SIGNAL - IMAGE - COMMUNICATIONS
FRE CNRS n°2731

AN EXPERIMENTAL COMPARISON OF ACCELERATION SCHEMES FOR DENSELY OCCLUDED ENVIRONMENTS

D. FRADIN, D. MENEVEAUX

RAPPORT DE RECHERCHE

n° 2006 - 01

Janvier 2006
SIC, Université de Poitiers - France

SIC, UFR SFA, Université de Poitiers
Bâtiment SP2MI - Téléport 2 - Bvd M. et P. Curie - B.P. 30179 - F - 86962 FUTUROSCOPE CEDEX - FRANCE
Tél. (+33) (0) 5 49 49 65 67 - Fax (+33) (0) 5 49 49 65 70

AN EXPERIMENTAL COMPARISON OF ACCELERATION SCHEMES FOR DENSELY OCCLUDED ENVIRONMENTS

David Fradin Daniel Meneveaux
SIC laboratory
University of Poitiers, France
{*fradin,meneveaux*}*@sic.univ-poitiers.fr*

Images with photon-mapping global illumination (final gathering) for three scenes: Octagon, Soda Hall and L-Building.

Keywords: Densely occluded scenes, acceleration data structures, cells and portals, topology, memory coherent ray shooting, global illumination.

Abstract: This paper presents a comparative study of acceleration schemes for global illumination and ray tracing in large buildings. The system implemented relies on a memory-coherent ray shooting architecture. The scene is subdivided into cells, corresponding either to rooms when the information is available, or to voxels obtained with a uniform grid subdivision. Inside each cell an additional acceleration data structure is placed. Regular grids, multi-grids or kd-trees have been used with various parameters. We discuss the criteria for constructing different acceleration data structures as well as their advantages and drawbacks.

1 Introduction

Ray shooting is often used as a basis for computing realistic-looking synthetic images and/or global illumination. Many applications such as recursive ray tracing strongly rely on ray shooting efficiency. Since the early years of ray tracing, space partitioning methods have been proposed to reduce the number of intersection tests for each ray during scene traversal. For general scenes, the most popular acceleration schemes are based on uniform/hierarchical grids, octrees and kd-trees.

Computing global illumination for large scenes still remains a difficult task for two main reasons. First, the whole scene, including geometry, photometry and radiometry, together with an acceleration structure, is too memory-intensive. Second, an accurate global illumination solution requires several hours of computing time.

In the context of densely occluded indoor scenes, several subdivision methods have been proposed (John M. Airey, 1990; Teller et al., 1994; Meneveaux et al., 1998). These methods subdivide the environment into cells using large occluders. Each cell boundary is thus handled by walls with portals. Intuitively, ray shooting should benefit from this type of subdivision since most rays intersect walls or objects located in a cell. Only a few rays should leave a cell through (relatively small) portals. However, such a topology-based subdivision alone is not sufficient for accelerating ray shooting in the case of large cells which contain a high number of objects.

In this paper, we compare experimentally several schemes for accelerating ray shooting. First, we wish to quantify the advantage of using topology instead of general acceleration schemes which are independent of the scene structure. Second we wish to compare the efficiency of several acceleration data structures placed inside cells.

Since, for complex models, the whole geometry cannot be stored in the memory, ordering strategies have to be implemented. We use a memory-coherent ray tracing approach, relying either on a coarse uniform grid as in (Pharr et al., 1997), or on a list of topology-based cells as in (Fradin et al., 2005).

We provide some results for photon-mapping global illumination (Jensen, 2001). Ray tracing with final gathering was used for generating the images shown on the first page. In addition to ray-object intersection, we are also interested in the preprocessing time required for constructing the data structure.

This paper is organized as follows. Section 2 discusses previous work concerning the acceleration schemes for complex scenes. Section 3 gives details about the scenes used. Section 4 describes the acceleration schemes we compare. Section 5 further details the principle used for memory-coherent ray tracing in complex architectural scenes. Finally, Section 6 discusses the results obtained.

2 Related Work

In (Szirmay-Kalos et al., 2002; Havran, 2000), the advantages and drawbacks of three acceleration schemes are discussed: uniform grids, kd-trees and octrees. According to their study, kd-trees and octrees perform better than uniform grids with sparsely populated scenes since large empty areas are adaptively processed. The authors are essentially interested in ray-object intersection. The time required for constructing the acceleration structure is not discussed.

For complex environments, ray tracing has proven efficient with an appropriate acceleration structure. For instance, in (Wald et al., 2001; Keller and Wald, 2000; Wald et al., 2004), a kd-tree is used for interactively rendering various types of scenes, comprising large buildings. The structure provided by the subdivision is efficiently stored on the disk so that blocks of information can be retrieved every time a new part of the scene is required. The major drawback for constructing effective kd-trees remains in the preprocessing time required. Several hours can be necessary for scenes composed of several millions of triangles.

In (Pharr et al., 1997) a memory-coherent ray tracing approach is proposed, based on a uniform subdivision. Voxels should contain about a thousand polygons. They are processed separately during traversal and thus need not be in the memory. Inside each voxel an acceleration structure is used (e.g. a uniform grid).

For buildings, large occluders, such as walls, considerably reduce the number of objects seen from a given viewpoint or region. Global illumination techniques can benefit from this information, as shown in previous approaches for the radiosity method, with a *binary space partitioning* method (John M. Airey, 1990; Teller et al., 1994) or according to construction rules (Meneveaux et al., 1998). For each resulting cell a visibility graph indicates the set of potentially visible cells, commonly known as the *Potentially Visible Set* or *PVS*.

Cells-and-portals data structures have mainly been used for radiosity. For example, in (Teller et al., 1994), PVS are used for loading only a subpart of the scene. Various ordering strategies have been proposed for reducing computing time and taking into account memory management. In (Fradin et al., 2005), a photon-mapping approach was implemented without PVS for further reducing the number of polygons required in the memory. In each cell, a uniform grid is used for reducing ray-object intersections.

The methods described above were successfully used for global illumination and rendering. Some of them exploit scene topology with cells and portals for reducing visibility computation and the set of polygons stored in the memory. However, as shown in (Pharr et al., 1997), cells can be generated using a regular grid with more generality. In this paper, we propose to compare several acceleration structures for ray shooting and global illumination, with and without topology.

3 Scene Representation

With a *topological* subdivision, cells should correspond to rooms and portals should be doors or windows (Figure 1). When two cells share the same portal, they are considered as *adjacent*. With a uniform subdivision, cells correspond to voxels and portals are delimited by voxel faces.

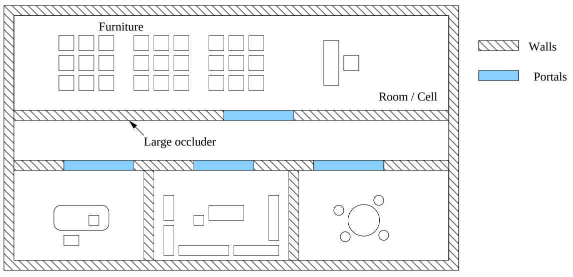

Figure 1: Scene representation with cells and portals.

In our implementation, cells can be created either from a topology-based subdivision or using a uniform grid. Portals are also required for storing rays leaving a cell through a portal during memory-coherent ray shooting. We do not use PVS since the number of triangles contained in one cell can be high and thus also become too memory-intensive.

4 Acceleration Schemes

For tracing rays in a given cell, we used one of the following acceleration structure with various parameters: uniform grids, hierarchical grids or kd-trees.

We are mainly interested in the choice of an acceleration structure to place inside cells for shooting rays. We wish to take into account the computing time for both data structure construction and ray-object intersection.

Ray traversal complexity has already been discussed in details by several authors (MacDonald and Booth, 1990; Havran, 2000). In this paper, we will also briefly discuss the complexity associated with the construction of each data structure.

4.1 Uniform and hierarchical Grids

Uniform grids have many advantages. Construction is fast with a complexity in $\mathcal{O}(n)$ for distributing polygons in the grid voxels, n being the number of polygons. For scene traversal the incremental process is fast since it only requires a few operations.

For complex scenes, this structure can be used for a coarse cell representation such as in memory-coherent ray tracing. Another acceleration structure (for instance another uniform grid) can then be placed inside each voxel for accelerating local ray tracing.

However, in general, objects are far from uniformly distributed. Moreover, finding the appropriate resolution for a given scene is not obvious: the number of objects inside each voxel should not be too high; the number of empty cells should be as low as possible. This is why memory allocation has to be carefully implemented with high resolutions.

A hierarchical representation can thus be used. When a voxel contains too many triangles, it is subdivided again into another uniform grid. This recursive initialization highly depends on the grid resolution and the maximum of triangles stored in a voxel. The construction process has a complexity in $\mathcal{O}(n)$ for each grid level, and thus in $\mathcal{O}(n \log n)$ for the whole hierarchy. During ray traversal, every time a finer grid is reached, an initialization process is required for computing ray-subgrid traversal.

4.2 Kd-Trees

Kd-trees have been successfully used for rendering very complex scenes at interactive frame rates (Wald et al., 2001; Wald et al., 2003). Traversal only requires a few operations for a given ray. It only relies on the intersection between ray and splitting plane for each kd-tree level.

During data structure construction, the chosen splitting plane determines ray traversal efficiency (MacDonald and Booth, 1990; Havran, 2000). We have implemented three strategies:

Spatial-median:
The splitting plane systematically splits the current node into two halves each having the same size. This subdivision method is fast since it only relies on the systematic choice of the splitting plane; geometry contained in each node is only used for distributing objects in the current node and possibly for algorithm termination. The complexity for choosing the splitting plane is in $\mathcal{O}(1)$ for each level and in $\mathcal{O}(\log n)$ for the whole scene. The repartition of triangles in the nodes has a complexity in $\mathcal{O}(n.\log n)$ in the worst case as for the other strategies.

Cost model:
Heuristic based on the number of objects and the volume area of each child. Choosing the best splitting plane with this model requires testing all the vertices contained in the current node (Havran, 2000). Thus, the computing time required for the structure construction is higher than with a spatial-median subdivision. Choosing the splitting requires $\mathcal{O}(n^2)$ operations since for each polygon, the cost function requires testing all the others. Thus, the whole tree construction has a complexity in $\mathcal{O}(n^2.\log n)$.

Bounding box model:
Heuristic based on a quick sort of object bounding boxes in each node. This heuristic also minimizes the number of objects cut by the plane and balances the tree (Szécsi, 2003). Sorting is fast and choosing the best splitting plane is done with respect to the number of objects for each splitting plane. This scheme is a compromise for reducing construction time compared to the cost model and ray traversal time compared to the spatial median strategy. The complexity for sorting the objects with a quick sort and finding the splitting plane is in $\mathcal{O}(n^2)$ in the worst case, but in $\mathcal{O}(n.\log n)$ practically.

4.3 Cells and portals

For large buildings, a subdivision scheme relying on topology seems intuitively better than a brute-force subdivision since large occluders considerably reduce visibility.

With a radiosity approach as proposed in (Teller et al., 1994), every time a cell is loaded into the memory for shooting/gathering light energy, the set of cells corresponding to the PVS has to be loaded into the memory as well. Portals can also be used as interfaces between cells with a memory-coherent approach as proposed in (Fradin et al., 2005). Ideally, such a subdivision should be available from the original model since architects or artists actually *know* the scene structure.

We do not discuss the method used for generating a topological subdivision. It can be provided by a Binary Space Partitioning (or BSP) method (John M. Airey, 1990; Teller et al., 1994), a semi-automatic process (Meneveaux et al., 1998), or using a building modeler as in (Fradin et al., 2005).

5 Program Architecture

Memory coherence is a key point for accelerating ray tracing. Not all the geometric primitives should be checked every time a ray is traced. Instead, groups of rays can be traced within a cell. When a ray gets out of one cell, it is stored and propagated later, when the corresponding cell is processed.

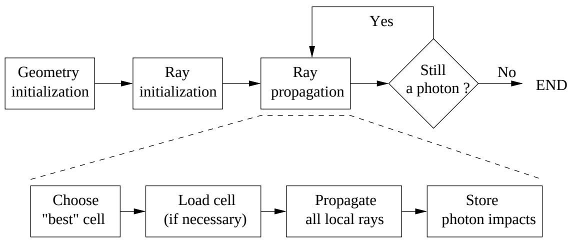

Figure 2: Ray propagation system. *Geometry initialization* corresponds to the construction of an acceleration structure. *Ray initialization* consists in shooting photons from light sources (or primary rays from the camera). Portals are used for propagation in cells.

Figure 2 presents our ray/photon propagation system. The architecture is the same as in (Fradin et al., 2005). The initialization process precomputes acceleration data structure for each cell. It also estimates the size necessary for storing each cell in the memory, including geometry, portals, acceleration structure and so on. For global illumination, photons are first shot from light sources then propagated within the environment.

Photon impacts are stored in several photon maps (one for each cell). Russian Roulette is used to decide whether photons are reflected. When a photon hits an object, it is stored in the photon table of the corresponding cell. Each photon table is partially stored on the disk; when the number of photons stored in the memory is too high, the list is appended into the corresponding file. When a photon hits a portal, it is stopped and stored in a specific data structure. Its path is continued when the corresponding adjacent cell is processed. A least recently used (LRU) scheme is used for memory management of cells.

When photon shooting is complete, photon-maps are constructed for each cell. During ray tracing, the same principle is applied. Rays are shot within the environment, stored at the portals and propagated when the corresponding cells are processed.

6 Results

We used a Dual Intel Xeon 2GHz processor with 2GB of RAM and a 1TB RAID/disk. We used the architecture described above for both global illumination and ray tracing in several large buildings with furnished rooms.

6.1 Scene Description

Both the rendering process and photon propagation method are based on ray shooting with triangulated polygons. Test scenes are illustrated in Figure 3.

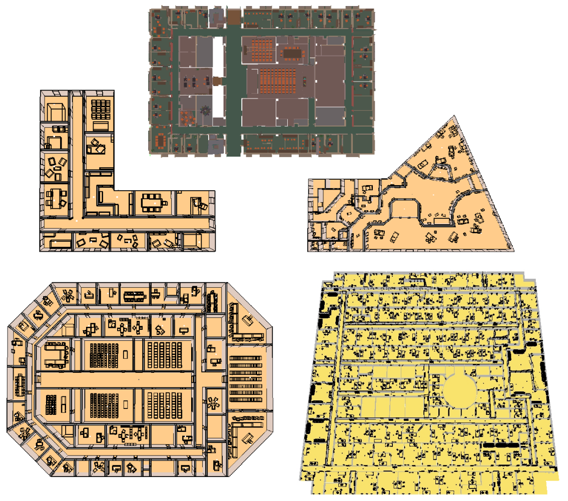

Figure 3: Top: Soda Hall; Top-left: Z-Building. Top-right: L-Shape building. Bottom-left: Octagon building. Bottom-right: Tower101. Furniture has been placed manually in L-Shape / Octagon and automatically in Z-building / Tower101.

Building	Polygons (millions)	Floors	Rooms	Lights	Photons (millions)
L-Shape	0.34	2	27	24	1.2
Z-Building	1.08	1	22	15	0.75
Octagon	5.25	4	232	155	7.75
Soda Hall	1.51	7	250	2.7K	135
Tower_100	1074	101	17.8K	61K	3050

Table 1: Building properties for our test scenes.

Table 1 provides the number of triangles for each building as well as the total number of photons shot. A seed is used for random numbers so that the number of photons stored for all scenes remains the same independently of the used acceleration scheme.

Database information is given in Table 2. The Octagon and Z-building are made up with a few rooms containing a high number of triangles (the most furnished room contains more than 400K triangles). The other buildings contain a high number of smaller rooms. For the Soda Hall building, the subdivision process produces 225 cells (that do not exactly correspond to rooms). Note that the provided database sizes do not include photon maps.

6.2 Cells with Topology

The following results use the memory-coherent approach described above. For scenes with a cells-and-portal subdivision relying on rooms, we have applied several acceleration schemes: uniform grids,

Building	Avg tri. in room	Max tri. in room	Disk space uncompr.	Disk space compr.
L-Shape	12 020	68 400	37.9MB	3.55MB
Z-Building	48 816	217 429	127MB	10.4MB
Octagon	22 621	451 606	624MB	54.8MB
Soda Hall	6688	44 156	199 MB	17 MB
Tower_100	60 134	172 772	110 GB	8.5 GB

Table 2: Database information for our test scenes with (i) average number of triangles in the rooms, (ii) max number of triangle in the rooms (iii) uncompressed database size (iv) compressed database size.

hierarchical grids and kd-trees with various heuristics.

Uniform grids
When associating uniform grids with cells, propagation time still highly depends on the grid resolution which should not be too high or too low. Figure 4 illustrates computing time with respect to resolution for the most complex building we have.

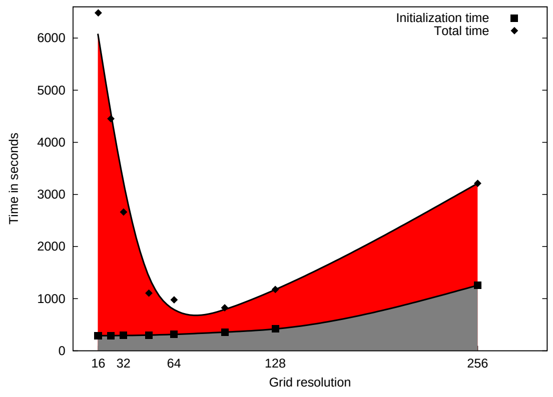

Figure 4: Cumulative time for initializing grids and propagating photons in one floor of Tower101 building with several grid resolutions (50 000 photons shot for each light source), a total of 4.14 billions of photons are stored for 1.07 billion triangles.

When the grid resolution is low, each voxel contains a high number of triangles, intersection tests are more numerous in each voxel and thus more time consuming. With increasing resolutions, triangle lists are smaller but lots of empty voxels are crossed during ray traversal.

For all the tests we made, the curve aspect shown in Figure 4 remains the same independently of the number of rays/photons shot, even though computing time increases.

For each building, the optimal resolution can be different. Figure 5 gives the total computing time of photon-mapping in four buildings at different grid resolutions. The ideal resolution for manually furnished buildings (L-Building and octagon) is close

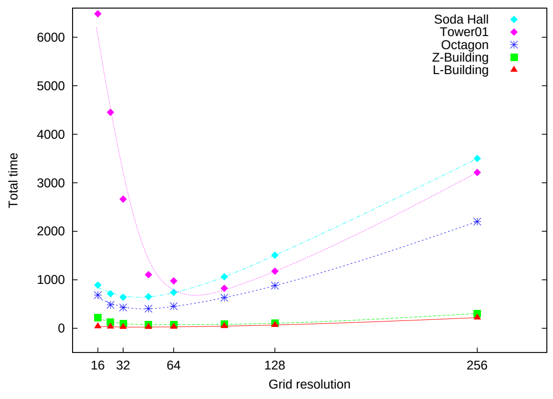

Figure 5: Grid resolution over runtime (in seconds) for four different buildings (50 000 photons by source).

to 48 while for automatically furnished buildings, it is rather close to 96 (Z-Building and Tower). According to our experiments, tests using uniform aggregation of objects do not provide the same type of results than with hand made models.

Hierarchical grids

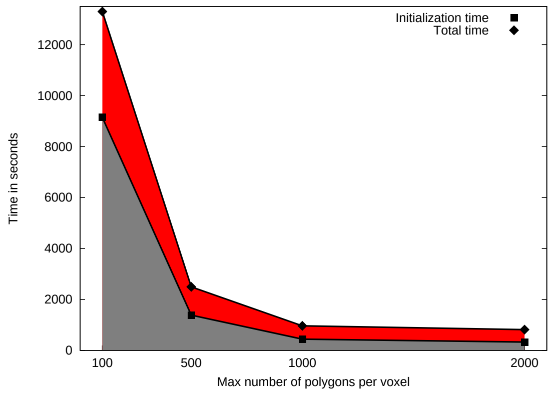

Figure 6: For a given building (first floor of Tower101) and a given grid resolution (64^3 voxels), the maximum number of triangles in each voxel clearly affects the initialization time.

Figure 6 shows that initialization is the critical phase of this acceleration data structure. When the maximum of triangles per voxel is reduced, the hierarchy depth increases, more subgrids are created and the initialization time is higher. Conversely, when increasing this number, hierarchy depth decreases and computing time gets closer to a uniform grid.

Hierarchical grids are advantageous with sparsely populated scenes. When buildings are subdivided using cells and portals, the number of polygons in each cell is not high enough for having an efficient data structure.

Kd-trees
With kd-trees, polygon density in the scene is taken into account more efficiently. Table 3 represents the average depth obtained in cells for each building.

Building	Spatial median	Cost model	Bounding boxes
L-Shape	15,3	26,04	27,63
Z-Building	15,18	22,36	32,73
Octagon	16,73	26,33	28,08
Soda Hall	16,2	26,99	27,80
Tower_100	15,88	28,24	40,26

Table 3: Average depth of kd-trees in cells.

Figure 7 illustrates the total computing time required for global illumination in one floor of Tower101 scene using kd-trees with cells and portals.

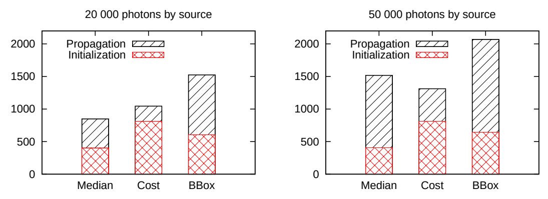

Figure 7: Computing time for photon-shooting in one floor of Tower101 with a kd-tree acceleration data structure.

The bounding box heuristic is a compromise between construction and ray traversal. Object density is taken into account together with tree balancing. Construction is faster than with the cost model heuristic. However, contrary to our expectations, this can be the worst choice with topological cells.

Spatial median cut does not depend on the objects contained in each nodes. This is why data structure construction is very fast. However, the time required for ray traversal is high compared to the cost model heuristic.

Finally, the cost model kd-tree is time consuming to construct. Nevertheless, traversals are much faster because cells are distributed efficiently with respect to objects density. When only a few photons are shot, construction remains too high compared to scene traversal.

Note that depending on the scene and the number of rays to propagate, the *best* heuristic is not systematically the same. With 20 000 photons, spatial median plane heuristic is faster because of the initialization phase. When the number of photons increases, the cost model heuristic is definitely more efficient. Figure 8 shows that depending on the scene structure and on the number of rays / photons, the heuristic providing the lowest computing time is not systematically the same.

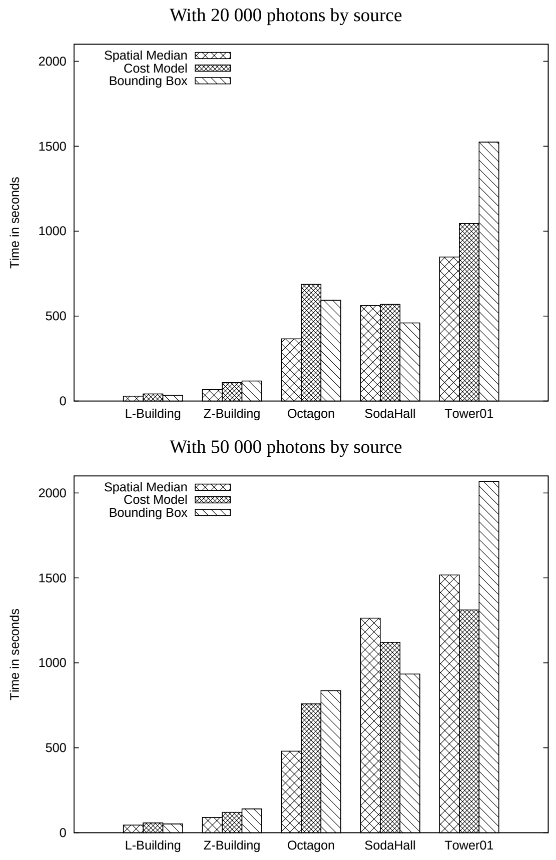

Figure 8: Total computing time obtained for global illumination in four buildings according to splitting plane choice heuristics. (top) with 20K photons for each light source; (bottom) with 50K photons for each light source.

Figure 9 provides a comparison between the best results for each acceleration structure. With topological cells, a regular grid always provides the shortest overall computing time (including structure construction).

6.3 Cells with Uniform Grids

For the following results, cells are constructed using a uniform subdivision, without taking large occluders into account.

We made some experiments with various resolutions. Table 4 gives only results for the best parameters. For global illumination with a photon-shooting approach, low resolution grids remain more efficient. For example, a grid with a resolution of 64^3 creates more than 260K cells; the high number of cells, portals and photons to propagate decreases dramatically the algorithm performances.

Note that large triangles are referenced in several cells; the triangle repartition of each cell is not uniform. A kd-tree is thus more efficient in this con-

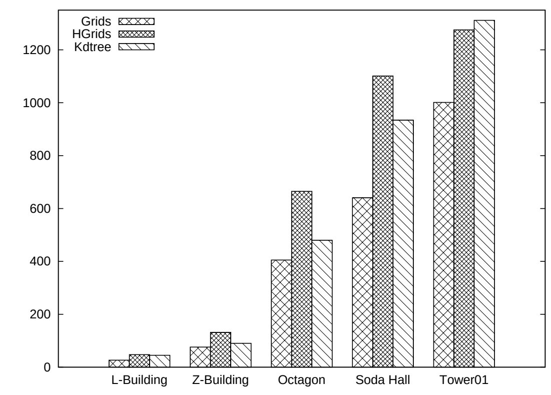

Figure 9: Computing time for each acceleration structure associated with a topological subdivision, 50 000 photons are shot for each light source source.

Building	RG / Cells	Avg # tri	Grids	H-Grids	Kd
L-Building	10 / 300	1159	113	210	46
Octagon	16 / 768	7206	935	1101	593
Soda Hall	16 / 1760	978	1575	2760	928

Table 4: Best computing time (in seconds) for photon-shooting including acceleration structure construction. Cells are constructing using a uniform subdivision. *RG* corresponds to the grid resolution for cells. *Cells* correspond to the number of cells generated for the given grid resolution.

text (Table 4). The bounding-box model is the most suitable since the uniform cells subdivision creates large gaps between triangles (Figure 10) and the time required for constructing the structure remains short compared to the cost model.

6.4 Rendering

Once photon propagation is complete, images can be generated with or without final gathering. Photon maps are constructed from tables of photons in each cell (Jensen, 2001). This process is very fast with less than a second for each cell. For instance, in the Soda Hall scene, this operation lasts 6 min 46 for shooting 1.35 billion photons in 1760 cells.

Primary rays are shot, from the viewpoint. For each intersection point I_p, direct and indirect illumination is estimated. For indirect illumination, we made some tests with (i) the photons nearest to I_p; and (ii) a final gather approach.

Since initialization has already been computed, the kd-tree data structure remains the most effective with final gather, especially with the cost model heuristic. Nevertheless, contrary to photon propagation, the computing time of each structure is not extremely different since much time is spent for searching photons in the photon-maps. Ray shooting and photon search represent between 90% and 95% of total computing

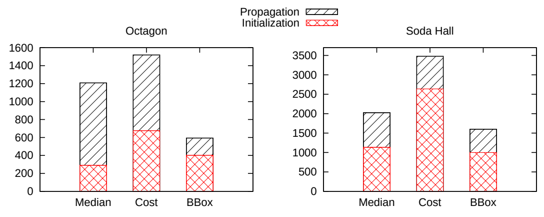

Figure 10: Results obtained with the 3 heuristics used for splitting plane choice.

time. Alone, photon search represents at least 70% of time for generating one image.

Eventually, with a topological subdivision, the image generation is at least 4 times faster than with a uniform cell subdivision since portals are much smaller and only a few rays cross several voxels before intersecting a face. In practice, less than 5% of rays hit portals.

6.5 Comparison

Generally speaking, for global illumination and ray tracing, a topological cell representation is preferable (when possible) since cells are delimited by large occluders (Figure 11). Rays or photons only leave cells through small portals, favoring data locality.

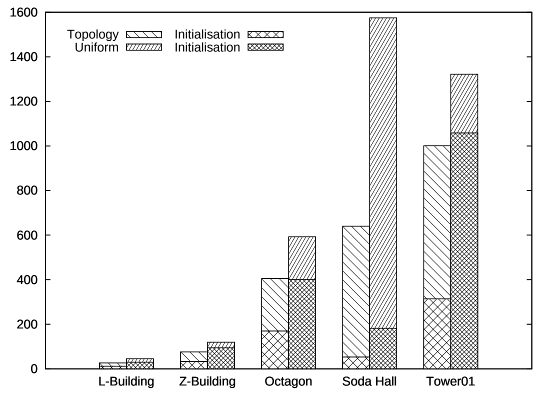

Figure 11: Comparison between smallest computing time: uniform cells vs. topology-based cells.

With topological cells, regular grids have provided the best overall computing time (including data structure construction), even compared to kd-trees. However, kd-trees perform better for ray propagation since the scene subdivision is more adapted to objects density. Generally-speaking, kd-trees would be preferable when a high number of rays have to be shot. For instance, if a huge number of photons require to be shot in a large building during global illumination - even with 4 billion photons shot in the Tower101 building, regular grids still remain faster; if many images with final gather have to be produced for the same scene or with interactive ray tracing as in (Wald et al., 2001).

When cells are produced with a uniform subdivision, the most efficient acceleration data structure is a kd-tree.

7 Conclusions

Depending on the application, the time required for constructing the acceleration data structure can have to be taken into account, especially for very large scenes. For interactive walkthrough, this preprocessing can be as long as necessary since the aim is to produce high quality images as fast as possible. However, for global illumination in scenes composed of several million triangles, the overall computing time including scene preprocessing has to be as small as possible.

Automatically generated scenes have specific properties different from the scenes manually produced. According to the tests we made, concluding about one algorithm efficiency with only such scenes is hazardous and might provide surprisingly different results when applied to realistic scenes.

A difficult problem concerns the best parameters to choose depending on the scene. As shown in the previous sections, the uniform grid resolution cannot be chosen automatically. Several tests have to be performed. Moreover, depending on the application, parameters can be quite different. In the case of large buildings, it would be interesting to classify the acceleration data structure using the number of triangles and an estimation of the number of rays to cast. Then, for each cell, a specific acceleration structure can be employed.

When possible, large occluders should be taken into account for subdividing the scene into cells. With such a subdivision, cells favor data locality, the number of cells is lower and thus disk accesses are reduced. Consequently, the choice of an acceleration data structure is easier. A regular grid with a resolution close to 64^3 always produces good results.

When only a few images have to be computed, the whole scene does not require to be processed. A *lazy* acceleration structure can also be used (Poitou et al., 2000) for avoiding the complete scene subdivision.

REFERENCES

Fradin, D., Meneveaux, D., and Horna, S. (2005). Out-of-core photon-mapping for large buldings. In *Rendering Techniques, Proceedings of Eurographics Symposium on Rendering*, pages 65–72.

Havran, V. (2000). *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Czech Technical University in Prague.

Jensen, H. (2001). *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd.

John M. Airey, John H. Rohlf, F. P. B. (1990). Towards image realism with interactive update rates in complex virtual building environments. *ACM Siggraph*, pages 41–50.

Keller, A. and Wald, I. (2000). Efficient Importance Sampling Techniques for the Photon Map. In *Vision Modelling and Visualization 2000*, pages 271–279.

MacDonald, D. J. and Booth, K. S. (1990). Heuristics for ray tracing using space subdivision. *Vis. Comput.*, 6(3):153–166.

Meneveaux, D., Maisel, E., and Bouatouch, K. (1998). A new partitioning method for architectural environments. *To appear in Journal of Visualization and Computer Animation*.

Pharr, M., Kolb, C., Gershbein, R., and Hanrahan, P. (1997). Rendering complex scenes with memory-coherent ray tracing. In *Computer Graphics (SIGGRAPH 97 Conference Proceedings)*, pages 101–108. ACM.

Poitou, O., Bermes, S., and Lecussan, B. (2000). Laziness, a way to improve distributed computation of the ray tracing algorithm. In *WSCG*.

Szécsi, L. (2003). An effective implementation of the k-d tree. In *Graphics programming methods*, pages 315–326, Rockland, MA, USA. Charles River Media, Inc.

Szirmay-Kalos, L., Havran, V., Balazs, B., and Szecsi, L. (2002). On the efficiency of ray-shooting acceleration schemes. In *Proceedings of SCCG'02 conference, Budmerice, Slovakia*, pages 89–98. ACM SIGGRAPH.

Teller, S., Fowler, C., Funkhouser, T., and Hanrahan, P. (1994). Partitioning and ordering large radiosity computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 443–450.

Wald, I., Benthin, C., Wagner, M., and Slusallek, P. (2001). Interactive rendering with coherent ray tracing. In Chalmers, A. and Rhyne, T.-M., editors, *Computer Graphics Forum (Proceedings of Eurographics)*. Blackwell Publishers, Oxford.

Wald, I., Dietrich, A., and Slusallek, P. (2004). An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Proceedings of Eurographics symposium on Rendering*, pages 81–92.

Wald, I., Purcell, T. J., Schmittler, J., Benthin, C., and Slusallek, P. (2003). Realtime Ray Tracing and its use for Interactive Global Illumination. In *Eurographics State of the Art Reports*.