INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Parallel Hierarchical Radiosity for Complex Building Interiors

Daniel Meneveaux, Kadi Bouatouch IRISA, Campus de Beaulieu 35042 Rennes

Cedex, France

N ° ????

Mai 1998

THÈME 3

ISSN 0249-6399

INRIA
RENNES

Parallel Hierarchical Radiosity for Complex Building Interiors

Daniel Meneveaux, Kadi Bouatouch IRISA, Campus de Beaulieu 35042 Rennes Cedex, France

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

Rapport de recherche n???? — Mai 1998 — 18 pages

Abstract: In this paper we propose a SPMD parallel hierarchical radiosity algorithm relying on a novel partitioning method which may apply to any kind of architectural scene, not necessarly axial. This algorithm is based on a public domain software called MPI (Message Passing Interface) which allows the use of either a heterogeneous set of concurrent computers or a parallel computer or both. The database is stored on a single disk and accessed by all the processors (through NFS in case of a network of computers). As the objective is to handle complex scenes like building interiors, to cope with the problem of memory size, only a subset of the database resides in memory of each processor. This subset is determined with the help of partitioning into 3D cells, clustering and visibility calculations. A graph expressing visibility between the resulting clusters is determined, partitioned (with a new method based on classification of K-means type) and distributed among all the processors. Each processor is responsible for gathering energy (using Gauss Seidel method) only for its subset of clusters. In order to reduce the disk transfers due to downloading these subsets of clusters, we use an ordering strategy based on the traveling salesman algorithm. Dynamic load balancing relies on a task stealing approach while termination is detected by configuring the processors into a ring and moving a token round this ring. The parallel iterative resolution is of group iterative type. Its mathematical convergence is proven in appendix.

Key-words: lighting simulation, parallel algorithm, complex environments

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Radiosité Hiérarchique Parallèle pour des Environnements Architecturaux Complexes

Résumé : Nous proposons un algorithme de radiosité hiérarchique parallèle, de type SPMD pouvant être appliqué à tout type d'environnement architectural complexe. Cet algorithme utilise l'environnement de programmation parallèle MPI (Message Passing Interface) et peut être exécuté sur un réseau hétérogène de machines, une machine parallèle ou bien les deux ensemble. La base de données est stockée sur un disque commun, accessible par tous les processeurs (à l'aide de NFS dans le cas d'un réseau de machines). Afin de faire face aux problèmes dus au stockage de l'environnement en mémoire, seule une petite partie de la scène est traitée à la fois. Pour cela, l'environnement est découpé en plusieurs régions 3D que nous appelons *cellules*. Pour chaque cellule, nous déterminons des groupes de surfaces (ou *clusters*) et construisons un graphe exprimant les relations de visibilité entre ces *clusters*. Un nœud représente alors un *cluster* et un arc entre deux nœuds indique que les *clusters* correspondants sont mutuellement visibles. Ce graphe est ensuite découpé en plusieurs sous-graphes à l'aide d'une technique de classification de type *k-means* issue des techniques d'analyse de données. Chaque sous-graphe est attribué à un processeur et chaque processeur est chargé de calculer l'éclairement des clusters qui lui ont été attribués en utilisant la méthode de résolution itérative de Gauss-Seidel. Afin de réduire les accès au disque impliqués par les chargements et déchargements des sous-ensembles de clusters en mémoire, nous ordonnons les calculs selon un algorithme du voyageur de commerce. Par ailleurs, nous avons mis en œuvre un équilibrage de charge dynamique de type "vol de tâche" (ou *task-stealing*). La terminaison de l'algorithme est détectée en reconfigurant les processeurs en anneau dans lequel circule un jeton. Cet anneau est également utilisé pour traiter la convergence de notre algorithme de radiosité. La résolution parallèle de l'algorithme est de type groupe itératif. Une preuve matématique de la convergence est donnée en annexe.

Mots-clé : simulation d'éclairage, parallélisme, environnements complexe

1 Introduction

Hierarchical radiosity provides high level realistic images at the expense of high 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 computed, which entails a very high 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 [1, 2, 3, 4] where the authors propose to partition the scene into subsets of polygons (called *cells*) before performing radiosity computations whether sequentially [5] 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 large that they cannot fit in memory. One solution to this problem is, once again, to partition the scene into cells so that only a subset of the scene is maintained in memory for illumination computations. This is made possible with the help of spatial and visibility calculations. For example, an object (which can be a polygon or a cluster of polygons or other objects) within in a cell C shoots/gathers energy to/from all the objects lying in C as well as those visible through the holes (also termed *portals*) in the boundary of C like windows or doors for buildings interiors. Thus, one needs to load in memory the objects (stored on the disk) concerned with the radiosity computations, to withdraw the non-necessary ones from memory and move them back to the disk. This operation requires many disk transfers which are very time consuming. These transfers must be ordered so as to reduce expensive read and write operations (from/onto the disk or memory). Several ordering strategies have been proposed in [5, 7].

The partitioning of a scene into 3D cells makes possible to perform, in parallel, radiosity computations for all the subscenes (made up of cells and objects). Such a parallelism combined with ordering strategies could be a solution for coping with the high memory and computation constraints. A few works deal with parallel hierarchical radiosity in the context of complex scenes containing millions of objects. To our knowledge, only two papers address this problem [6, 8]. The corresponding works are briefly reviewed in the next section as well as their advantages and drawbacks. Note that a recent parallel radiosity algorithm, not hierarchical but using a static uniform meshing of the input polygonal surfaces, has been presented in [9].

In this paper we propose a parallel hierarchical radiosity algorithm relying on a novel partitioning method [10] which may apply to any kind of architectural scene, not necessarly axial. This algoritm is based on a public domain software called MPI (Message Passing Interface) which allows the use of either a heterogeneous set of concurrent computers or a parallel computer or both. After a brief review of related works in section 2, the following section describes our algorithm together with the different steps involved: data distribution, task scheduling, load balancing and termination. As our algorithm is based on a group iterative approach using Gauss Seidel method, we give a proof of its convergence in section 8. The remaining sections are dedicated to implementation aspects, results and conclusion.

2 Related works

2.1 Master-Slave parallelism

The parallel hierarchical radiosity algorithm described in [6] is based on a group iterative approach and runs on a master-slave parallel environment. The master processor partitions the scene into groups of clusters (a cluster is a small set of neighboring patches). More precisely, a BSP technique is performed and results in a set of 3D cells containing clusters for which visibility relationships are determined. These latter are represented by a valuated graph in which a node is a cluster and the value associated with an edge is equal to the estimated form factor between two linked nodes. This graph is split into groups of nodes corresponding to groups of clusters. Once this splitting has been performed, the master distributes these groups among the slave processors which perform radiosity computations for their own groups and send the master the obtained results. Each slave maintains a group of clusters which varies over time, allowing then dynamic load balancing performed by the master. This process is repeated iteratively till global convergence. The slaves and the master communicate via TCP messages only. Note that the granularity of the parallel algorithm is relatively coarse.

2.2 Finer grain parallelism

In a more recent paper, Feng and Yang describe a parallel algorithm with a finer granularity running on a master-slave parallel environment. Like in [6], the scene is partitioned into 3D cells with a BSP technique and CV-sets are computed. A CV-set, associated with a cell C, is a set formed with all the patches within C and those lying in other cells but visible to C. The CV-sets are arranged in a sequential order according to a traveling salesman graph, and each CV-set is in turn replicated to all slave processors for reducing the number of exchanged messages. When a CV-set is replicated in each processor, its associated cell C shoots its energy. To this end, the master processor partitions the cell C into groups of patches (by estimating their workload) and distributes these groups to the slaves. Next, each slave shoots the energy of the patches within its own group. The energies gathered by patches within C but assigned to other processors are sent to these latter. In a second step

job scheduling is based on the amount of unshot energy of the CV-sets. Note that dynamic load balancing is achieved with a task stealing strategy.

2.3 Discussion

Funkhouser's work is original since it is based on a group iterative approach the convergence of which is mathematically proved. On the other hand the method entails a lot of very long messages between the master and the slave processors, which dramatically limits its performances.

Feng and Yang tried to reduce the granularity of their algorithm by distributing the patches within the current replicated CV-set among all the slave processors. However the replication of a CV-set requires more memory capacity per processor than Funkhouser's algorithm. In addition, the number of messages still remains high and the used dynamic strategy relies, in our opinion, on an inefficient criterion. The second step of job scheduling is based on the amount of unshot energy of the CV-sets. We have made experiments that showed the inefficiency of this kind of scheduling. Moreover, the authors do not give a convergence proof of their parallel iterative resolution method.

3 Multi-platform Solution

3.1 Objectives

- As our parallel hierarchical radiosity is based on a group iterative approach, we tried to make it faster by using Gauss-Seidel iterative resolution method rather than Jacobi's one.
- Prove the convergence of this approach.
- In case of a parallel computer, reduce the number as well as the size of the exchanged messages by putting the shared database on a common disk.
- When using a set of computers connected to a network (like ethernet for example), all the database is located on the same disk and is accessed by NFS in a way transparent for the user, which facilitates the implementation of the parallel algorithm.
- Propose an efficient load balancing and an elegant termination algorithm.
- Use a parallel programming environment like MPI or PVM to run the algorithm in different plateforms: parallel computer, heterogeneous network of computers, etc.

3.2 Principle

Our parallel algorithm allows radiosity computation for complex scenes such as building interiors made up of polygonal objects. Unlike those described in [6, 8], our algorithm is of SPMD type. A preprocessing step partitions the scene into 3D cells

and determines the clusters included in each cell. A cluster can be viewed as either a polygon, or a small set of neighboring polygons, or the polygons making up an object (like a chair, a table,a lamp, a teapot, etc.). This partitioning helps to efficiently compute visibility betwen the resulting cells. A cell views another cell through portals such as windows and doors. A graph G_{cell} expresses all the visibility relationships between the cells. We use G_{cell} to determine another visibitlity graph $G_{cluster}$ for clusters. To this end, we suppose that a cluster in a cell C views all the other clusters within the same cell C, and we determine, for each cluster in C, all the clusters within other cells and visible to this cluster. A node of $G_{cluster}$ is then a cluster and an edge links two mutually visible clusters. The edges of the graph $G_{cluster}$ are valuated according to a technique explained in section 3.3.1. This valuation gives rise to a new graph G_{val}. This latter is partitioned into subgraphs the nodes (clusters) of which are distributed among the processors. More precisely, all the nodes within a subgraph are assigned to the same processor. To sum up, preprocessing is performed by one processor and results in: a set of clusters, a graph expressing visibility relationships between clusters and a set of subgraphs.

Once this preprocessing has been performed, the parallel algorithm is run. It operates as follows. We use MPI as a parallel programming environment. We do not use a master-slave approach but assign to each processor a list of clusters corresponding to the nodes of a subgraph of G_{val}, say a subset of neighboring clusters. Each processor performs radiosity computation for its own clusters by using gathering. It gathers energy impinging on its own clusters. To do that, it selects one of its own cluster (called receiving cluster), gathers energy for this cluster, switches to another cluster and repeats the selection/gathering process till all the clusters assigned to the processor be selected as receiving cluster. The selection of a cluster for gathering may require to download other clusters not asssigned to the processor but visible to its clusters. To reduce the disk transfers entailed by downloading clusters, we use a cluster selection algorithm based on a traveling salesman strategy as used in [7, 8]. This algorithm makes use of another oriented and valuated graph OVG. Each processor builds its own OVG in which a vertex is one of its assigned clusters and the value of an edge (C1,C2) is the Input/Output cost due to download caused by the selection of C2 as the next receiving cluster after C1. From this OVG graph the processor determines a cycle (called CYCLE from now on) passing through all the vertices of OVG. This CYCLE is traversed sequentially to select the receiving clusters.

Let us recall that all the data structures manipulated by the processors are placed on a common disk. More precisely, the database is structured as follows. A directory containing all the database (say all the clusters) can be accessed by all the processors while each processor is assigned one local directory not accessible from the other processors. Each cluster is stored in a specific file, consequently there are as many files as clusters. The common directory contains all these files representing then the whole database. When a processor gathers energy for one of its own clusters, it modifies (by adding new surface elements, updating radiosities) the data structures associated with some of its clusters and with other clusters assigned to other processors. After each gathering the processor writes the modified data structures, regarding its own

clusters only, in files located in its local directory. The other modified data structures are handled by other processors that are responsible for.

When all the processors have performed gathering for all their clusters, the current iteration of the global iterative resolution method is completed. Then the processors move the contents of their local directories to the common directory and a new iteration may start. Note that, during one iteration, as soon as a processor has gathered energy for all its clusters it sends a request for job (for getting clusters previously assigned to other processors) to the other processors according to a task stealing strategy. If the request for job fails for all the processors, the current iteration is completed and the processors are configured into a ring for testing convergence and termination.

3.3 Algorithm

The pseudo-code of parallel radiosity running on each processor is given in figure 2. In all the algorithms, the type **Cluster** is an integer corresponding to the suffix of the name of the file where is stored a cluster.

3.3.1 Graph partitioning

In this section we show how the graph G_{val} is built and partitioned into subgraphs the nodes of which are destinated to be distributed among all the processors. Note that the nodes of a subgraph are assigned to the same processor.

First of all, the graph $G_{cluster}$ is copied into a new graph datastructure G_{val}. Each edge of G_{val} is attributed the value 1. Then we apply Warshall's algorithm to this graph in order to compute its transitive closure and the distance between any pair of nodes as well. The distance between two nodes of G_{val} is equal to the number of edges of the shortest path joining these nodes. Now, we apply to G_{val} a classification algorithm (figure 1) to partition it into subgraphs, each one corresponding to a group of nodes representing neighboring clusters. With each group is associated a center of gravity (COG) which is a node of G_{val}. These groups as well as their COGs are determined iteratively as explained in figure 1. The used classification algorithm is of K-means [11] type where each class (called a group of nodes in our case) is represented by its most central element, say its center of gravity. We think that our graph partitioning algorithm is more intuitive and more efficient than the one proposed in [6].

3.3.2 Iterative resolution

A processor performs gathering for its clusters sequentially. Recall that whenever a processor has gathered energy for a cluster, it saves it into a separate file. There are as many files as clusters. These files are located in a directory local to this processor. Once all the processors have completed the current gathering iteration, they move the contents of their local directories to the common directory. Afterwards a new iteration starts if the convergence criterion is not met. Else the parallel algorithm ends up.

```
GraphPartitioning(G_val, NumberOfProcessors) {
    GroupOfNodes G^k = ∅, G^{k-1} = ∅ ; /* Groups of nodes, */
    /* k denoting the current iteration while (k-1) the preceding one */
    Node COG[NumberOfProcessors]; /* array of centers of gravity */
    ChooseCenterOfGravities(G_val, COG) ;
    /* The first centers of gravity are randomly chosen */
    /* among the nodes of G_val */
    Do {
        G^{k-1} = G^k ;
        for each node N_i of G_val {
            /* Determine the center of gravity I which is closest to N_i */
            /* using the distance between two nodes given by G_val */
            I = ClosestCenterOfGravity(N_i, COG);
            /* include N_i in the group associated with COG[I] */
            G^k[I] = G^k[I] + {N_i} ;
        }
        /* Compute the actual center of gravity of each group and */
        /* store the new centers in COG */
        RecomputeCentersOfGravity(G^k, COG) ;
    } while G^k ≠ G^{k-1} ;
}
```

Figure 1: Graph partitioning

Gathering is performed with Gauss Seidel method rather than with Jacobi's one because it is faster. In fact these two methods are used simultaneously as explained in the following. Recall that a processor $Proc_i$ is responsible for a group of clusters C_i which are loaded in its memory. $Proc_i$ must also download other clusters C_j, visible to clusters C_i, but owned by other processors $Proc_j$. The radiosities of patches within the clusters C_j are not modified during the iteration performed by processor $Proc_i$ since they are assigned to other processors and serve just for shooting their energy. This explains why we use Gauss Seidel method when the shooting patches belong to the clusters C_i and Jacobi method when energy is shot from the patches within the clusters C_j. Section 8 gives a mathematical proof of the convergence of this iterative group resolution algorithm.

3.3.3 Dynamic load balancing with task stealing

Recall that dynamic load balancing is performed according to a task stealing strategy and is implemented through the function RequestForCluster() given in figure 4. For a given processor $Proc$, this function invokes another function SendRequestForCluster($Proc_i$) which sends the processor $Proc_i$ the list of the numbers of the clusters in $Proc$'s memory. If processor $Proc_i$ still possesses clusters which have not already gathered energy at the current iteration, it determines (thanks to this list) the one C minimizing the cost of disk transfers due to downloading the clusters visible to C by processor $Proc$. Then processor $Proc_i$ sends processor $Proc$ the number associated with C. The determination and the sending of C is performed by the function ReplyToRequest($Proc$). Note that as soon as $Proc_i$ gives up a cluster to another processor, it removes the associated node in its CYCLE and links together the preceding an succeeding nodes. Recall that CYCLE serves for implementing the traveling salesman ordering strategy as explained in section 3.2.

3.3.4 Termination and Convergence

The function **termination()** detects the convergence of the parallel algorithm. It is implemented as follows. First, the set of processors is configured into a ring. Each processor i computes a radiosity value $T_i = max_{j \in [1,N_i]} |B_j^k - B_j^{k-1}|$, where B_j^k is the radiosity of patch j at iteration k and N_i the number of patches belonging to the clusters which the processor i is responsible for. A processor is idle if it has processed all its own clusters and its requests for job fail. When the processor, whose number is 0, is idle, it sends its successor (processor 1) a token whose value is T_0. This token moves round the ring with a value which may be modified by the visited processors. When a processor i gets a token of value T (from the preceding processor $(i-1)$) it sends its successor the maximum value of T and T_i only if it is idle. When the token has completed a tour round the ring, processor 0 compares its updated value T_{final} with a threshold T_{thres} fixed by the user. If $T_{final} > T_{thres}$ then processor 0 sends its successor a new token with T_{final} as a value to notify the other processors to perform a new iteration. We can see that the first role of the token is to synchronize all the processors. Conversely, if $T_{final} \leq T_{thres}$ then processor 0 sends its successor a new token with -1 as a value to notify the other processors the termination of the parallel hierarchical radiosity algorithm. This is the second role played by the token.

```
ParallelRadiosity(S) {
    ListOfClusters S ; /* list of integers associated with */
                       /* the clusters assigned to the processor*/
    Cluster C ;
    Boolean convergence = FALSE;

    while ( not convergence ) {

        /* gathers energy for clusters within C */
        For each cluster C in S {
            GathersParallel(C);
            /* write back the updated cluster C on the local directory */
            WriteOnLocalDirectory(C);
        }

        /* When all the clusters in C have gathered energy */
        /* start dynamic load balancing with task stealing strategy */
        Do {
            C = RequestForCluster();
            if C ≠ NULL
                S = S + C ;
                GathersParallel(C);
                WriteOnLocalDirectory(C);
        } while (C ≠ NULL);

        /*End of the current iteration */
        /*Move all the clusters from the local to the common directory */
        MoveToCommonDirectory(S);

        /* Finally we test convergence of the parallel algorithm */
        convergence = Termination();
    }
}
```

Figure 2: Radiosity on one processor.

```
GathersParallel(C) {
    C : set of clusters assigned to the processor;
    CYCLE : cycle passing through all the vertices of OVG built from §C§. ;
    Env : environement containing a receiving cluster and ;
    CYCLE = TravelingSalesman(C);
    For each cluster C_i in CYCLE {
        Env = Download C_i and the clusters it views;
        For each patch S_i in C_i {
            For each patch S_j in Env {
                /* S_i gathers energy from S_j */
                GatherEnergyFromSurface(S_i, S_J);
            }
        }
    }
}
```

Figure 3: Gathering for all the clusters within a subset of clusters

```
Cluster RequestForCluster() {
    int MyRank ; /* the number associated with the processor */
    Cluster C ;
    MyRank = MPI_Rank() ;

    For each processor Proc_i ≠ MyRank {
        SendRequestForCluster(Proc_i);
        C = WaitForReply(Proc_i) ;
        If C ≠ NULL return C ;
    }
}
```

Figure 4: Request for cluster during load balncing

```
Cluster WaitForReply(Proc) {
    int Proc, Proc_j ; /* processor numbers */
    Cluster C; /* Identifier of the received cluster */

    While not ReceiveReply(Proc, &C);
        /* If processor Proc does not reply to the request for job */
        /* then try to reply to requests for job steming from */
        /* other processors Proc_j in order to avoid deadlock*/
        For each processor Proc_j {
            if ReceiveRequestForJob(Proc_j)
                ReplyToRequest(Proc_j) ;
        }
    }
    return C;
}
```

Figure 5: Wait for reply

4 Implementation

4.1 Files structure

Each cluster is stored in a file located in the common directory. This file contains a list of input surfaces, reflectances, spectral power and intensity distributions of the light sources. The visibility graph $G_{cluster}$ is also saved in a file containing the description of each cluster. Indeed, in this file a cluster is described by the following data: (i) the name of the file containing the cluster, (ii) the number of the cluster, (iii) the list of clusters visible to it.

4.2 Modeling and Partitioning into 3D cells

Only building interiors are considered as scenes. These latter are modeled with planar convex polygons with the help of a modeler we have developed specially for complex environments. This modeler is capable of creating different kinds of geometric entities such as polygonal objects, walls, ceilings, doors, windows, etc. It also allows the duplication, removal, geometric transformation of all these entities.

To partition the scene (not necessary axial) into 3D cells, we have used the method described in [10].

4.3 Radiosity computation

Radiosity computation starts once partitioning and clustering have been completed. During this computation the contents of the resulting clusters is modified to include

the created surface elements and associated links. Whenever a cluster has gathered energy, it is saved in the local directory assigned to its processor.

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

4.4 Memory management

Disk transfers involve memory allocations and releases which are managed by the routines *malloc()* and *free()* of the C library *libc.so*. However, after several experiments we have ascertained many swaps whereas the data resident in memory did not require any swap. This is due to the fact that the fragmentation of the main memory is inefficiently managed by these routines. Consequently, we have decided to manage memory by ourselves.

5 Results

This section provides some results only for one scene which is a building composed of three floors (figure 6). It is made up of $57,786$ input surfaces among which about $2,000$ are light sources. This scene has been partitioned into 615 cells containing 12942 clusters. The number of resulting surface elements is about $360,000$ while the number of links (between surface elements) is equal to 3.7 millions. The storage of all these datastructures would need about $3Go$ which is not offered by most of the computers even with a virtual memory.

Our parallel algorithm has been run on several different computers connected to our ethernet local area network. The characteristics of the used computers are given in table 1.

Given name	Kind of computer	Processor	Memory	Clock
SGI1	SGI	R10000	380 Mb	195 MHz
SGI2	SGI	R5000	128 Mb	180 MHz
SGI3	SGI parallel computer	4 x R10000	380 Mb	195 MHz
SUN1	SUN	UltraSpark	256 Mb	200 MHz
SUN2	SUN	SuperSpark	128 Mb	100 MHz

Table 1: Characteristics of the used computers.

Tests have been performed for different combinations of these computers. The obtained results are given in table 2.

The speeds up given in this table have been computed with respect to the run time on one SUN1 which is the most powerful computer among those used for our tests. We can remark that the speeds up obtained with two and three processors exceed the ideal speeds up. This is due to the use of the network by other users, which slows down the disk acesses through NFS. Despite this, the obtained speeds up are encouraging. For SGI3 which is a parallel computer with a shared memory

Combination of computers	Run time	Speed up
1 SUN1	16920 mn	1
2 SUN1	6552 mn	2.5
3 SUN1	5394 mn	3.2
4 SUN1	4338 mn	3.9
1 SGI3	4594 mn	3.9
4 SUN1 + 1 SGI1 + 3 SGI2	2880 mn	5.9
4 SUN1 + 1 SUN2 + 1 SGI1 + 4 SGI2	2160 mn	7.8

Table 2: Results.

and 4 processors, the run time is equivalent to that obtained with 4 SUN1. This can be explained by the fact that SGI3 is not supplied with a specific hardware allowing parallel disk Inputs/Outputs.

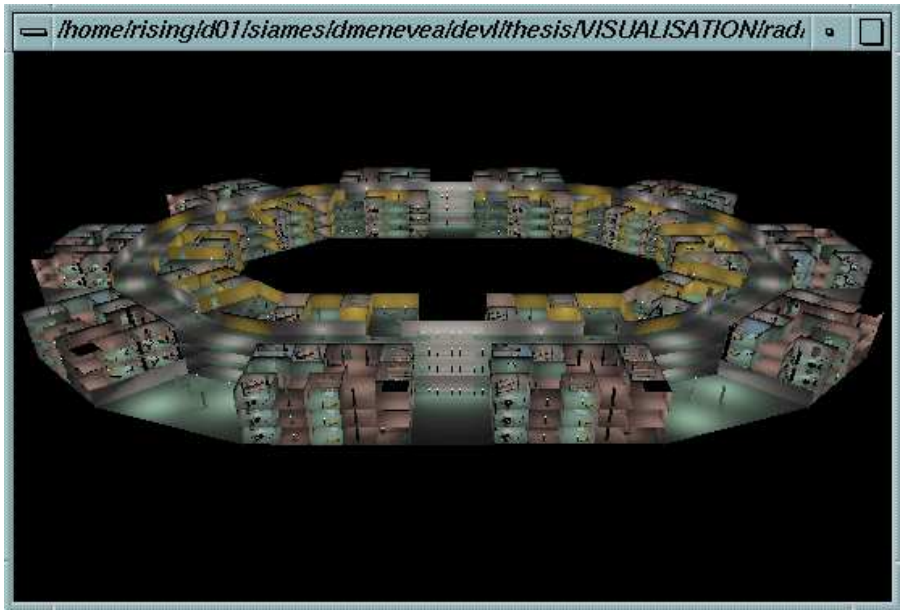

Figure 6: Image of the total scene once the complete global illumination has been performed.

6 Discussion

As seen before, our parallel hierarchical radiosity algorithm can be easily implemented on a heterogeneous network of computers or on a parallel machine with distributed or shared memory. Since the used data structure resides on the same disk, the performances of the algorithm strongly depend on the time needed for the numerous disk accesses. To improve the performances of our algorithm a reduction of this time is indispensable. To this end, for some parallel machines, a hardware

system is added for allowing the processor nodes to access the disk simultaneously. Unfortunately, this kind of machine is not available in our institute.

In case of a network of computers two improvements can be carried out. The first one would be to place each local directory on the disk physically connected to the associated processor. In this way, the accesses to the local directories are performed in parallel. Second, the common directory would be split into several pieces, each one being placed on a different disk. Each processor can access any piece through NFS. Consequently, different clusters can be read simultaneously and, at the end of an iteration, the files containing the clusters can be updated in parallel.

7 Conclusion

We have proposed a parallel hierarchical radiosity algorithm based on a public domain software called MPI. It does not rely on master-slave environment and can be implemented on any kind of heteregeneous network of computers or any parallel machine provided that they are supplied with MPI. This latter does not slow down our algorithm since the extra computation time due to message communication is very low. Indeed, the number of messages is insignificant and their size very small. To cope with the problem of memory size, only a subset of the database resides in memory. This subset is determinded with the help of partitioning into 3D cells, clustering and visibility calculations. In order to reduce the disk transfers due to downloading these subsets of clusters, we use an ordering strategy based on the traveling salesman algorithm. In another paper [10] we show that this ordering strategy seems to outperform others like: greedy, random, maximum energy, back tracking, etc. The use of local and common directories ensures data coherence. The advantage of our parallel algorithm is better exploited in case of parallel computers with processors accessing the same disk because no messsages are needed to access the commun disk, and only a small number of short messages are required for synchronizing the processors and detecting the termination of the parallel algorithm. For numerous experiments, the termination algorithm and the used dynamic load balancing technique seem efficient. Finally, as our group iterative resolution is based on Gauss Seidel method, we give a proof of its convergence.

References

[1] J. M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculation.* PhD thesis, University of north Carolina at Chapel hill, 1990.

[2] Seth Teller & Pat Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 239–246, 1993.

[3] Seth Jared Teller. *Visibility Computations in Density Occluded Polyhedral Environments.* PhD thesis, University of California at Berkeley, 1992.

[4] D. Meneveaux & E. Maisel & K. Bouatouch. A new partitioning method for architectural environments. Technical Report 3148, INRIA, April 1997.

[5] 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.

[6] Thomas Funkhouser. Coarse-grained parallelism for hierarchical radiosity using group iterative methods. *ACM SIGGRAPH'96 proceedings*, pages 343–352, August 1996.

[7] D. Meneveaux, K. Bouatouch, and E. Maisel. Memory management schemes for radiosity computation in complex environments. In *CGI'98 Hannover*, June 1998.

[8] C C Feng and S N Yang. A parallel hierarchical radiosity for complex scenes. *Parallel Rendering Symposium*, October 1997.

[9] B. Arnaldi, T. Priol, L. Renambot, and X. Pueyo. Visibility masks for solving complex radiosity computations on multiprocessors. Technical Report 1055, IRISA, 1996.

[10] D Meneveaux, K Bouatouch, E Maisel, and R Delmont. A new partitioning method for architectual environments. *To appear in Visualization and Computer Animation*, 1998.

[11] D. J. Hall and G. H. Ball. Isodata a novel method of data analysis and pattern classification. Technical Report 5 RI project 5533, Stanford Research Institute, CA, USA, 1965.

[12] D. M. Young. *Iterative solution of large linear systems.* Computer science and applied mathematics. Academic Press, New York, 1971.

8 Appendix : Convergence proof

Let us give now the proof the convergence of our Gauss Seidel-based iterative group resolution method. It is valid for constant hierarchical radiosity only. Its extension to higher order hierarchical radiosity is being investigated.

The radiosity system can be written as:

$$Ax = b,$$

where x is the unknown radiosity vector, b the vector of self-emttance and A the system matrix.

This system can be rewritten as:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{1}$$

We can write:

$$x = A^{-1}b.$$

if A is non singular.

Gauss Seidel method expresses this system under the following form where n is the total number of patches and $(k+1)$ is the the number of the current iteration:

$$\begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ 0 & 0 & 0 & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix} \tag{2}$$

Each processor L is in charge of the p patches of indices ranging from L_1 to L_p. Thus, for one processor the above system can be written as:

$$\begin{pmatrix} \tilde{0} & \tilde{0} & \tilde{0} \\ \vdots & \vdots & \vdots \\ \tilde{0} & M^L & \tilde{0} \\ \vdots & \vdots & \vdots \\ \tilde{0} & \tilde{0} & \tilde{0} \end{pmatrix} \begin{pmatrix} x_1^{k+1} \\ \vdots \\ \vdots \\ \vdots \\ x_n^{k+1} \end{pmatrix} + \begin{pmatrix} \tilde{0} & \tilde{0} & \tilde{0} \\ \vdots & \vdots & \vdots \\ \tilde{A}_1^L & \tilde{0} & \tilde{A}_2^L \\ \vdots & \vdots & \vdots \\ \tilde{0} & \tilde{0} & \tilde{0} \end{pmatrix} \begin{pmatrix} x_1^k \\ \vdots \\ \vdots \\ \vdots \\ x_n^k \end{pmatrix} = \begin{pmatrix} \vdots \\ b_{L_1} \\ \vdots \\ b_{L_p} \\ \vdots \end{pmatrix}.$$

The first term of the lefmost hand side corresponds to the contributions of the patches assigned to processor L, while the second expresses the contributions computed by the other processors at the preceding iteration k.

M^K is a lower triangular matrix whose elements are $(M^L)_{i,j}$, where $i, j \in [L_1, L_p]$. $\tilde{A}_1^L$ and $\tilde{A}_2^L$ are composed of elements a_{ij} of matrix A. $\tilde{A}_1^L$ is a matrix with elements $(\tilde{A}_1^L)_{ij} = (a_{ij})$, where $i \in [L_1, L_p]$ and $j \in [1, L_1]$. $\tilde{A}_2^L$ is a matrix with elements $(\tilde{A}_2^L)_{ij} = (a_{ij})$, where $i \in [L_1, L_p]$ and $j \in [L_p{+}1, n]$. $\tilde{0}$ are square matrices containing null elements $(\tilde{0})_{ij} = 0$, $i \in [i_1, i_2]$, $j \in [j_1, j_2]$, where $i_1, i_2, j_1, j_2 \in [1, n]$.

If we consider N processors involved in the prarallel radiosity algorithm we get:

$$\begin{pmatrix} M^1 & \tilde{0} & \tilde{0} \\ \tilde{0} & \ddots & \tilde{0} \\ \tilde{0} & \tilde{0} & M^N \end{pmatrix} \begin{pmatrix} x_1^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix}$$

$$+ \begin{pmatrix} \tilde{0} & \tilde{0} & \tilde{0} \\ A_1 & \tilde{0} & \tilde{0} \\ A_2 & A_3 & \tilde{0} \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} + \begin{pmatrix} \tilde{0} & A_4 & A_5 \\ \tilde{0} & \tilde{0} & A_6 \\ \tilde{0} & \tilde{0} & \tilde{0} \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Each square matrice M^L is associated with processor L. The matrices $A_1, A_2, A_3, A_4, A_5, A_6$ are composed of matrices A_i^L, where $L \in [1, N], i = 1, 2$.

We can rewrite this system in a condensed form if D is non singular as:

$$\begin{array}{rl} & Dx^{(k+1)} - Lx^{(k)} - Ux^{(k)} = b \\ \Leftrightarrow & Dx^{(k+1)} = b + (L+U)x^{(k)} \\ \Leftrightarrow & x^{(k+1)} = D^{-1}b + D^{-1}(L+U)x^{(k)} \end{array}$$

where $U \geq 0$ and $L \geq 0$ since $\tilde{A}_i^L \leq 0$.

Note that D is a lower triangular matrix and contains values "1" (since $a_{ii} = 1, \forall i$) on its diagonal. Thus $det(D) = 1 \neq 0$, which means that the matrix D is not singular.

Now that we have expressed $x^{(k+1)}$ as a function of $x^{(k)}$, let us show that our group iterative method converges.

Let $\varepsilon^{(k)}$ be the error expressed as:

$$\varepsilon^{(k)} = x^{(k)} - x$$

We get then:

$$\left\{ \begin{array}{ll} (a) & Dx = b + (U+L)x \\ (b) & Dx^{(k+1)} = b + (U+L)x^{(k)} \end{array} \right.$$

$$\begin{array}{rl} (b) - (a) & \Leftrightarrow \quad D(x^{(k+1)} - x) = (U+L)(x^{(k)} - x) \\ & \Leftrightarrow \quad D\varepsilon^{(k+1)} = (U+L)\varepsilon^{(k)} \\ & \Leftrightarrow \quad \varepsilon^{(k+1)} = D^{-1}(U+L)\varepsilon^{(k)} \end{array}$$

We also have the following equivalence:

$$\forall \varepsilon^{(0)}, \lim_{k \to \infty} (D^{-1}(U+L))^k = 0 \Leftrightarrow \rho(D^{-1}(U+L)) < 1$$

Now we have to prove that $\rho(D^{-1}(U+L)) < 1$, $\rho()$ being the spectral radius.

Let us show that $\rho(D^{-1}(U+L)) < 1$ by using corollary 5.6 on page 125 of Young [12] (this corallory has also been used in [6]).

Corollary 1
Let A be a monotone matrix and let $A = Q_1 - R_1$ and $A = Q_2 - R_2$ be two regular splittings of A. If $R_2 < R_1$ then $\rho(Q_2^{-1}R_2) \leq \rho(Q_1^{-1}R_1)$.

As D is a lower diagonal matrix which contains values "1" on its diagonal and negative or null values elsewhere ($D_{ij} = -\rho_i F_{ij}, \rho_i$ being the reflectivities and F_{ij} the form factors between patches i and j), we have then:

$$A = I - N = D - (U+L) = (I - L') - (U+L)$$

Hence:

$$U + L = N - L' \leq N$$

with $N \geq 0$ and $L' \geq 0$.

Let us write $Q_1 = I$, $R_1 = N$ and $Q_2 = D$, $R_2 = (U+L)$. By using corollary 1 we get $\rho(D^{-1}(U+L)) \leq \rho(I^{-1}N) = \rho(N) < 1$ $\square$.

Consequently our group iterative method based on Gauss Seidel converges.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-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

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