of the object can lead to a new occlusion in a shaft, as well as it can undo a previous one. A shaft that was occluded (and therefore not subdivided), will require a complete computation of its hierarchy. To avoid part of this particularly bad situation, we can only deactivate the part of the hierarchy that is not useful anymore when a change

happens. We will only have to reactivate it when necessary. In case of partial visibility, we only need to go deeper in the hierarchy and do the necessary changes.

The potential number of shafts intersected by the moving object approximately corresponds to $o(n^2)$ complexity.

Once the change made in the shaft hierarchy, we can go back to the iterative resolution process to compute the new lighting of the scene. We can either do a full gather pass, or only a local gather if the scene is too big. The iterative resolution process will apply the changes in the whole scene. Anyway, this is not the point in this paper.

The shaft hierarchy provides an excellent way to locate the link that will change because the movement of the object. Though we only have to test at the higher level of the hierarchy to see which shaft is concerned by the moving object, we must go down and test any concerned part of the shaft hierarchy. And the bigger the hierarchy, the more shaft to test and thus, the more time spent. So we must absolutely reduce the total amount of shafts in the scene. Here are a few approaches for computing the shaft hierarchy, keeping in mind that we must be quite fast!

3. THE CASE STUDY

Before going any farther with some possible way to limit the shaft hierarchy, we must briefly view how a shaft is represented. This implementation as several useful properties we shall use.

3.1 Shaft implementation

To represent a shaft we need to use planes. We can define a plane 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.

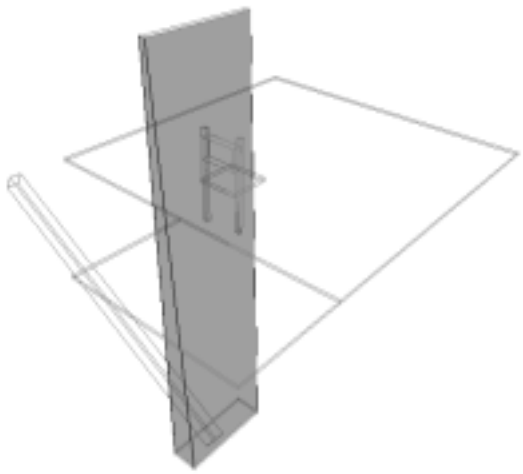


Figure 1. 3D shaft. The set of occluders is also represented.

We use E. Haines method [6] to build the shaft between two objects. We obtain an axis aligned bounding box, bounding the two AABB of the objects, and a set of planes, each plane lying on one edge of each AABB. For further algorithms purposes, those planes are sorted out, so that two successive planes have exactly one common edge. The shaft is the intersection of the insides of the box and the planes. Figure 1 is a 3D example of a shaft between a neon light and a step in the Duplex scene.

3.2 Axis Aligned Bounding Boxes

AABB is a very fast (computationally speaking) and easy representation of an object, as we just need to know about the min and max values for the three reference axes. They behave very well with shafts, as shafts themselves are built on AABB of objects. The main drawback is the poor fit of an object by its bounding box. Using AABB approximation of objects can lead to erroneous intersection considerations.

Anyway, this is not a real problem. If we use a ray casting for the final visibility computation, it will cope with possible intersectors that should not be considered. It will only have to test for ray intersection with more patches than real possible candidates.

3.2.1 Intersection

Computing an intersection between a shaft and an axis aligned bounding box is really straightforward. An AABB is outside of a volume defined by a set of planes, if one of the planes satisfies all the vertex of the box are outside of that plane. In other words, a box intersects 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).

To avoid testing all eight vertices of a box against each plane, we use arithmetic for intervals [13]. 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 the maximum value of an axis, depending on the sign of the normal vector coordinate.

3.2.2 To cast, or not to cast?

We build most of our link hierarchy without really computing any visibility value. At any node, if there is an occluder, the visibility is set to “partial” (0.5), else, it is set to “full” (1). This avoids the calculation of the visibility with for instance a ray casting. As far as the size of the hierarchy is concerned, this of course not a good thing. We can develop some part of the hierarchy though it is fully occluded.

So we also tried to create the links with a systematic visibility computation, in order to avoid those unwanted branches of the hierarchy. Anyway, this method cannot work properly: the ray casting can hardly deal with small objects. To be correct, we must cast a lot (how many?) of rays between two patches. But this would require too much time. We chose to cast 16 rays, and we will carefully use this other method, mostly to characterize the others.

3.3 Oriented Bounding Boxes

Minimal volume oriented bounding box provides a much better approximation of a convex object than the AABB. Figure 2 (left) gives a good example of this feature. Of course, there are some convex objects that can hardly be represented through their oriented bounding box. Even worse, a sphere or a tetrahedron will never fit in a box.

For visibility computations, we replace each object by its OBB. This leads to wrong but plausible results, all the more than the objects are properly subdivided into convex parts. But if we consider that an OBB is precise enough to represent an object, we can use the OBB to compute whether an object fully (not partly as we did for AABB) occludes a shaft. Of course, we reduce the number of objects to test for occlusion by testing only objects that intersect the shaft.

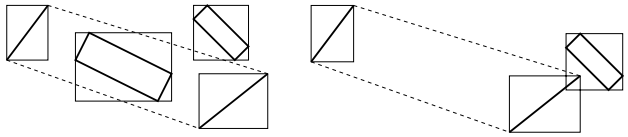


Figure 2. Didactic 2D examples of bounding volumes

3.3.1 Intersection

Using the same method as for axis aligned bounding box, leads to wrong results. There might be, for each plane of the shaft, a vertex inside, but no intersection. Figure 2 right is a 2D example. As we said for AABB, a wrong intersection itself is not annoying. But, this time, we test for occlusion. This means that we cannot consider an object that is not really intersecting the shaft. Otherwise, it could result in very disturbing artificial shadows in the rendered solution.

This why we use a totally different algorithm, based on the separating axis method. It is based on the following statement. For any line of 3D space, an object projects itself onto that line as a segment. If there is a line for which two objects project onto two separate segments, the two object do not intersect. As it is stated in Dave Eberly's paper [4], we only need to test for a finite set of lines. In our case, the set contains:

- the normals to the faces:
  - the three axes of the oriented bounding box
  - the three scene reference axes and the normals of the planes
- the cross products of pair of edges:
  - the three axes of the OBB
  - the three scene reference axes and the edges of the planes

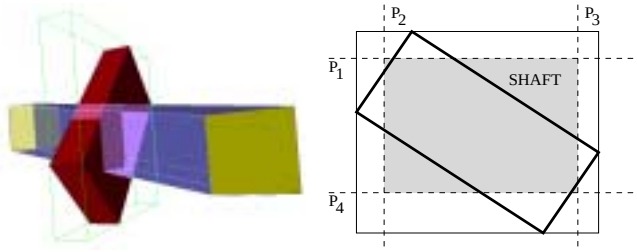
Of course for both normals and edges we must avoid as many redundancy as possible. This is done when creating the shaft: we only keep the necessary vectors (e.g. two opposite normals are considered as a single vector) and that way, reduce the size of the set. We also profit by the specific geometry of boxes and shafts to prevent projection whose we already know the result.

To compute the projected segment of the box, we just have to calculate the minimum and maximum boundaries of the segment. We then project the origin and add the projections of the axes.

To project the shaft, we can see no alternative to project the vertices of the shaft. We can slightly reduce the number of projected points by only taking into account the vertices that are on the surface of the shaft, not fully in it. These vertices are also computed when creating the shaft.

3.3.2 Occlusion

We already presented our method to compute occlusions in [3]. To sum up briefly, the occlusion of a shaft by an AABB is straightforward. If we find a vertex of the box outside each side plane of the shaft, the box occludes the shaft. This condition is absolutely not enough for OBB, as we can see on Figure 3 (a). Figure 3 (b) clearly explains why this is not sufficient.

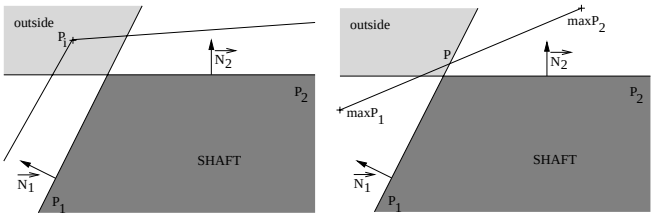


(a) AABB occludes, OBB does not

(b) 2D slice of 3D shaft

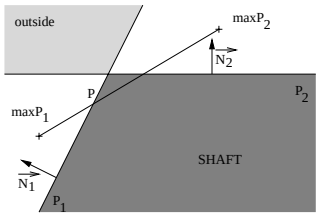
Figure 3. 2D and 3D occlusions considerations

So we treat the side planes of the shaft, a pair at a time. We try to find a point outside both planes, in other words a point in the intersection of the outsides of each plane. The different cases to consider are shown on Figure 4.



(a) Vertex of the box outside both planes

(b) No vertex but an edge outside both planes



(c) Neither a vertex nor an edge outside the planes: there can NOT be an occlusion

Figure 4. Possible geometric configurations.

Studying each pair of planes until we find a case (c), we can tell whether there is an occlusion or not. Though we can regret the lack of theoretical validation of the whole method (see in [3], the trick to cope with wrong occlusion), we did not notice any artifact on the rendered pictures.

4. STATIC RESULTS

In order to compare the previous methods, we used them to compute different kind of scenes. Here are some quite representative results we obtained for the scenes shown at the end of this paper. For each scene and each method (respectively Oriented Bounding Box, Axis Aligned Bounding Boxes with systematic visibility with ray-casting, Axis Aligned Bounding Boxes without systematic calculation of the visibility), we present in Table 1 the total number of links. We made 20 iterations to ensure the convergence of the solution, and give the computation time as an indication (600 MHz Pentium powered by Linux).

Scene		OBB		AABB with systematic RC		AABB without systematic RC	
		Links	Time	Links	Time	Links	Time
Cornell	(18 faces)	44688	11 s	55900	13 s	57796	16 s
Shadow	(60 faces)	30593	10 s	35417	11 s	35501	12 s
Four Rooms	(192 faces)	77612	36 s	113165	41 s	116931	44 s
Duplex	(252 faces)	138173	155 s	184698	166 s	188973	163 s

Table 1. Quantitative results of the three methods, for different scenes

These results seem promising. Depending on the scene, the use of the OBB method reduces the number of shafts by 15 to 35%. As the hierarchical radiosity method is very sensitive to convergence parameters, we cannot determine any straight relation between the final number of links and the initial number of faces. Anyway, reducing the size of the link hierarchy, we expect to reduce the total amount of computations for dynamic update.

As far as time is concerned, the gain is a bit smaller, from 5 to 30%, but is not linearly dependent. If we try to define a "time per shaft", we get numbers quite coherent to what we could expect: the complexity of the OBB algorithms involves a greater computation time for each shaft compared to the AABB algorithms. Additionally, casting rays systematically costs more than computing the visibility for the leaves of the hierarchy only.

Last, we can notice that the systematic visibility computation is costless than casting rays for the leaves only. We could explain that with the compared cost of computing the intersection of objects and rays, and computing a shaft and its intersection with an AABB. We will only focus on the two best methods for the rest of this paper.

5. DYNAMIC MODIFICATION

This work primarily aimed at reducing the time needed to compute the new radiosity solution after a geometric change in the scene. We therefore compared the above methods in a dynamic environment.

5.1 Update strategy

As it was our first goal, we can greatly profit by the shaft hierarchy to determine the part of the hierarchy that must be recomputed.

Our algorithm is based on clustered hierarchical radiosity. We use Smits [11] denomination of the links: β-links, α-links, and patch-links. We associate one shaft to each link and thus, indistinctively deal with the link hierarchy or the shaft hierarchy.

The higher-level shafts, which are shafts (or links) representing the interaction between entire objects, necessarily correspond to β-links. Therefore, we must have a specific treatment for those links, whereas α-links and patch-links will have a similar but reduce one.

Once the links computed, we collect the radiosity for the new or modified links. After a quick push-pull, we obtain a first (but already good) approximation of the final solution. We just need to run the main resolution process to perform any additional transfer.

5.1.1 β-links particularity

The first thing we must test is whether the source or the receiver is the moving object. If this is the case, we must update the whole involved hierarchy of the current link. As one of the base objects of the shaft moves, we must recompute every shafts of the hierarchy. This means that we have to destroy the children of the current link, and recalculate everything from scratch.

This results in a o(n) complexity, and only concerns a maximum of n-1 β-links at the top of the hierarchy (where n is the initial number of objects).

If the moving object is neither the source of the β-link, nor its receiver, we apply the same algorithm than the other links. Of course, this the most used part of the update process.

5.1.2 All links

The determining parameters of the algorithm are "WAS the moving object in the current shaft?" and "IS the moving object in the current shaft?".

First case, the object was in the shaft. If it is not in the shaft anymore, we must erase it from the list of potential occluders. Then we have to see whether the shaft is occluded or not. There are three possibilities left:

- If the shaft was previously occluded and is not after the movement, we must develop the hierarchy. If it was deactivated we only need to reactivate it. Else, we create new links.
- If the shaft was not but is occluded after the displacement, we only have to deactivate the whole children hierarchy of the current link.
- If the shaft was not and is still not occluded, we recursively update each children of the next level of the hierarchy.

Or, the object was not previously in the shaft. We only need to know if the shaft is occluded or not. And this is really easy: either it was already occluded, necessarily not by the moving object, and it remains unchanged, or the moving object involves a new occlusion. In the last case we just have to deactivate the children hierarchy. If the shaft is not occluded, we only need to update the existing next level of the hierarchy.

5.2 Results

In Table 2, we sum up some results we obtained applying the mentioned algorithms, for the same scenes. We moved either a simple object or a composed one (a chair for instance).

	OBB					AABB with systematic ray casting				
Scene	Time	Parsed links	Modified links	Deleted links	Created links	Time	Parsed links	Modified links	Deleted links	Created links
Cornell	4 s	6125	4430	14545	9246	7 s	18670	13108	13125	14956
Shadow	3 s	6352	4063	1051	1113	3.5 s	6915	4572	964	1478
Four Rooms	4,5 s	5310	3144	2645	1331	5,5 s	7466	4568	4045	2445
Duplex	70 s	50587	32512	3704	1392	78 s	63904	40083	3206	2030

Table 2. Dynamic update after some movements

These are quite early results, but we can anyway conclude some interesting facts. Of course, the movements are not exactly the same for both studied method as we moved the objects interactively, but we tried to obtain very similar "after" positions.

First interesting result is the close relation between the number of links parsed and the time needed to update the solution. As we could have expected after the static results, the OBB methods perform very well compared to the AABB algorithms. We always get better results with Oriented Bounding Boxes. This confirms are first will of reducing the hierarchy to reduce the update time.

The algorithm behavior is very sensitive to the scene geometry. We can see that there is absolutely no rule for the different numbers of links. For instance, while the number of parsed links is about the same for Cornell and Shadow scenes, the number of created (or reactivated) and deleted (deactivated) links vary a lot. Anyway, this does not seem to have a huge impact on the total time needed for the full update process.

Though for the first three scenes, the update time match our initial desire, the Duplex scene clearly shows that there is a lot of work still to be done.

6. CONCLUSION AND FUTURE WORK

All these results seem very promising. But in order to confirm, and maybe improve them, there are several ideas that should be interesting to explore.

First of all, we showed that the better was the accuracy of the bounding volumes, the smaller was the shaft hierarchy. A shaft is a quite good approximation of the volume of the possible interactions between two objects. Anyway, there is a better approximation, especially when the objects are flat patches. In this case, it would be much more interesting to use the convex hull of the two patches, instead of the shaft. By definition, the convex hull of a set of points is the smallest convex polyhedron containing these points. So we think the convex hull is a better approximation because we avoid the loss of accuracy of the AABB necessary to build the shaft.

In the same way, it would be much better to use biggest convex volume included in each object instead of convex bounding volumes; thus, there would be no error at all on occlusion. But the methods mentioned in [14], seem to complex to be useable in our case.

Last point on occlusion, it might be very interesting not only to consider each blocker individually, but group the potential occluders in order to generate bigger occluding volumes and that way identify more occlusion. [8] is a nice approach for huge city scenes, and might be adapted to other scenes.

It also seems promising not to systematically compute a shaft for each link. When building the link hierarchy we have to link very small patches together. We think we could use a threshold under which we would not build a new shaft but use the parent one. This would also clearly lead to an important saving in memory, as a shaft is heavy to store (planes and set of occluders).

Finally, it would be interesting to compute an optimal initial cluster subdivision, in order to easily distinguish the different rooms. That way we could distinguish the displacement in a room and in the room next door (or upstairs in a duplex). This should help to update the links in the room where the user is, or where the moved object is (and was), and perform the other links with a background process.

7. REFERENCES

[1] Cyrille Damez and François Sillion. *Space-time hierarchical radiosity. In Dani Lischinski and Greg Ward Larson, editors, Rendering Techniques'99, Eurographics, pages 235-246, Springer-Verlag, Wien New York, 1999. Proc. 10th Eurographics Rendering Workshop, Granada, Spain, June 21-23, 1999.*

[2] George Drettakis and François Sillion. *Interactive update of global illumination using a line-space hierarchy. In TurnerWhitted, editor, SIGGRAPH 97 Conference Proceedings, Annual Conference Series, pages 57-64. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.*

[3] Yann B. Dupuy, Mathias Paulin and René Caubet. *Occlusion evaluation in hierarchical radiosity. WSCG'01 Conference Proceedings, pages (SC)52-59, Plzen, Czech Republic, February 5-9, 2001. ISBN 80-7082-713-0.*

[4] Dave Eberly. *Testing for intersection of convex objects: the method of separating axis. http://www.magic-software.com, 2000.*

[5] 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.*

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

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

[8] Gernot Schaufler, 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), July 2000.*

[9] Franck Schöffel and Andreas Pomi. *Reducing memory requirements for interactive radiosity using movement prediction. 10th Eurographics Workshop on Rendering, Granada, Spain, June 1999.*

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

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

[12] Brian Smits and Henrik Wann Jensen. *Global illumination test scenes. Technical Report UUCS-0-013, Computer Science Department, University of Utah, June 2000. http://www2.cs.utah.edu/~bes/papers/scenes.*

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

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

About the authors

Yann Dupuy is a PhD student, Mathias Paulin is an Associate Professor and René Caubet is the Professor in charge of the Image Synthesis and Virtual Reality staff of the IRIT Laboratory.

IRIT – Université Paul Sabatier
118, route de Narbonne
31062 Toulouse cedex 04
France

E-mail: **ydupuy@irit.fr, paulin@irit.fr, caubet@irit.fr**

(a) Cornell Box

(b) Duplex

(c) Duplex: moving a chair

(d) Four Rooms

(e) Shadow [12]

(f) Shadow: moving two blockers