Volume xxx, (1999) number yyy pp. 000–000

# Synchronisation and Load Balancing for Parallel Hierarchical Radiosity of Complex Scenes on a Heterogeneous Computer Network

Daniel Meneveaux, Kadi Bouatouch

IRISA, Campus de Beaulieu
35042 Rennes Cedex, France

**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. This algorithm is based on MPI (Message Passing Interface), a communication library which allows the use of either a heterogeneous set of concurrent computers or a parallel computer or both. The database is stored on a common directory and accessed by all the processors (through NFS in case of a network of computers). As the objective is to handle complex scenes such as 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 a 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 amongst all the processors. Each processor is responsible for gathering energy (using the 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 around this ring. The parallel iterative resolution is of group iterative type. Its mathematical convergence is proven in the appendix.

## 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 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 unnecessary such as those between a polygon and some objects occluded by a large polygon such as a wall. This problem has already been addressed in [1, 2, 3, 4, 5] where the authors propose to partition the scene into subsets of polygons (called *cells*) before performing radiosity computations either sequentially [6] or in parallel [7, 5].

Another problem is 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 $C$, for example windows or doors for buildings interiors. Thus, one needs to load into memory the objects (stored on the disk) concerned with the radiosity computations, to remove the unnecessary ones from memory and 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 (to/from the disk or memory). Several ordering strategies have been proposed in [6, 8].

The partitioning of a scene into 3D cells makes it 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 papers (refer to [9] for further details) deal with parallel hierarchical radiosity in the context of complex scenes containing millions of objects. To our knowledge, only two papers address this problem [7, 10]. They are briefly reviewed in the next section and their advantages and drawbacks discussed. Note that a recent parallel radiosity algorithm, not hierarchical but using a static uniform meshing of the input polygonal surfaces, has been presented in [11].

In this paper we propose a parallel hierarchical radiosity algorithm relying on a novel partitioning method [12] which may be applied to any architectural scene, not only axial ones (in an axial scene, the most occlusive polygons are perpendicular to one axis of the coordinate system). This algorithm is based on a communication library 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 work 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 the Gauss Seidel method, we give a proof of its convergence in the appendix. The remaining sections are dedicated to implementation aspects, results and conclusions.

## 2. Related works

### 2.1. Master-Slave parallelism

The parallel hierarchical radiosity algorithm described in [7] 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 which results in a set of 3D cells containing clusters for which visibility relationships are determined. These are represented by a weighted graph in which a node is a cluster and the weight 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 amongst the slave processors which then perform radiosity computations for their own groups and send the master the results. Each slave maintains a group of clusters which varies over time, allowing dynamic load balancing to be performed by the master. This process is repeated iteratively until global convergence. The slaves and the master communicate via TCP messages only. 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. As in [7], 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 in order to reduce the number of exchanged messages. When a CV-set is replicated at 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 processors. In a second step, job scheduling is based on the amount of unshot energy of the CV-sets. Dynamic load balancing is achieved with a task stealing strategy.

### 2.3. Discussion

Funkhouser's work is significant since it is based on a group iterative approach, the convergence of which is mathematically proved. However, the method entails a lot of very long messages between the master and the slave processors, which, in our opinion, limits its performance.

Feng and Yang tried to reduce the granularity of their algorithm by distributing the patches within the current replicated CV-set amongst 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 dynamic load balancing strategy used relies on a very approximative formula estimating the workload of each patch. The second step of job scheduling is based on the amount of unshot energy of the CV-sets. The authors do not give a convergence proof of their parallel iterative resolution method.

## 3. Parallel algorithm

### 3.1. Motivations and objectives

We would like all the radiosity computations be performed in parallel by all the processors, each of them being in charge of a different portion of the database. This amounts to a group iterative approach [7]. To make this approach faster, we propose to use a Gauss-Seidel iterative resolution method rather than the Jacobi one. To run the algorithm on different computer plateforms such as a parallel computer or an heterogeneous network of computers, it is preferable to use a parallel programming environment such as MPI or PVM. We have opted for MPI but one can use PVM as well.

As noted above, implementing a parallel radiosity algorithm on a set of computers entails a huge amount of messages handled by the program, especially when the algorithm relies on a master-slave approach. We propose reducing these by placing the shared database (representing the entire scene) on a common directory, either located on one disk or distributed amongst several disks in the network. In the case of a heterogeneous network, this database is accessed by NFS in a way transparent to the user, which facilitates the implementation of our parallel algorithm. Note that, when the common directory is placed on a single disk, the messages induced by the NFS system are still numerous, but are not managed by the parallel algorithm.

Another objective is to devise a load balancing strategy which keeps the number of messages as well as their size as small as possible. In addition, as our algorithm is not based on a master-slave approach, the associated termination algorithm must be distributed amongst all the processors.

Finally, we mathematically prove the convergence of the group iterative method used by our parallel algorithm.

### 3.2. Principle

Our parallel algorithm allows the radiosity computation of complex scenes such as building interiors made up of polygonal objects. Unlike those described in [7, 10], 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 (such as a chair, table, lamp, teapot, etc.). This partitioning helps to compute visibility efficiently 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 visibility 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 evaluated according to the technique explained in section 3.3.1. This evaluation gives rise to a new graph $G_{val}$. This new graph is partitioned into subgraphs, the nodes (clusters) of which are distributed amongst 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 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 the radiosity computation for its own clusters using gathering. It gathers energy impinging on its own clusters. Note that none of the accelerating techniques based on clustering is used by our hierarchical radiosity ([13, 14, 15]). Consequently, when we say that a cluster gathers energy, it means that all the input surfaces within it gathers energy in turn. Each processor selects one of its own clusters (called the receiving cluster), gathers energy for this cluster, switches to another cluster and repeats the selection/gathering process until all the clusters assigned to the processor have been selected as a receiving cluster. The selection of a cluster for gathering may require other clusters (not assigned to the processor but visible to its clusters) to be downloaded. To reduce the disk transfers entailed by downloading clusters, we use a cluster selection algorithm based on a traveling salesman strategy as used in [8, 10]. This algorithm makes use of another oriented and weighted graph OVG. Each processor builds its own OVG in which a vertex is one of its assigned clusters and the weight of an edge (C1,C2) is the Input/Output cost of downloading caused by the selection of C2 as the next receiving cluster after C1. This cost is evaluated as the number of clusters to be downloaded and uploaded [8]. 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 directory. 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 owned by this processor) 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 and discards the other modified data structures which are handled by other processors.

When all the processors have performed gathering for all their clusters, the current iteration of the global iterative resolution method is completed. The processors now move the contents of their local directories to the common directory and a new iteration may start. Moving these directories require no messages to be communicated. During one iteration, as soon as a processor has gathered energy for all its clusters it sends a request for a job (for getting clusters previously assigned to other processors) to the other processors according to a task stealing strategy. If the request for a job fails for all the processors, the current iteration is completed and the processors are configured into a ring for testing convergence and termination.

Note that our method does not make use of any clustering method for accelerating radiosity computation, employs wavelet radiosity rather than classical HR, and is suited for building interiors with many rooms and corridors but not for a simple room with objetcs of highly detailed geometry.

### 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 a cluster is stored.

#### 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 destined to be distributed amongst all the processors. 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 weight 1. Then we apply Warshall's algorithm (well known in graph theory) 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 classification algorithm used is of K-means [16] 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 this graph partitioning algorithm is more intuitive and more efficient than the one proposed in [7].

#### 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.

Gathering is performed with the Gauss-Seidel method rather than the Jacobi 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. Let $C$ be the current receiving cluster belonging to this group. $C$ is then downloaded into $Proc_i$'s memory together with another set of clusters $C_i$, visible to $C$ and also owned by processor $Proc_i$. In addition, the set of clusters $C_j$ visible to $C$ but owned by other processors $Proc_j$ are also downloaded. The patches within $C_i$ and $C_j$ shoot their energy to the receiving cluster $C$. The radiosities of patches within the clusters in $C_j$ used by the processor $Proc_i$ (for gathering energy collected by the receiving cluster $C$) are those computed at the preceding iteration because these clusters are located on local directories not reachable from the processor $Proc_i$. To sum up, the resolution method makes use of:

- the radiosities of patches within $C_i$ computed at the preceding and current iterations (imposed by the Gauss-Seidel resolution method), and
- all the radiosities of the patches within $C_j$ computed at the preceding iteration (Jacobi).

The appendix gives a mathematical proof of the convergence of this iterative group resolution algorithm used.

#### 3.3.3. Dynamic load balancing with task stealing

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 and the visibility graph $G_{cluster}$) 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 and succeeding nodes. Recall that CYCLE serves for implementing the traveling salesman ordering strategy as explained in section 3.2. A processor may be in a situation where it gives up all its clusters, which makes it idle. To remedy this, a processor which owns a number of clusters smaller than a certain threshold (fixed by the user) is not allowed to give up its clusters.

All the communications are performed with the help of non blocking MPI functions for sending or receiving messages.

#### 3.3.4. Termination and Convergence

The function **termination()** detects the convergence of the parallel algorithm. As shown in figure 2 this function is called at the end of each iteration. It is implemented as follows. First, the set of processors is implicitly configured into a ring. This is easily implemented since MPI provides each processor an identification number ranging from 0 to $N-1$, $N$ being the number of processors. 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 for which the processor $i$ is responsible. A processor is idle if it has processed all its own clusters and its requests for a 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 around 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 around 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.

```
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 */
    /* amongst 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*

## 4. Implementation

### 4.1. Radiosity Method

Our radiosity method relies on projection techniques with wavelets as basis functions and offers two resolution schemes: Gathering and shooting. More precisely, our wavelet radiosity is hierarchical and the degree of the wavelet basis functions varies from 0 to 3, which corresponds to a number of vanishing moments ranging from 1 to 4. We used multi-wavelet basis functions [17]. As said before, our radiosity methods does not employ any clustering-based accelerating techniques as in [13, 14, 15]. The first reason of this is that these techniques lack precision since they do not account precisely for the orientation of the patches within the emitting and receiving clusters (which is not acceptable in lighting simulation). The second reason is that in a building interior the number of actual clusters (in the sense of these accelerating techniques) is low compared to the number of input surfaces.

Our method is spectral. In other words, it makes use of the spectral form of light and of the reflectance of the materials. The number of wavelength samples depends on the spectral power distribution of the used light sources [18].

### 4.2. 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.

### 4.3. Modeling and Partitioning into 3D cells

Only building interiors are considered as scenes. These 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 and geometric transformation of all these entities.

To partition the scene (which is not necessarily axial) into 3D cells, we have used the method described in [12].

### 4.4. Radiosity computation

The radiosity computation starts once partitioning and clustering have been completed. During the 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 a patch) is identified by its number and the number of the cluster containing it. This identification makes possible read and write (from or to disk) of links between surface elements belonging to different clusters.

### 4.5. 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, several experiments have shown many swaps of the data resident in memory which did not require any swap. This is due to the

```
ParallelRadiosity(S) {
    ListOfClusters S ; /* List of integers associated with */
                      /* the clusters assigned to the processor*/
    CYCLE : Cycle passing through all the vertices of OVG built from S ;
    Cluster C ;
    Boolean convergence = FALSE;

    while ( not convergence ) {

        /* Determine a cycle passing through all the clusters within S */
        CYCLE = TravelingSalesman(S);
        /* Gathers energy for each cluster in the cycle */
        For each cluster C in CYCLE {
            GathersParallel(C);
            /* write back the updated cluster C on the local directory */
            WriteOnLocalDirectory(C);
        }

        /* When all the clusters in S 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) {
    P : Set of processor numbers;
    C : Cluster;
    Env : Set containing a receiving cluster and the clusters visible from it ;
    Env = Download C and the clusters it views;
    For each patch S_i in C {
        For each patch S_j in Env {
            /* S_i gathers energy from S_j */
            GatherEnergyFromSurface(S_i, S_j);
        }
    }
    /* Check if requests for clusters have been sent by other processors for load
       balancing; P is the set of numbers identifying the processors which effectively
       sent a request for a cluster */
    P = CheckIfRequestForCluster(); /* Non blocking receipt */
    if (P ≠ ∅) SendClusters(P); /* Send clusters to each processor of P */
}
```

**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); /* Non blocking sending */
        C = WaitForReply(Proc_i) ;
        If C ≠ NULL return C ;
    }
}
```

**Figure 4:** *Request for cluster during load balancing*

```
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 stemming from */
        /* other processors Proc_j in order to avoid deadlock*/
        For each processor Proc_j {
            if ReceiveRequestForJob(Proc_j); /* non blocking receipt */
                ReplyToRequest(Proc_j) ;
        }
    }
    return C;
}
```

**Figure 5:** *Wait for reply*

fact that the fragmentation of the main memory is inefficiently managed by these routines. Consequently, we have decided to manage memory by ourselves [8].

## 5. Results

This section provides some results for one scene, a building composed of three floors (figure 6). It is made up of 57,786 input surfaces of which about 2,000 are light sources. This scene has been partitioned into 615 cells containing 12,942 clusters. The number of resulting surface elements is about 360,000 while the number of links (between surface elements) is 3.7 million. Seven wavelengths are used (see [18] for further details) for representing the radiosities and the material reflectances, and linear wavelet basis functions. Therefore, 4 and 16 projection coefficients are needed for expressing radiosity and the kernel of the radiance equation respectively. The storage of all these data structures would need about $3Gb$ which is not offered by most 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 computers used are given in table 1. The preprocessing corresponding to the graph partitioning takes about 10 minutes.

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

The speed ups given in this table have been computed with respect to the run time on one SUN1 which is the most powerful computer amongst those used for our tests. We can remark that the speed ups obtained with two and three processors exceed the ideal speed up. This can be explained as follows. The resolution process repeatedly selects a receiving cluster, then downloads it together with its visible clusters either from the disk cache (if they are already in the cache) or the disk. Each computer has its own data and local disk cache. The more the processors, the smaller the data assigned to each processor and the higher the efficiency of the local disk caches. However, when one processor is used, the cache capacity is low relative to the large data size, which entails a high number of cache misses.

The obtained speed ups are encouraging. For SGI3 which is a parallel computer with a shared memory

| 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.*

| 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 specific hardware for allowing parallel disk Inputs/Outputs.

It is well-known [19] that our wavelet radiosity is slower than classical hierarchical radiosity, but is more elegant and more accurate. Therefore, comparing our parallel algorithm with another one relying on classical hierarchical radiosity (using either clustering techniques or not) does not make sense.

We could not compare our approach with those of [7, 10] as we are using different kinds of computers.

## 6. Discussion

As shown, our parallel hierarchical radiosity algorithm can be implemented on a heterogeneous network of computers or on a parallel machine with a distributed or shared memory in a straightforward manner. Since the data structure used resides on the same disk, the performance of the algorithm strongly depends on the time needed for the numerous disk accesses. To improve the performance of our algorithm a reduction of this time is indispensable. To this end, for some parallel machines, a hardware system allows the processor nodes to access the disk simultaneously. Unfortunately, this kind of machine is not available at our institute.

In the 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 may be performed in parallel. Secondly, the common directory could 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 the MPI communication library. It does not rely on a master-slave environment and can be implemented on any kind of heteregeneous network of computers or parallel machine provided that they have MPI. In our approach the communication overheads are reduced since only a few small sized messages are required. To cope with the problem of memory size, only a subset of the database resides in memory. This subset is determined by the partitioning into 3D cells, clustering and visibility calculations. In order to reduce the disk transfers due to downloading these subsets of clusters, an ordering strategy based on the traveling salesman algorithm is used. In another paper [12] we have shown that this ordering strategy outperforms others such as: Greedy, random, maximum energy, back tracking, etc. The use of local and common directories ensures data coherence. The advantage of our parallel algorithm is more noticeable in the case of parallel computers with processors accessing the same disk because no messsages are needed to access the common 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

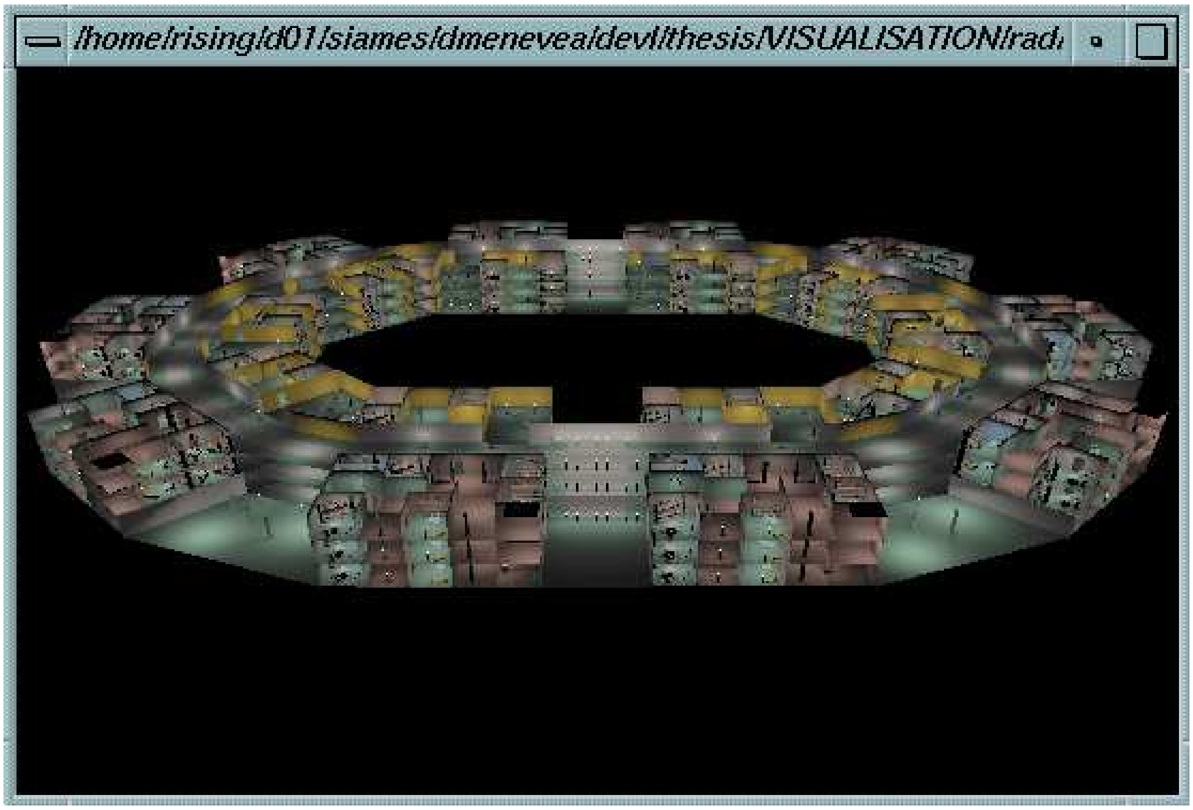


**Figure 6:** *Image of a 3 floor building once the complete global illumination has been performed.*

technique seem efficient. Finally, as our group iterative resolution is based on the Gauss Seidel method, a proof of its convergence is given in the appendix.

### 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. B. Arnaldi, X. Pueyo, and J. Villaplana. On the division of environments by virtual walls for radiosity computation. In Springer Verlag, editor, *Second Eurographics Workshop on Rendering, Barcelona*, pages 198–205, 1994.

6. 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.

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

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

9. Alan Chalmers and Jonathan Tidmus. *Practical Parallel Processing, An Introduction to Problem Solving in Parallel.* Thomson Computer Press, 1996.

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

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

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

13. F. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and

objects clusters. *IEEE Visualization and Computer Graphics*, 1(3), 1995.

14. Brian Smits & James Arvo & Donald Greenberg. A clustering algorithm for radiosity in complex environments. In *Computer Graphics Proceedings, Annual Conference Series*, pages 435–442, 1994.

15. S. Gibson and R.J. Hubbold. Efficient hierarchical refinement and clustering for radiosity in complex environments. *Computer Graphics Forum*, 15(5):297–310, December 1996.

16. 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.

17. Steven J. Gortler & Peter Schroder & Michael F. Cohen & Pat Hanrahan. Wavelet radiosity. In *Computer Graphics Proceedings, Annual Conference Series*, pages 221–230, 1993.

18. E. Zeghers, S. Carre, and K. Bouatouch. Error-bound wavelength selection for spectral rendering. *The Visual Computer*, 13:424–434, December 1997.

19. A. J. Willmott and P. Heckbert. An empirical comparison of progressive and hierarchical radiosity. Eurographics Workshop on Rendering, Springer Verlag, June 1997.

20. 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 of 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-emittance 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.

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

$M^L$ 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 parallel 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 matrix $M^L$ 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:

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

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 now show that our group iterative method converges.

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

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

We get then:

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

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

We also have the following equivalence:

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

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

First $r(D^{-1}(U+L)) < 1$ may be shown by using corollary 5.6 on page 125 of Young [20] (this corallory has also been used in [7]).

**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 $r(Q_2^{-1}R_2) \leq r(Q_1^{-1}R_1)$.

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 $r(D^{-1}(U+L)) \leq r(I^{-1}N) = r(N) < 1$ $\square$.

Consequently our group iterative method based on the Gauss Seidel method converges.