# Parallel Hierarchical Radiosity*

R. Şireli
Computer Engineering and
Information Science Department
Bilkent University
Ankara, Turkey

A. Gürsoy
Computer Engineering and
Information Science Department
Bilkent University
Ankara, Turkey

**Abstract** *This paper discusses parallelization of a hierarchical radiosity algorithm used in realistic image generation. Due to its highly irregular computational structure, hierarchical radiosity algorithms do not yield easily to parallelization on distributed memory machines. Dynamically changing computational pattern of the algorithm results in complex parallel program structure and also in severe load imbalances. We present design and implementation of a parallel hierarchical radiosity algorithm/program which separates the complexity of parallelization from the computational algorithm by using proxy-objects. This allows us to reduce communication cost and also to implement dynamic load balancing mechanisms easily. We also discuss various methods of partitioning of computational domain and present their effect on the performance.*

*Keywords:* Distributed memory, dynamic load balancing, parallel programming, computer graphics, global illumination, n-body

## 1 Introduction

An important problem in achieving realistic image generation is computing the distribution of light in an environment. Radiosity approach has been proposed to solve this problem [1]. The method requires all of the surfaces in a scene to exchange light energy according to some configuration factors. Calculation of these configuration factors is the most important and time-consuming part of the radiosity method. The input scene is represented as a collection of polygons (called patches). The radiosity gathered by patches from the other patches are formulated as a linear system of $n$ (number of patches) equations. Potentially, there can be interaction between every pair of patches in the scene which results in $O(n^2)$ complexity for finding configuration factors between patches. Hierarchical radiosity is a recent approach that reduces computational requirements by careful error analysis. It has its idea from the solution methods of N-body problems. N-body solution methods approximate the interactions between well separated groups of objects by a single interaction. Consequently many expensive calculations that have little effect on the accuracy of the solution can be avoided. Hierarchical radiosity applies this approach by eliminating the interactions of surfaces that do not effect the accuracy of the overall solution. As stated in [2], hierarchical techniques reduce the complexity of radiosity computations from $O(n^2)$ to $O(n + k^2)$, where $k$ is the number of input patches, and $n$ is the total number of resultant elements (patches) in an environment ($n \gg k$). Although hierarchical approach has greatly reduced the amount of calculations, satisfactory results still cannot be obtained in terms of time. Parallel processing is one of the most practical ways to reduce the computation time further. Due to its highly irregular computational structure, hierarchical radiosity algorithms do not yield easily to parallelization on distributed memory machines. In this pa-

*This work partially supported by TÜBİTAK under the grant EEE-AG 247.

per, we present design and implementation of a parallel hierarchical radiosity program. This work consists of three main parts: (a) design of a parallel algorithm and dynamic load balancing mechanisms, (b) design and implementation of a parallel object-oriented program, and (c) performance study and impact of various design decisions. The parallel algorithm is based on the sequential hierarchical radiosity algorithm described in [3]. The parallel object-oriented implementation has been done using Charm++ programming environment [4].

## 2 Related Work

Parallelization of hierarchical radiosity on distributed memory computers is not easy due to its prohibitive characteristics of input data. Most of the existing studies are on shared memory machines which are generally based on management of a task pool [5, 6, 7, 8]. Distributed memory implementations have not produced satisfactory results as others so far. Garmann et al. [9] presented a parallelization study on CM-5. Their speed-up shows the difficulty of parallel processing for hierarchical radiosity algorithm, 8.4 on 64 processors. In [10], 11-fold speed-up on 32 processors was reported. In [11], Feng and Yang claim that they have developed an efficient implementation. Their study is based on CV-sets for visibility computations. They presented performance results of complex (large) scenes up to 8 processors. However their input scenes are composed of fairly independent subscenes which are slightly interacting with each other. Therefore CV-set approach assigns each subscene to a different processor which results in good speed-up. However, CV-set approach will not handle dynamically changing load balance for arbitrary input scenes.

## 3 Characteristics of Hierarchical Radiosity Algorithm

As an important property of the algorithm, the patches are subdivided during the solution process. Evaluation of an interaction between two patches may result in such a subdivision. Refinement of an interaction is applied when the radiant flux between the patches is big enough to effect the accuracy of resultant radiosity. After refinement, the old interaction is removed and new interactions from the undivided patch to the subpatches of the divided patch, which must also be evaluated, are established. As a result, the data to be processed dynamically grows and achieving good load balance becomes more difficult.

If the patches of an interaction are located on different processors, the radiosity values need to be communicated. The interactions between patches depend completely on the input scene and don't have a fixed pattern. The message for a patch includes only a few floating point numbers, that is, there will be many small sized messages to be exchanged. Therefore, startup cost (basically at the software level) of sending/receiving messages are expected to dominate the communication cost.

## 4 A Parallel Scheme

The major work done in hierarchical radiosity is to calculate interactions between pair of patches that are visible to each other. This involves determining if any two patches, say patch-$i$ and patch-$j$, are visible to each other or not, and then calculating the radiosity from patch-$i$ to patch-$j$ and from patch-$j$ to patch-$i$ (interaction-$ij$ and interaction-$ji$). Checking the visibility constitute a major part of the computation in non-hierarchical approaches, however, in the hierarchical approach, the input patches can be duplicated on each processor to calculate visibility on each processor locally.

We can map interactions to the processors which own the interacting patches. If both patches of an interaction are on the same processor, then we can map the interaction to that processor. However, if one of the patches is on a different processor, we can map interaction $i$-$j$ to processor-patch-$i$ (processor that owns patch-$i$) and interaction $j$-$i$ to processor-patch-$j$. So it requires total 4 messages (sending patch data and receiving updated data per interaction) and interactions $i$-$j$ and $j$-$i$ are evaluated in parallel. If we map both interactions $i$-$j$ and $j$-$i$ to one processor, either processor-patch-$i$ or processor-patch-$j$, then the number of messages will be two and both interactions will be evaluated on one processor one after the other. Since there are many interactions in the radiosity calculations that can keep processors busy, we decided to map interactions $i$-$j$ and $j$-$i$ to one processor to reduce the communication cost. But, how will processor-patch-$i$ and processor-patch-$j$ decide which one will evaluate the interactions? Making this decision dynamically is again attractive from load balancing point of view, however, it will cause communication between processors. Instead, a static decision scheme which eliminates communication has been used. Both interactions between patch-$i$ and patch-$j$ are assigned to the processor who owns the patch with larger area. The rationale behind this is that, the larger patch will most likely be divided into smaller subpatches and interactions can be pushed down more easily.

### 4.1 Decoupling computational algorithm and parallelization management: Proxy Patches

Since the interaction pattern is quite irregular, whatever the mapping is, there will always be many across-processor interactions. If we design the parallel software such that each interaction object is responsible for gathering the data that it needs, this will result in multiple messages for the same patch data (because a patch might interact with many other patches). What we need is to send the patch data in a single message and allow interactions at the remote processor to share the same data. Interaction objects can do this by communicating with each other, and figure out which ones need the same data etc. But this will result in inefficient and complex code for interaction objects. In order to simplify the design of interaction calculations (i.e., free them from parallelization issues and let them do only calculations) and also reduce the number of messages, we propose to represent patches in remote processors by special objects called *proxy patches*.

If we let each patch to update its proxies, then there will be communication for each of them. Instead, all the updates of proxies between any two processors can be handled together. This requires a communication mechanism which gathers the data of all patches on a processor and distributes them to only the processors that need them. A special coordination object, *proxy manager*, is used to maintain the consistency among patches and their proxies as illustrated in Figure 1. To evaluate interactions *1-3*, *1-4* and *2-3* shown in the figure, the proxy of patch-*1* is used by the interaction objects as if they are local patches. The advantages of using such a structure:

**(a)** Using proxies decreases volume of communication between processors. Proxies provide that there is at most one representative of each patch in a processor. Static parts of the proxy data are transferred only once to the relevant processors at its first visit.

**(b)** Proxies provide the interaction object to operate as if its patches are local (using inheritance of C++ language). This flexibility greatly removes complexity of interaction objects.

### 4.2 Partitioning and Dynamic Load Balancing

The goal here is to assign patches to processors such that the processors have approximately equal computational load and at the same time the communication across processors is manageable. Due to the irregular communication pattern, there will be communication between almost every pair of processors. However, we expect the proxy-patch design to greatly reduce number of messages. In addition, the partitioning method must try to assign highly interacting group of patches on the same processors to maintain locality.

A successful partitioning algorithm, first of

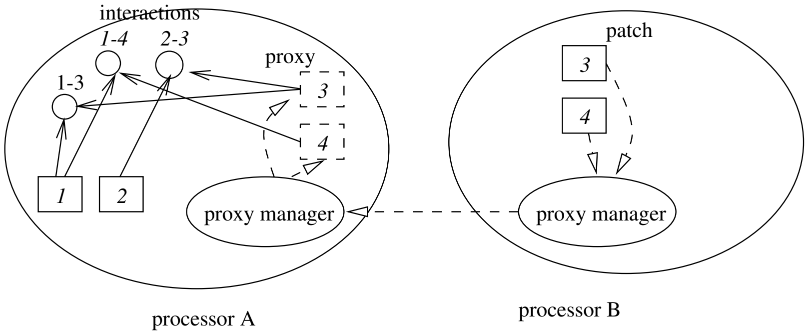


Figure 1: Proxy Patches

all, must estimate the computational load due to a patch as correctly as possible. A naive way would be to distribute equal number of patches to processors (thus assuming each patch has a uniform computational load). However, the computation per patch (interactions) is highly dependent on the visibility. A better approach would be the current number of interactions which the patch is involved. But this is not enough either, because, as the patches get refined, the number of interactions changes dynamically. We have developed a way to predict the load of a patch involving its area, current interaction list, and visibility information which results in better estimation. Assume that the current interaction list of a patch, patch-$i$, is known. Then, the number of total interactions that might be added to interaction list of patch-$i$ and of its subpatches can be estimated as follows:

$$f_n(i)\sum_{j\in I}[f_n(j)f_v(i,j)f_b(i,j)]$$

where

$\mathbf{f_n(i)}$ is the maximum number of subpatches that the patch-$i$ can be subdivided,

**I** is the current list of patches that patch-$i$ is interacting,

$\mathbf{f_v(i,j)}$ **and** $\mathbf{f_b(i,j)}$ are factors based on visibility and radiosity of patches $i$ and $j$. As visibility between $i$ and $j$ increases or $i$ and $j$ have large values of radiosity then it is more likely that these patches will be subdivided and more interactions will be created.

Once we can estimate the computational load of patches, the initial set of patches can be partitioned among processors. This partitioning must be in such a way that the load of each processor is similar. We experimented with various methods such as random distribution, input-order, and oct-tree based decomposition using the load estimation formula given above.

**Random distribution:** Patches are assigned to processors randomly. Randomness may produce equal load balance, however nearby objects which might have strong interactions might be placed on different processors.

**Oct-tree based spatial decomposition:** Nearby patches are forced to be on the same processor. For this, we have used the space decomposition based on an oct-tree structure which is already constructed for visibility calculations. The criterion of oct-tree based distribution is only based on the distance of patches and this may not reflect truly the interaction relationship (for example, two nearby patches might not be visible to each other).

**Input order:** This is another form of spatial distribution. When the input file is prepared, usually, patches of the same object are put one after the other. That is, patches of the same object are expected to be defined in the input file successively. This is different from oct-tree based decomposition in the following way: If we have a scene with 2 rooms, we usually define the objects in one room first, then define the objects in the other room. With input order, the objects of a room will possibly be assigned to the same processor. However, in octtree based decomposition, room boundaries are not taken into account. Objects from different rooms might be assigned to the same processor even there is a wall between them which prevents interactions.

We have also developed a dynamic load balancing scheme. The scheme is based on patch migration from highly loaded processors to less loaded ones. We have experimented with a few heuristics to select which patches to which processors to send. Instead of migrating individual patches, we migrate a patch with its all subpatches to maintain better locality.

# 5 Experimental Results

We conducted experiments to investigate the impact of partitioning and load balancing techniques discussed in the previous section. The runs were conducted on a 24-processors Parsytec-CC distributed memory machine. The details of this performance study is presented in [12]. We have used several input scenes for the performance studies. Scene-1 has 30 input patches. It is a box containing a rectangular trapezoid prism and a triangular prism inside. The wireframe model and the shaded image of Scene-1 (after the radiosity calculations) are shown in Figure 2 and Figure 3. Scene-1 is an example of simple scene which does not have a lot of computation. Other scenes are more complicated. For example, as shown in Figure 2 (lower part), they have varying visibility structures and more patches. A summary of the effect of the various design issues on the performance is given below:

**Impact of Load Estimation:** The results show that the load estimation function that we have developed performs better than the uniform or the number-of-interactions approach. Table 1 illustrates the speed-up of the parallel algorithm where the computational load of patches (used in partitioning) are estimated with different ways (uniform, number of interactions, and the load estimation formula). For some cases, the execution time was reduced by half with the estimation formula that we have developed. In these runs, the input-order partitioning method was used and the numbers given in each scene entry of the table are the initial and final number of patches.

Table 1: Comparison of load estimation methods: (u: uniform, i: number of interactions, f: load estimation formula), #P: number of processors.

| Scenes | # P | u | i | f |
|---|---|---|---|---|
| Scene-1 30 4677 | 1 | 57.8 | 57.8 | 57.8 |
| | 2 | 58.2 | 34.9 | 35.2 |
| | 4 | 35.3 | 20.0 | 17.5 |
| Scene-2 216 6610 | 1 | 171.8 | 171.8 | 171.8 |
| | 2 | 177.3 | 172.9 | 141.9 |
| | 4 | 173.6 | 158.4 | 80.8 |
| | 8 | 177.3 | 94.0 | 61.2 |
| | 16 | 176.7 | 61.0 | 60.7 |
| Scene-3 252 14170 | 1 | 198.0 | 198.0 | 198.0 |
| | 2 | 129.0 | 119.0 | 117.3 |
| | 4 | 90.5 | 67.8 | 60.0 |
| | 8 | 60.6 | 50.0 | 39.6 |
| | 16 | 46.4 | 31.2 | 34.0 |

**Impact of Initial Patch Distribution:** We have observed that none of the initial distribution methods are superior to any other one. This result is due to highly irregular structure of the computations. A static distribution based on initial information of the patches does not guarantee load balance throughout the execution.

**Impact of Dynamic Load Balancing:** With dynamic load balancing, consistent improvements in execution time have been observed as the number of processors increases. Table 2 gives the results of the dynamic load balancing experiment. These results show the necessity of dynamic load balancing in hierarchical radiosity calculations.

Table 2: Static Versus Dynamic Load Balancing

| # processors | timings for static versus dynamic load balancing (seconds) | | | |
|---|---|---|---|---|
| | Scene-1 | Scene-2 | Scene-3 | Scene-4 |
| 1 | 57.8 | 171.8 | 198.0 | 213.3 |
| 2 | 34.2/33.3 | 110.8/96.9 | 116.7/116.7 | 126.8/126.8 |
| 4 | 21.0/17.8 | 76.6/55.0 | 60.7/60.7 | 65.0/65.0 |
| 8 | 16.0/11.9 | 60/35.5 | 40.5/35.5 | 39.5/35.5 |
| 16 | 17.7/9.2 | 60.0/24.1 | 29.2/23.3 | 28.9/26.1 |
| 24 | 18.8/9.3 | 62.0/18.9 | 25.5/19.0 | 28.6/22.1 |

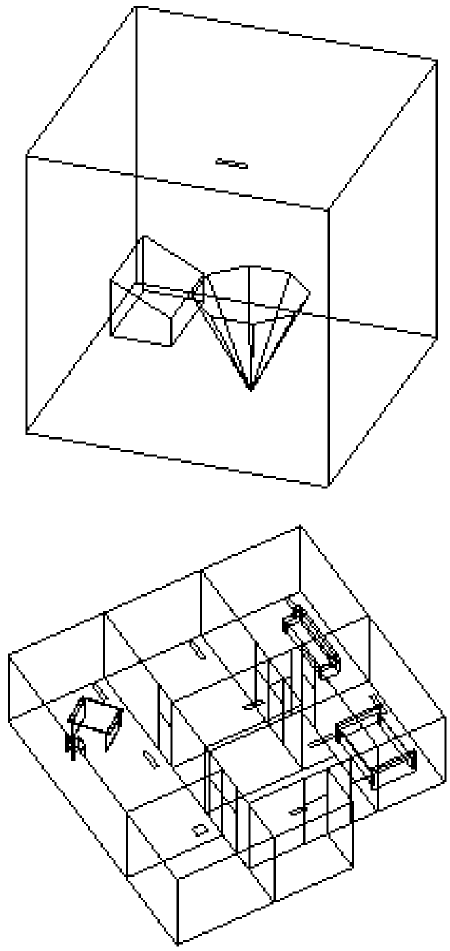

Figure 2: Wireframe model of some input scenes.

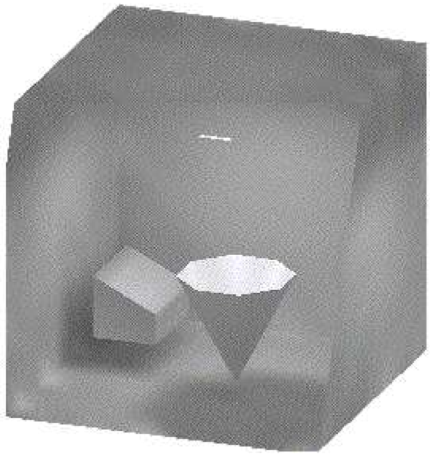

Figure 3: Shaded image of Scene-1

## 6 Conclusion

We presented the design an implementation of an hierarchical radiosity algorithm on distributed memory machines. Due to its highly irregular computational structure, hierarchical radiosity algorithms do not yield easily to parallelization on distributed memory machines. Dynamically changing computational pattern of the algorithm results in complex parallel program structure and also results in severe load imbalances. We have developed a mechanism to separate the management of parallelization from the computational algorithm based on proxy objects and proxy manager objects for accessing remote data. This simplified greatly the design and implementation of the program, implementation of dynamic load balancing mechanisms, and also helped us reduce the communication cost. The initial results shows that an accurate estimate of the load per patch and dynamic load balancing are important to get good speed-ups. As future work, the performance of the algorithm and the program will be improved by optimizing critical parts of the code and by incorporating the patch-distribution heuristics used in the static load balancing part to the dynamic load balancing phase.

## References


[1] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, Jul 1984.

[2] M. F. Cohen and J. R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, NY, 1993.

[3] P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, Jul 1991.

[4] L. V. Kale and S. Krishnan. Charm++: A portable concurrent object oriented system based on C++. In *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications*, ACM Sigplan Notes, pages 91–108, Sep-Oct 1993.

[5] A. Podehl, T. Rauber, and G. Runger. Scalability and granularity issues of the hierarchical radiosity method. In *Euro-Par Vol. I*, pages 789–798, 1996.

[6] C. Vinod and V. Chaudhary. Parallel hierarchical radiosity algorithms : Case study on a dsm-coma architecture. In *Int'l Conf. on Parallel and Distributed Computing Systems*, ISCA, Sept 1995.

[7] J. Richard and J. Pal Singh. Parallel hierarchical computation of specular radiosity. In James Painter, Gordon Stoll, and Kwan-Liu Ma, editors, *IEEE Parallel Rendering Symposium*, pages 59–70. IEEE, Nov 1997. ISBN 1-58113-010-4.

[8] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. Load balancing and data locality in adaptive hierarchical n-body methods: Barnes-hut, fast multipole, and radiosity. *Journal of Parallel and Distributed Computing*, 27:118–141, Jun 1995.

[9] R. Garmann, C-A. Bohn, and H. Muller. Parallel hierarchical radiosity on the cm-5. Technical Report 557, Department of Computer Science, Dortmund University, 1994.

[10] J. P. Singh, A. Gupta, and M. Levoy. Parallel visualization algorithms: Performance and architectural implications. *IEEE Computer*, 27(7):45–55, Jul 1994.

[11] C-C. Feng and S-N. Yang. A parallel hierarchical radiosity algorithm for complex scenes. In *IEEE Parallel Rendering Symposium*, pages 71–77, 1997.

[12] Ahmet Reşat Şireli. Parallelization of hierarchical radiosity algorithms on distributed memory computers. Master's thesis, Bilkent University, Jan 1999.