

# Memory Management Schemes for Radiosity Computation in Complex Environments

Daniel Meneveaux, Kadi Bouatouch, Eric Maisel

► **To cite this version:**

Daniel Meneveaux, Kadi Bouatouch, Eric Maisel. Memory Management Schemes for Radiosity Computation in Complex Environments. [Research Report] RR-3149, INRIA. 1997. <inria-00073540>

**HAL Id: inria-00073540**

**<https://hal.inria.fr/inria-00073540>**

Submitted on 24 May 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

*Memory Management Schemes  
for Radiosity Computation in Complex  
Environments*

Daniel Meneveaux, Kadi Bouatouch and Eric Maisel

**N° 3149**

Avril 1997

\_\_\_\_\_ THÈME 3 \_\_\_\_\_



*Rapport  
de recherche*







## Memory Management Schemes for Radiosity Computation in Complex Environments

Daniel Meneveaux, Kadi Bouatouch and Eric Maisel

Thème 3 — Interaction homme-machine,  
images, données, connaissances  
Projet SIAMES

Rapport de recherche n° 3149 — Avril 1997 — 18 pages

**Abstract:** Hierarchical radiosity is a very demanding process in terms of computation time and memory resources even for scenes of moderate complexity. To handle complex environments, not fitting in memory, new solutions have to be devised. One solution is to partition the scene into subsets of polygons (3D cells or clusters) and to maintain in memory only some of them. The radiosity computation is performed only for this resident subset which changes during the resolution process. This change entails many read and write operations from or onto the disk. These disk transfers must be ordered to make the radiosity algorithms tractable. We propose in this paper different ordering strategies which can be seen as complementary to those devised by Teller [1].

**Key-words:** Computer graphics, Complex scenes, Radiosity method, Architectural environments

*(Résumé : tsvp)*

# Méthodes de Gestion de Mémoire pour le Calcul de Radiosité dans des Environnements Complexes

## Résumé :

La radiosité hiérarchique est un processus très coûteux en terme de temps de calcul et de ressources mémoire, même pour des scènes de complexité moyenne. Pour traiter des environnements complexes ne tenant pas en mémoire, de nouvelles solutions sont à envisager. L'une d'entre elle consiste à structurer la scène en sous-ensembles de polygones et ne conserver en mémoire qu'une partie d'entre eux. Les calculs de radiosité sont effectués seulement pour ces sous-ensembles résidant en mémoire et pouvant changer au cours du temps. Ces changements provoquent de nombreuses opérations de lecture et d'écriture sur disque. Cependant, il est possible de réduire les temps de calculs en ordonnant efficacement ces transferts de données. Nous proposons dans ce rapport différentes stratégies d'ordonnement pouvant être considérées comme complémentaires à celles décrites par Teller dans [1].

**Mots-clé :** Synthèse d'images, Scènes complexes, Radiosité, Environnements architecturaux

## 1 Introduction

Hierarchical radiosity provides high level realistic images at the expense of important computation and memory resources, even for scenes of moderate complexity. This is due to the need of meshing surfaces into elements and linking these latter one to another. In other words, the number of surface elements and links increases drastically with the precision required. This is why performing lighting simulation becomes tricky for complex scenes such as buildings containing millions of geometric primitives. Indeed, visibility relationships between surface elements have to be performed, which entails a very important computing time. However, many visibility computations are useless such as those between a polygon and some objects occluded by a large polygon like a wall. This problem has already been addressed in [2, 3, 4, 5] where the authors propose to partition the scene into subsets of polygons (called *cells*) before performing radiosity computations whether sequentially [1] or in parallel [6].

Another problem is due to the fact that the amount of data needed for radiosity computations (for very complex scenes) is so important that they cannot fit in memory. One solution to this problem is to partition the scene into cells so that only a subset of cells is maintained in memory for illumination computations. This is made possible with the help of visibility relationships between these cells. Indeed, a polygon in a subset  $C$  shoots/gathers energy to/from all the objects lying in  $C$  as well as those visible through the holes (also termed *portals*) in its boundary like windows or doors for buildings interiors. Thus, one needs to load in memory the cells (stored on the disk) concerned with the radiosity computations, to withdraw the non-necessary cells from memory and move them back to the disk. This operation requires many disk transfers which are very costly in terms of computation time. These transfers must be ordered so as to reduce expensive read and write operations (from/onto the disk or memory).

In this paper, several ordering strategies, different from the ones described in [1], are proposed. These strategies make use of the cells resulting from the scene partitioning as well as a graph expressing visibility between these cells.

The organization of this paper is as follows. Previous works are first summarized then followed by an overview of our algorithm. Next, our ordering strategies are described in detail. Finally, results of experiments are provided to demonstrate the usefulness of these kinds of strategy.

## 2 Previous Works

A few works address the problem of photorealistic rendering of complex scenes containing millions of objects. To our knowledge, Teller [1] is one of the first author who proposed an algorithm for solving this problem. His objective is to use radiosity for global illumination computations and a gathering scheme for the resolution. To this end, his approach consists in subdividing the scene into 3D cells with the help of a BSP-based partitioning technique. This latter allows to efficiently compute visibility between clusters. A datastructure is

associated with each visibility relationship between two visible clusters. This datastructure contains: a list of blockers, a list of links between the surfaces of both clusters, convex hulls of clusters... These data enable each cluster to gather energy from the other clusters visible from it.

The datastructures resulting from the BSP partitioning, clustering and visibility calculations are stored on disk. Only a small portion of these data reside in memory for radiosity computations. As the algorithm utilizes gathering, it gathers energy impinging on each cluster. To this end, it loads in memory a receiving cluster  $R$  as well as one of the visible cluster  $S$ . Afterwards  $R$  gathers energy from  $S$ . This operation is repeated for all the other clusters visible from  $R$ . Each source cluster  $S$  modified by the gathering operation (creation of new surface elements) is saved on disk. Despite the use of the disk cache, these disk transfers remain too important. This is why Teller proposes several heuristics for efficiently ordering these disk transfers so that the contents of the cache changes slowly. These heuristics are:

- **Random order:** Gathering is randomly performed.
- **Model order:** Clusters gather energy in the order in which they were instantiated by the modeler.
- **Source order:** Select the receiving cluster which has most often acted as a source.
- **Cell order:** Schedules clusters by traversing the cells of the BSP tree. In this way consecutive cells are chosen by selecting the neighbor cell seen from the largest portal. This approach exploits the visibility coherence of clusters due to proximity and local intervisibility.

From the experiments done by Teller, the fourth approach which exploits the result of the BSP partitioning method, seems giving the best results.

A parallel algorithm for radiosity computation in large environments has been proposed by Funkhouser [6]. This algorithm makes use of the same datastructures as those proposed by Teller. The scene is organized in groups of clusters which are distributed by the host processor to the slave processors. Each slave processor is in charge of computing the radiosity solution for the group it is responsible for. Funkhouser proposes two different strategies for load balancing giving interesting speed-ups.

### 3 Overview

Our algorithm handles complex scenes such as buildings interiors made up of polygonal objects. A preprocessing step partitions the scene into subsets called *cells*, and a graph expressing visibility relationships between these cells is built [5]. In this visibility graph, a vertex represents a cell and an edge between two vertices  $A$  and  $B$  means that objects within  $A$  see other objects within  $B$  and vice versa. This visibility graph together with the obtained cells are used by our ordering strategies for radiosity computations. Our radiosity method relies on projection techniques with wavelets as basis functions and offers

two resolution schemes: gathering and shooting. In our implementation, we use shooting because of its ability to quickly provide useful images at the early stages of the resolution and to interactively walk through the scene while the radiosity computations are being performed. This is made possible even for complex environments containing thousands of polygons thanks to the visibility graph.

The main contribution of our work is concerned with new ordering strategies. Ordering has already been addressed by Teller [1]. As seen in the previous section, his approach is based on a precise but complex and very costly visibility preprocessing involving clusters, tubes, blockers... In addition, in [1] the ordering techniques consist in loading into memory a cluster, the clusters visible from it as well as potential blockers. Unlike Teller, our ordering strategies rely on very simple visibility computations between cells instead of clusters. Indeed, the method we use to compute visibility between two cells is due to Airey [2] and consists in tracing rays through the portals separating these cells.

Another difference with Teller's work is that our ordering algorithms allows to load into memory a cell  $C$  (instead of a cluster) together with those cells  $C_i$  visible from  $C$ . Once these cells have been loaded into memory, the surface elements within  $C$  shoot their energy towards the ones within the cells  $C_i$ . To sum up, our ordering methods differ from those proposed by Teller in the sense that they use other strategies and handle different datastructures.

As said in the previous section, Teller's method relies on an optimal use of the disk cache. However with this approach the user has no control on the cache, which is a serious problem in case of multiuser computer. For example one cannot impose maintaining in memory certain data necessary for the current radiosity computation. This is why one contribution of our work is to manage the memory (destinated to store the data corresponding to the cells) as a software cache. The management of this software cache is handled by our ordering strategies which predict for the short and medium range terms the disk transfer costs that determine the choice of the cells that will shoot their energy. These strategies rely on discrete optimisation techniques that explicitly account for the actual disk transfers. We do not claim that our strategies are better but just complementary to those proposed by Teller.

Our ordering strategies are part of a global illumination software package as shown in figure 1.

Let us recall that our resolution scheme is shooting. A cell whose surfaces are selected for shooting their energy will be called *shooting cell* from now on. The role of the ordering module is to select a shooting cell as well as those visible from it, with the help of one of the ordering strategies (which will be described later on) depending on the cells already in memory and the visibility graph. A linear array, termed *Cell\_In\_Memory[]* points out whether a cell is already in memory or not and if it will be used by the radiosity module. Note that a cell is used for radiosity computations if and only if it has been selected by the partitioning module. Once the cells have been selected by the ordering module, the radiosity module is invoked.

Our global illumination algorithm is given in figure 2. In one iteration (called ITERATION in this algorithm), all the cells are in turn selected as shooting cell. During the first iteration, only the light sources shoot their energy so as to get results very early.



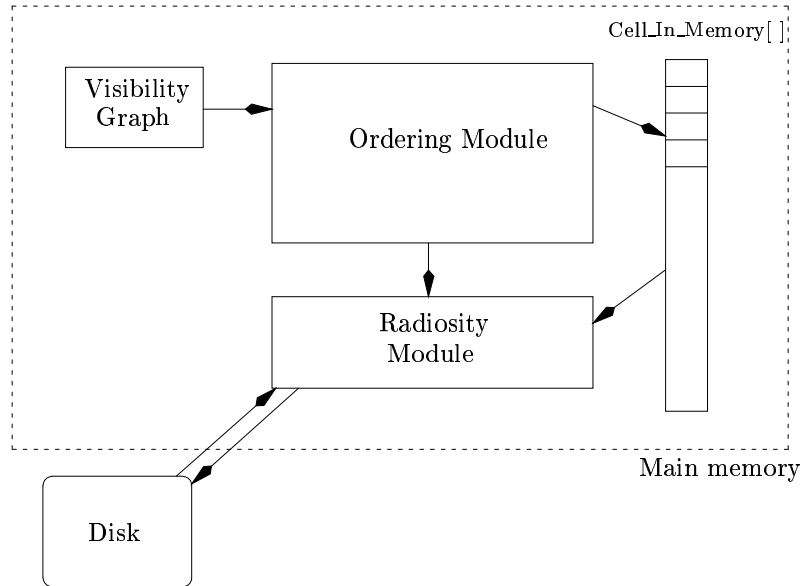


Figure 1: System architecture

The function *Choose\_Cell()* is no more than the ordering module. It corresponds to the implementation of one of the ordering strategies we have developed. As for the procedure *Shoot\_From\_Cell()*, it allows each surface of the selected cell  $C$  to shoot its energy towards the surfaces within  $C$  as well as those belonging to the cells visible from  $C$ .

The next section describes in detail our ordering strategies.

## 4 Ordering

### 4.1 Principle

Let us recall that the objective of this work is to compute global illumination in very large environments with the radiosity method. As in [1], to meet this goal, the scene is partitioned into 3D cells and a graph expressing visibility between these cells is built. This graph allows to maintain a subset of cells in memory for which the radiosity computation is performed. Radiosity computation consists in choosing a set of cells, selecting one of them for shooting its energy to the others belonging to the same set. This process is repeated for other sets of cells till convergence. It needs to load in memory the cells (stored on the disk) concerned with the radiosity computations, to withdraw certain non-necessary cells from memory and move them back to the disk. This operation requires many disk accesses, which is very costly in terms of computation time. These disk accesses correspond to the transfer of cells

```

global_illumination() {
    typedef struct CELL_FLAGS {
        boolean In_Memory;
        boolean To_Be_Used;
    } ;

    Integer C;          /* Cell identifier */
    CELL_FLAGS Cell_In_Memory[number_of_cells];
    ITERATION = 0;
    /* VG is the visibility graph */
    VG = Read_Visibility_Graph_From_Disk();
    Initialize(Cell_In_Memory[ ]);
    While not (convergence) AND (ITERATION < Max_Iterations) {
        Unmark_All_The_Cells();
        /* Choose the first shooting cell */
        C = Choose_Cell(VG, ITERATION, Cell_In_Memory);
        /* Choose_Cell() modifies Cell_In_Memory[ ] */
        Mark_Cell(C);
        /* The cell C is marked during the current iteration */
        while not (all the cells are marked) {
            /* The cell C shoots now its energy */
            Shoot_From_Cell(C, Cell_In_Memory);
            /* Choose a shooting cell which has not already */
            /* shot its energy during this iteration */
            C = Choose_Cell(VG, ITERATION, Cell_In_Memory);
            Mark_Cell(C);
        }
        ITERATION = ITERATION + 1;
    }
}

```

Figure 2: General algorithm

between the memory and the disk. These partitioned transfers must be ordered so as to reduce expensive read and write operations (from/onto the disk or memory). To be effective, an ordering algorithm must schedule successive shootings so as to minimize disk accesses, while maintaining rapid convergence properties. Unfortunately, this problem is equivalent to finding a solution to the traveling salesman problem which is more complex in our case because the database changes dynamically (creation of new links, new elements due to the hierarchical radiosity approach).

Our ordering strategies operate according to the algorithm given in figure 3. Whenever a cell is chosen for shooting its energy, it has to be in memory as well as those visible from it. In case one of these cells is not already in memory, it is read from the disk, implying then memory/disk transfers whose cost is evaluated in terms of input surfaces or cells depending on the used ordering strategy. This cost is evaluated by the function `evaluate_cost()`.

```

int Choose_Cell(VG,ITERATION,Cell_In_Memory)
{
    int cost_min, cost;
    cost_min = HUGE;
    for each cell  $C_i$  {
        /* Find the set VS of cells visible from  $C_i$  */
        VS = find_visible_Set( $C_i$ ,VG);
        cost = Evaluate_Cost( $C_i$ ,VS,Cell_In_Memory)
        if ( cost < cost_min ) {
            cost_min = cost;
            result =  $C_i$ ;
        }
    }
    Update(VG, $C_i$ ,Cell_In_Memory);
    return result;
}

```

Figure 3: General ordering algorithm

`Update()` is a procedure which sets to TRUE the fields *In\_Memory* and *To\_Be\_Used* of the datastructures *CELL\_FLAGS* (see figure 2) associated with the cells in VS (see figure 3). It also sets to FALSE only the field *To\_Be\_Used* associated with cells not visible from the selected shooting cell but already in memory.

In our implementation, we have limited to a value  $M_{limit}$  the size of the memory used for storing the cells, which corresponds to a certain number of cells that varies during the radiosity computation process. We have chosen a value proportional to the maximum number of input surfaces that can be stored in memory. We have preferred to make the limitations of memory size rely on the number of surfaces rather than the actual memory size (that closely depend on the number of surface elements and links) for two reasons. The first one is that as the number of surface elements as well as links can increase during the resolution process, the memory size they need for their storage can go beyond the authorized size  $M_{limit}$ . The second reason is since the number of surface elements and links has a complexity of  $O(n \log n)$ ,  $n$  being the number of input surfaces, we can a priori estimate  $M_{limit}$ , which is more reliable. This estimation prevents from exceeding  $M_{limit}$ . For the same reasons, for some of our ordering strategies the disk transfers cost is evaluated in terms

of number of input surfaces. For the other strategies the *cost* corresponds to the number of cells to be transferred from and onto the disk.

## 4.2 Strategies

### 4.2.1 Random method

This method will serve as reference to the other strategies described hereafter. The shooting cell is randomly chosen then loaded in memory together with the cells that are visible from it. This strategy does not account for the cost of disk transfers.

### 4.2.2 Greedy method *S*

The objective is to reduce the cost of disk transfers expressed in terms of number of input surfaces. Each cell is considered as a potential shooting cell and its associated cost is evaluated. This cost corresponds to the possible transfer from the disk of this cell and its associated visible set. This method does not take care of the convergence of the radiosity method.

### 4.2.3 Greedy method *C*

Due to the above remark, this strategy is similar to the previous one except the fact that the cost is evaluated in terms of number of cells.

### 4.2.4 The traveling salesman strategy

As ordering is equivalent to the traveling salesman problem, we propose an algorithm for solving this problem.

Using the visibility graph we build another graph which is oriented and valuated. In this new graph *OVG* a vertex corresponds to a cell and with each pair of cells (*C1*, *C2*) is associated an oriented edge. The value attached to an edge (*C1*, *C2*) is the cost due to the cell *C2* (in terms of number of surfaces) in case *C2* would be chosen as shooting cell. Note that the value of an edge corresponds only to the cost of reading cells from disk. This is due to the fact that since the construction of *OVG* is performed before the radiosity computation, we cannot foresee which cells, already in memory, have to be moved back to disk.

The graph *OVG* is used by the traveling salesman strategy which is given in figure 4.

The function `Choose_Min_Edge()` selects the unmarked edge of minimum value. `No_Cycle()` returns *TRUE* if the successive edges do not form a cycle. If the edges within *GA* form a cycle containing all the vertices of *OVG* then `Complete_Cycle()` returns *TRUE*.

```

Cell_List TravelingSalesman(Graph OVG) /* OVG is the oriented-valuated graph */
{
  Cell_List GA = NULL; /* List of edges representing the best travel */
  Unmark_All_Edges(OVG);
  While not complete {
    foreach edge A of OVG {
      Min = Choose_Min_Edge(A);
      Mark(Min);
      if ( No_Cycle(Min, GA, OVG) ) {
        Add_Edge_To_List(Min,GA);
      }
      else if ( Complete_Cycle(Min, GA, OVG) ) {
        Add_Edge_To_List(Min,GA);
        complete = TRUE;
      }
    }
  }
  return GA;
}

```

Figure 4: Traveling salesman algorithm

#### 4.2.5 Backtracking $S$

Recall that the greedy algorithms select one shooting cell at each step of the resolution process. As this choice cannot be reconsidered, this can make the ordering less efficient. One solution is to select, among all the lists of  $N$  successive shooting cells, the one entailing the lower cost. This amounts to predict, at one go,  $N$  best shooting cells. This solution is provided by the backtracking algorithm described hereafter.

Given a set of cells  $CELL$ , the objective is to find out a list of  $N$  cells which will be successively selected as shooting cell. To this end we select a first cell  $C_0$  as shooting cell. Next we determine the subsequent shooting cells:  $C_1 \in \{CELL - C_0\}$ ,  $C_2 \in \{CELL - \{C_0, C_1\}\}$ , ... ,  $C_{N-1} \in \{CELL - \{C_0, C_1, \dots, C_{N-2}\}\}$ . All the possibilities for choosing these  $N$  cells are represented by a tree (having a depth  $N$ ) whose root is  $C_0$  and vertices the cells. In this tree the value associated with each edge  $(C_i, C_j)$  represents the cost due to the disk transfers when  $C_j$  is selected as shooting cell just after  $C_i$ . The backtracking strategy consists in traversing this tree in order to determine the path of depth  $N$  and of minimum cost. The cost is evaluated in terms of number of input surfaces.

The backtracking algorithm, given in figure 5, returns such a list of  $N$  cells stored in the linear array `Chosen_Cells[ ]`. The function `Choose_Cell()` corresponding to this backtracking

strategy returns successively each element of this array. Once all the elements have been returned except `Chosen_Cells[ Max_Depth - 1]`, backtrack is again invoked with `C_First = Chosen_Cells[Max_Depth]`. This process is repeated till all the cells have been selected as shooting cell during one iteration.

Note that  $N$  is called *Max\_Depth* in this algorithm.

```

int Backtrack(Current_Cell, Cost, *Cost_Min, Depth, Max_Depth, Chosen_Cells[ ]) {
    /* Current_Cell: the current cell */
    /* Cost = cost due to the path starting at a cell with depth 0 */
    /* and ending at the current cell */
    /* Cost_Min : minimum value of the costs associated with */
    /* the paths of Maximum depth (Max_Depth) previously determined */
    integer Current_Cost_Min = HUGE;
    /* Current_Cost_Min : minimum value of the costs associated with */
    /* the paths starting at the current cell and ending at the deepest cells */
    integer Current_Cost;
    /* Current_Cost : cost due the path starting at the current cell */
    /* and ending at the leaves */
    /* Prune the paths whose cost is already greater than Cost_Min */
    if ( Cost > *Cost_Min ) return HUGE;
    if ( Depth == Max_Depth ) {
        if ( Cost < *Cost_Min ) *Cost_Min = Cost;
        return 0;
    }
    for each unmarked cell C {
        Current_Cost = Evaluate_Cost(Current_Cell, C);
        mark(C);
        Current_Cost += Backtrack(C, Cost, &Cost_Min, Depth + 1,
                                   Depth_Max, Chosen_Cells[ ]);
        if ( Current_Cost < Current_Cost_Min ) {
            Current_Cost_Min = Current_Cost;
            Chosen_Cells[Depth] = C;
        }
    }
    unmark(C);
    return Current_Cost_Min;
}

```

Figure 5: Backtracking algorithm

#### 4.2.6 Backtracking $C$

This strategy is similar to the previous one except the cost is evaluated in terms of number of cells to transfer from the disk.

#### 4.2.7 Max energy

The objective of this strategy is to speed-up the convergence to the radiosity solution. With this aim in view, the cell with maximum unshot energy is selected as shooting cell. Note that this method does not worry about the cost due to disk transfers.

### 4.3 Remark

The above strategies rely on either the cost due to disk transfers or the unshot energy. It turns out that combining these two criteria is a very difficult task.

## 5 Implementation and results

### 5.1 Implementation

In a preprocessing step the scene is partitioned into 3D cells. Each cell is stored on disk in a file containing a list of input surfaces, reflectances, spectral power and intensity distributions of the light sources. The visibility graph is also saved in a file containing the description of each cell. Indeed, in this file a cell is described by the following data: (i) the name of the file containing the cell, (ii) the number of the cell, (iii) the list of cells visible from it (the visible set VS).

The radiosity computation starts once this partitioning has been completed. During this computation the contents of these cells is modified to include the created surface elements and associated links. Whenever a modified cell is saved on disk its corresponding file is also modified to contain the new data.

Each surface element is identified by its number and the number of the cell containing it. This identification makes possible read and write (from or onto disk) of links between surface elements belonging to different cells.

An option is offered to the user for interactively rendering the scene at the intermediate stages of the resolution while pursuing radiosity computations. To do this, two processes are resident in memory: the first one performs the radiosity computations while the second is in charge of rendering. As soon as a cell has completed its shooting a *UNIX signal* is sent to the rendering process which renders the modified scene.

Another possibility of our global illumination algorithm (figure 2) is to stop the calculations after a number of iterations fixed by the user and to save the cells in memory which are in a certain state. Later on, we can restart, from this state, the program to continue the radiosity calculations till convergence.

Disk transfers involve memory allocations and releases which are managed by the UNIX operating system. As we have no control over the garbage collector incorporated in the UNIX system, we decided to manage memory by ourselves. Experiments have shown that the memory management offered by UNIX is not adapted to our application. Indeed, we have ascertained many swaps whereas the data resident in memory did not require any swap.

## 5.2 Results

The seven ordering strategies we have explained in the previous section have been applied to different scenes. This section provides some results only for one scene made up of 14148 input surfaces among which about 400 are light sources. This scene has been partitioned into 214 cells.

In order to compare the different strategies in terms of disk accesses only, the radiosity computations have been performed with a low precision (implying a very coarse meshing) so that the associated computing time is negligible compared to that of disk accesses.

The obtained results are shown in figure 7 for the seven ordering strategies. For each strategy there are three plots. The first one provides the number of surfaces resident in memory at the  $i^{th}$  iteration (red plot). The second gives the number of surfaces belonging to the current shooting cell and to the cells visible from it (green plot). The third one gives the cost of disk transfers expressed in terms of number of input surfaces for the  $i^{th}$  selected shooting cell (blue plot).

This figure shows that the *Random* and *Max\_Energy* methods involve a more important amount of disk transfers. This result is foreseeable since these latter are not accounted for by these methods.

The computing times corresponding to the ordering strategies are given by table 1.

Random Method	Greedy S	Greedy C	Traveling Salesman	Backtracking S	Backtracking C	Max Energy
229 mn	161 mn	140 mn	130 mn	1378 mn	975 mn	280 mn

Table 1: Computing time for the seven ordering strategies

Many conclusions can be drawn from these results. First of all, it is preferable to make use of ordering strategies relying on disk transfers rather than on energy to be shot. In the same way, the strategies based on the cell transfers are more efficient than those based on the number of surfaces. This can be explained as follows. When evaluating the disk transfer cost in terms of number of surfaces, the cells made up of many surfaces are selected only at the end of the current iteration. As these cells may not be linked by an edge of the visibility graph, the selection of one of them would imply loading the cells visible from it. Consequently, the number of disk transfers increases and so does the computing time. If the transfer costs are evaluated in terms of number of cells, a cell can be selected as shooting cell and then loaded into memory, not necessarily at the end of the iteration, even though it



contains many surfaces. Thus, expressing transfer costs in terms of number of cells is more efficient because spatial coherence is fully exploited.

So far, we have experimented with our ordering strategies for a few complex scenes because of the very important computing time needed. From these experiments, we have ascertained that the traveling salesman algorithm provides the best results even though it accounts for only read operations as explained before. As for the backtracking algorithm, although the time needed for predicting  $N$  shooting cells is significant (expensive tree traversal) the determined ordering is not more efficient, which explains the bad performance of this algorithm.

Figure 6 shows three different plots giving for three iterations and for the  $i^{th}$  selected shooting cell: (i) the computing time, (ii) the number of surface elements in memory, (iii) the number of links stored in memory. As during the first iteration only light sources shoot their energy, the entailed computing time is low. On the other hand, it increases drastically during the next iterations. Moreover the number of surface elements and the number of links increase progressively till reaching the maximum value related to  $M_{limit}$ .

Figure 8 and 9 show two images obtained with our method after complete resolution.

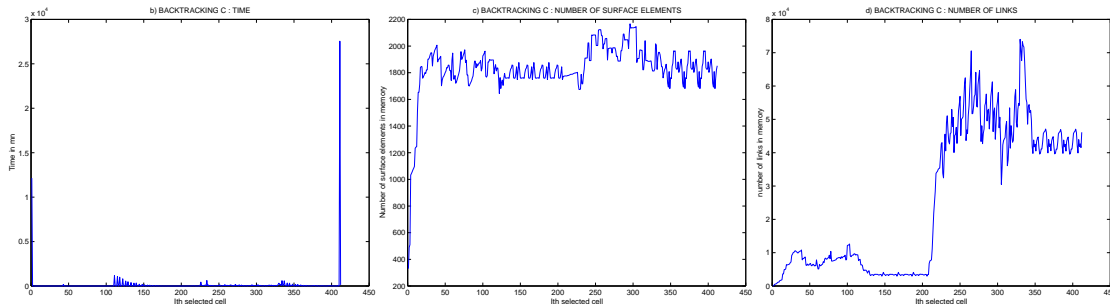


Figure 6: Computing time, number of links, number of surface elements in memory for the Bactracking C strategy.

## 6 Conclusion

We have proposed seven ordering strategies for computing radiosity solutions in large environments. They are based on the partitioning of the scene into cells and on simple visibility calculations between these cells. Only a subset of cells is resident in memory: the shooting cell and those visible from it. These resident cells change during the resolution process, implying then many read and write operations from or onto the disk. The role of these strategies is to reduce the number of these disk transfers. It turns out that the best strategy

is Traveling Salesman. Further experiments are needed to draw definitive conclusions on these strategies.

Note that for the sake of efficiency, our method manages the memory space where are stored the cells involved in the current radiosity computation.

Unlike the methods described in [1, 6] our algorithm uses shooting rather than gathering so as to offer to the user the possibility of interactively rendering the scene (at the intermediate stages of the resolution) while pursuing the radiosity computation.

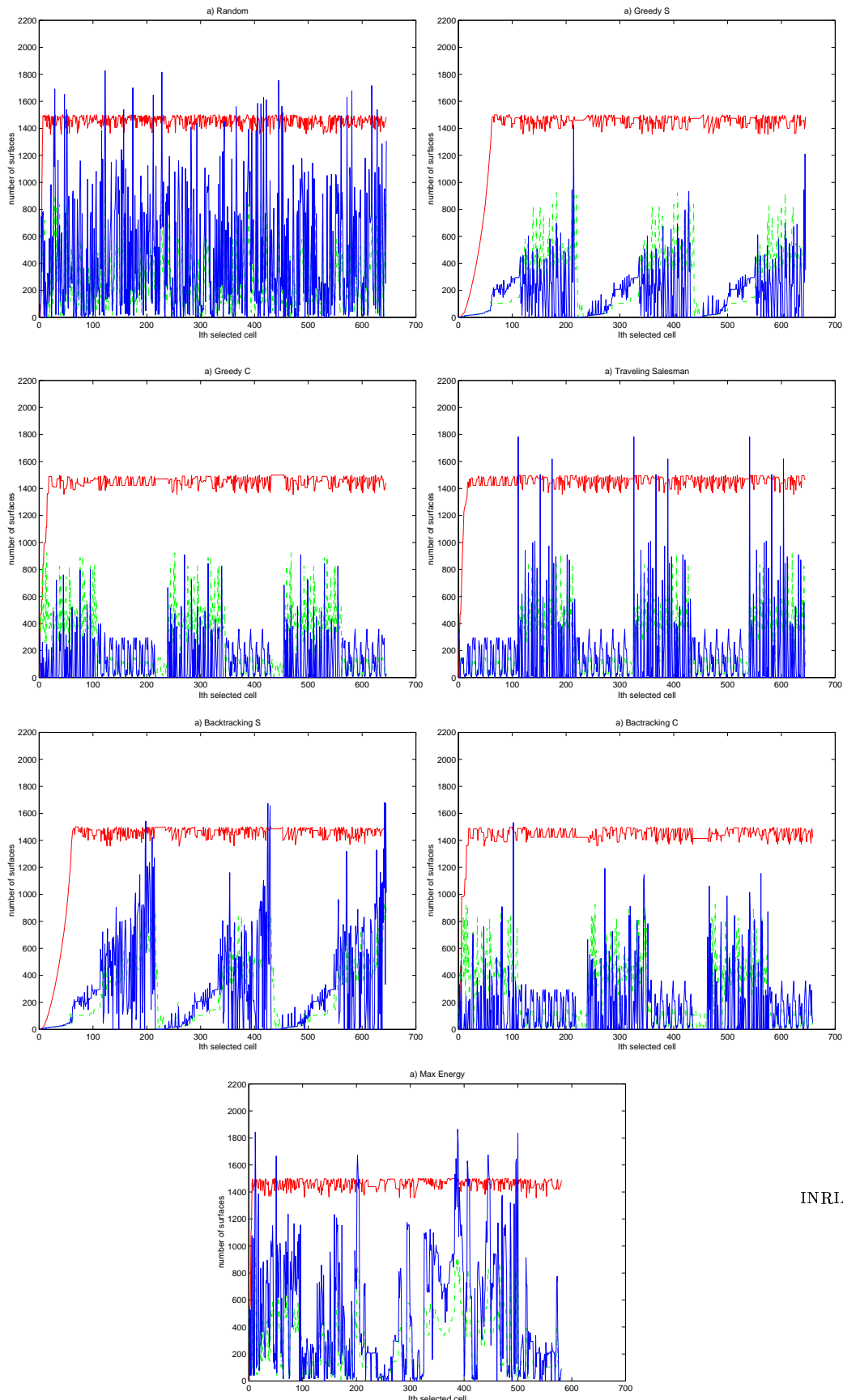
Another advantage of our method is its easy adaptation to the available computing resources. Indeed, it is possible to stop the calculations after a certain number of iterations and to restart them later on without affecting the final radiosity solution.

We are currently investigating other ordering strategies accounting for other criteria: energy distribution, visibility graph...

Finally, these ordering algorithms are the starting point for devising a parallel version. A processor could be in charge of a shooting cell and those visible from it which will ensure data locality. This latter would increase the performances of the parallel algorithm.

## References

- [1] Seth Teller & Celeste Fowler & Thomas Funkhouser & Pat Hanrahan. Partitioning and ordering large radiosity computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 443–450, 1994.
- [2] John M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision And Potentially Visible Set Calculations*. PhD thesis, University of North Carolina at Chapel Hill, 1990.
- [3] Seth Teller & Pat Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 239–246, 1993.
- [4] Seth Jared Teller. *Visibility Computations in Density Occluded Polyhedral Environments*. PhD thesis, University of California at Berkeley, 1992.
- [5] D. Meneveaux & E. Maisel & K. Bouatouch. Efficient partitioning of architectural environments. *Submitted to CAD*, 1997.
- [6] Thomas Funkhouser. Coarse-grained parallelism for hierarchical radiosity using group iterative methods. *ACM SIGGRAPH'96 proceedings*, pages 343–352, August 1996.



INRIA

Figure 7: Number of surfaces

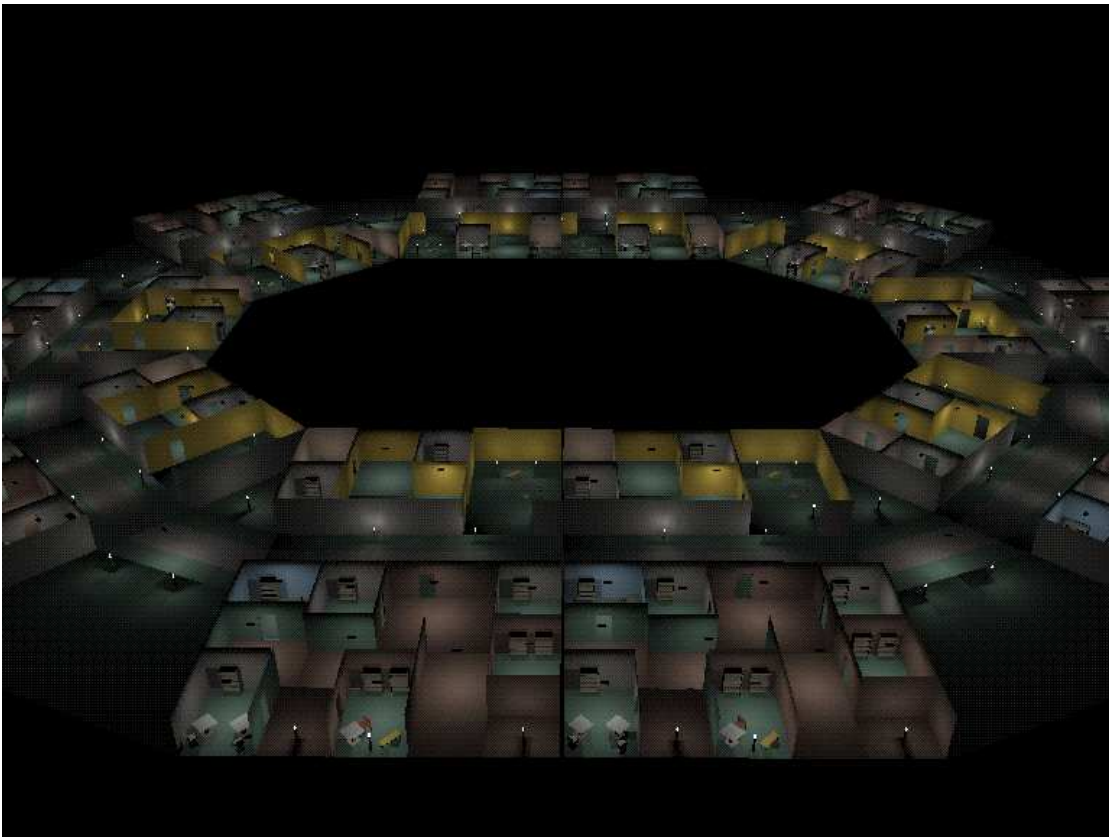


Figure 8: Top-view

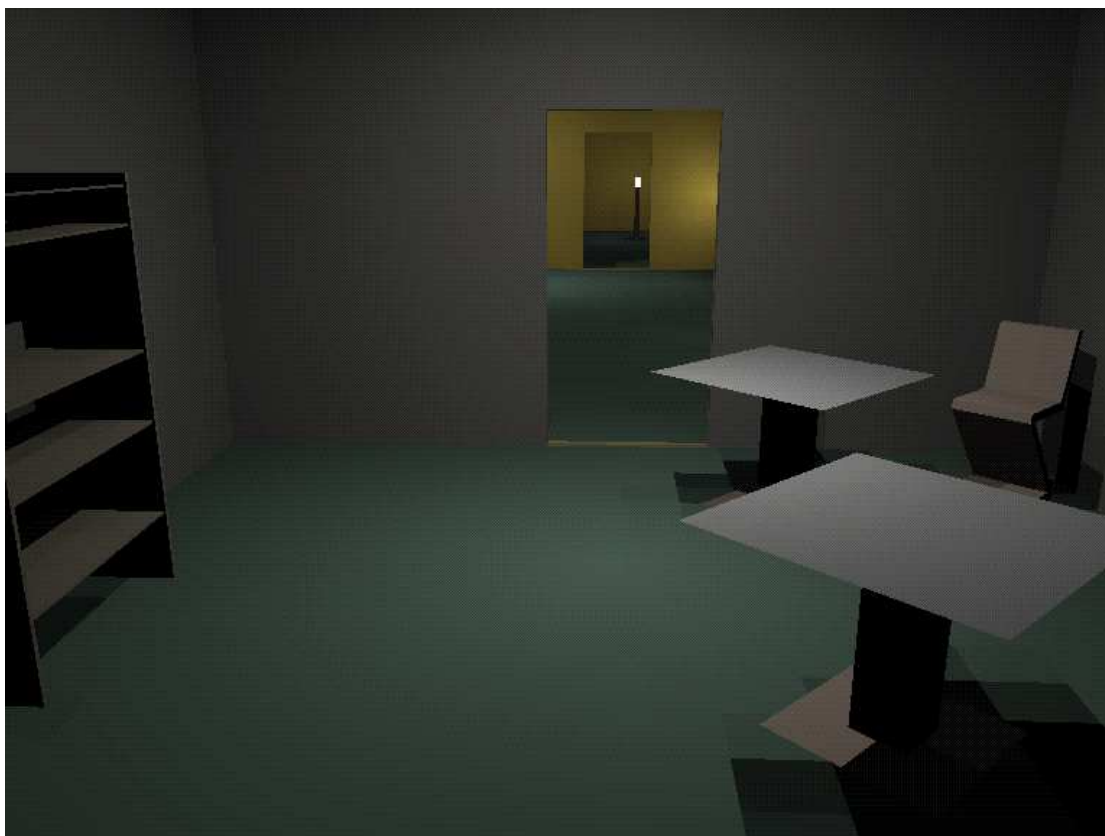


Figure 9: Interior view



---

Unit e de recherche INRIA Lorraine, Technop ole de Nancy-Brabois, Campus scientifique,  
615 rue du Jardin Botanique, BP 101, 54600 VILLERS L ES NANCY  
Unit e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex  
Unit e de recherche INRIA Rh one-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN  
Unit e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex  
Unit e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

---

 diteur  
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)  
<http://www.inria.fr>  
ISSN 0249-6399