The Journal of Supercomputing, 31, 249–263, 2005

PHR: A Parallel Hierarchical Radiosity System with Dynamic Load Balancing

ALI KEMAL SINOP kemalp@ug.bilkent.edu.tr
TOLGA ABACI* tolga.abaci@epfl.ch
ÜMIT AKKUŞ† umita@microsoft.com
Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey

ATTILA GÜRSOY agursoy@ku.edu.tr
Department of Computer Engineering, Koç University, Rumeli Feneri Yolu, 34450 Sariyer, Istanbul, Turkey

UĞUR GÜDÜKBAY gudukbay@cs.bilkent.edu.tr
Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey

Abstract. In this paper, we present a parallel system called PHR for computing hierarchical radiosity solutions of complex scenes. The system is targeted for multi-processor architectures with distributed memory. The system evaluates and subdivides the interactions level by level in a breadth first fashion, and the interactions are redistributed at the end of each level to keep load balanced. In order to allow interactions freely travel across processors, all the patch data is replicated on all the processors. Hence, the system favors load balancing at the expense of increased communication volume. However, the results show that the overhead of communication is negligible compared with total execution time. We obtained a speed-up of 25 for 32 processors in our test scenes.

Keywords: hierarchical radiosity, distributed memory architectures, load balancing

1. Introduction

Synthesis of photo-quality images is a difficult and time-consuming computer graphics problem. Essentially, the problem involves extensive simulation of the real-world light events such as reflection of light among surfaces. The ultimate goal is to develop a real-time interactive photo-realistic image generator. However, to get high-quality images utilizing state-of-the-art solutions, we have to wait for minutes even for the simplest scenes consisting of tens of polygons.

One class of solutions is the radiosity approach [13]. Radiosity, which is the basis of our work, can simulate area light sources with the underlying assumption that only diffuse reflection takes place between surfaces. Those surfaces that can be handled by radiosity algorithms are called Lambertian surfaces, which are ideal diffuse reflectors. The surfaces are subdivided into a mesh of elements, called *patches*, which have reflectivity and lighting properties. Hierarchical radiosity is an improvement over

*Present address: Virtual Reality Laboratory, Swiss Federal Institute of Technology, 1015 Lausanne, Switzerland.
†Present address: One Microsoft Way, Building 40, Office 5260, Redmond WA, 98052, USA.

traditional radiosity solutions to overcome the computational limitations inherent in these solutions.

We propose a parallel implementation of the hierarchical radiosity for distributed memory architectures. Recent work on this topic generally uses task stealing approaches to achieve a nearly linear speed-up, but such approaches are not suitable for distributed memory architectures where the initiation cost for message passing is high. Our work relies on evaluating interactions one level at a time in each iteration. When the processing of one level is finished, the load of each processor is computed and interactions are redistributed in order to keep load balanced. To allow efficient distribution of interactions, the scene geometry and patch information is duplicated in all of the processors without significant impact on the overall performance.

The rest of the paper is organized as follows. We discuss the related work on global illumination, hierarchical radiosity, and parallel implementation of the radiosity approach in Section 2. The parallel algorithm that we propose for hierarchical radiosity utilizing collective communication is described in Section 3. The details of representing the patches for communication between processors, evaluating the interactions, and load balancing process are described in this section. The results of the performance experiments to measure the speed-ups and the images produced using the implementation are presented in Section 4. Conclusions are given in Section 5.

2. Related work

2.1. Radiosity

The radiosity approach is based on thermodynamics and heat transfer. Instead of heat, light energy is actually traveling between surfaces (or patches). The approach proposes to exchange energies between surfaces with respect to some configuration parameters, such as visibility and form factors. In fact, this corresponds to solving the *global illumination equation* [17], which, in the special case where only diffuse, opaque surfaces are involved, takes the form of an integral equation of the form below:

$$b(x) = e(x) + \rho_d(x) \int_\Gamma dx' \frac{\cos\theta_i \cos\theta_j}{\pi r^2} v(x, x')\, b(x') \tag{1}$$

where $\rho_d(x)$ gives the diffuse reflectance, $e(x)$ gives the radiant emitted flux density, $b(x)$ gives the radiosity, $v(x, x')$ is the visibility function, and Γ represents the whole set of surfaces in the scene. The visibility function has the value of 1, if x and x' are visible to each other, and 0, if they are occluded. The angle θ_i (or θ_j) relates the normal vector of element i (or j) to the vector joining the two elements.

Calculation of this integral is done either by Monte-Carlo methods or by discretizing the scene into n finite elements [15]. Efficient solution methods for this system include Gauss-Seidel, Jacobi, and specialized methods such as progressive radiosity [7]. Besides, higher degree elements and Galerkin methods have been used to improve accuracy of the solutions [16].

2.2. *Hierarchical radiosity*

Hierarchical radiosity [14] is proposed to overcome the deficiencies of the progressive radiosity approach. It reduces computational requirements by careful error analysis. The idea behind hierarchical radiosity is the same as the *N*-body problem solution [3]. These problems share many similarities, mostly due to the similar nature of the interactions between particles in the *N*-body problem and the interactions between patches in the radiosity method. During the solution of the *N*-body problem, interactions between separated object groups (*clusters*) are computed as a single interaction. Hierarchical radiosity uses the same idea and does not compute the interactions that do not affect the accuracy of the whole image. However, it is important to note that while the *N*-body algorithms construct the upper portion of the hierarchy tree by forming groups of particles, the hierarchical radiosity algorithm constructs the lower portion of the tree by subdividing the patches.

Hierarchical radiosity algorithm stated in [14] recursively subdivides the initial patches, forming a quadtree, with respect to the form factor estimations. After the form factors are calculated this way and stored in a hierarchical structure, standard techniques of solving the radiosity system can be employed. In the variant of hierarchical radiosity to be employed by our system, the *BF*-refinement technique is used, which takes into account the form factor estimations and the brightness values of the patches when performing subdivisions. This is essentially a combination of the form factor estimation and solution processes.

In contrast to other radiosity approaches, hierarchical radiosity subdivides only some of the patches at the finest resolution. Therefore, the overall computational complexity of the algorithm is reduced from $O(n^2)$ to $O(n + k^2)$, where k is the number of input surfaces, and n is the total number of resultant elements in an environment [8].

Hierarchical radiosity algorithm consists of three main steps, initial linking, interaction evaluation and push-pull.

- *Initial Linking*: Before beginning the hierarchical radiosity solution, interactions between all initial patches in the scene are computed. This step involves computing the form factor between every patch pair. For this purpose, disk approximation method can be used [22]:

$$F_{ij} = v_{ij} \frac{\cos\theta_i \cos\theta_j A_j}{\pi r_{ij}^2 + A_i} \tag{2}$$

  In this equation, v_{ij} is the visibility function, θ_i and θ_j are the angles between the element normals and the connections of their centers, r_{ij} is the distance between the elements, and A_i and A_j are the areas. The overall effect of area j on area i is built by multiplying F_{ij} by the outgoing energy from area j. The accuracy of this method can be increased by firing multiple rays.
- *Interaction Evaluation*: In this step, interactions for all of the patches are evaluated. This involves computing the energy transfer from patch i to j and vice versa. If the exchanged energy happens to be above some threshold, the patch with the larger area can be subdivided in order to increase the accuracy of the solution. Subdivision of a

patch does not effect other interactions; they remain between the same patches. Hence, interactions may be present in all levels of a patch's hierarchy.

– *Push-Pull*: Since interactions exist at different levels of a patch's hierarchy and different amounts of energy are accumulated at the nodes, it is necessary to bring the whole tree structure into a consistent state at the end of each iteration. This is achieved by first propagating the energy of upper nodes to the leaf nodes and then leading the total weighted energy of leaf nodes to the upper ones.

At each iteration, the interactions of each patch are evaluated, refining the patches and interactions as necessary. When all patches are done, the patch hierarchies are brought to a consistent state by push-pull step. If the total energy change in the whole scene is above some threshold, the iterations should continue, otherwise it is not necessary to carry on the computation, hence we can output the energies of leaf level nodes in the scene.

2.3. Parallelization of radiosity

A parallel radiosity simulation system is proposed in [12], which uses perceptually-based calculations to control simulation process. In this work, a novel algorithm for computing radiosity solutions on distributed shared memory (DSM) architectures is described, which uses a queue based scheduling system to process the sub-iterations—transfer of energy from a single source to a subset of the scene's receiver patches—eliminating the need for processor synchronization between iterations of the algorithm. Another parallel radiosity algorithm based on patch data circulation is proposed for distributed memory architectures [2]. Their work uses a global circulation scheme for parallel light distribution computations, reducing the total volume of concurrent communication. Renambot et al. also obtained good speed-ups on DSM architectures with their parallel radiosity implementations [19].

Parallelization of hierarchical radiosity, on the other hand, is more challenging. The dynamic nature of the hierarchical radiosity algorithm makes it very hard to equally distribute the computations on numerous processors. Several approaches for parallel implementations of hierarchical radiosity were proposed. The work of Sillion et al. [20] demonstrated an extension to the hierarchical radiosity algorithms on DSM architectures, in which a main process holds a queue of individual interactions, and when a processor finishes its job, it retrieves new interactions from this list, also enqueuing any new interactions resulting from subdivisions. A nearly perfect speed-up is obtained for some scenes (a speed-up of 39.4 on 40 processors), and good speed-ups are obtained in general (a speed-up of 25.7 on 30 processors). However, the slower shared memory is heavily used in this system, and the general computation process is slowed-down.

Good speed-ups are achievable in parallel implementations of hierarchical radiosity for shared memory architectures since hierarchical radiosity computations contain many fine grain sub tasks. However, there is not much work achieving high performance on distributed memory machines. The main reason behind this is that previous works such as [9] concentrate on distributing the scene geometry on different processors so as to be able to render very complex scenes, whereas our work aims to reduce the rendering time.

Another work on parallel hierarchical radiosity for distributed memory architectures is proposed in [1], which aims to reduce the rendering time instead of distributing the geometry. In this work, every processor keeps track of processed patches by other processors and as soon as it finishes processing a patch, it gets another unprocessed one. However, their work relies on heavy message passing in between iterations and they try to achieve load balancing at patch level. In our work, we try to achieve load balancing at interaction level, distributing every interaction individually.

Funkhouser describes an algorithm in which multiple hierarchical radiosity solvers work in parallel [10]. The set of polygons is distributed over the workstations, where partial radiosity solutions are computed for each part. Then, a master process collects and merges the solutions, iterating the process until convergence. For complex scenes, which can not be duplicated on every processor, this approach is well suited. They obtained a speed-up of 5.5 with eight SGI workstations for the Soda Hall model. Bohn et al. [4] proposed a parallel hierarchical radiosity algorithm on a *Connection Machine 5*, with a speed-up of 8.4 on 64 processors. Another work on cluster of PCs is reported in Sireli [21]. In their work, each patch is assigned to a processor, and a representative of the patch (proxy patch) exists in other processors if needed. Caballer et al. proposed a parallel version of the hierarchical radiosity algorithm, which utilizes the advantages of both shared and distributed memory architectures. Their motivation is similar to ours in the sense that they also focused on reducing the processing time, rather than rendering very large scenes [6].

Another similar work was proposed in [18], in which the scene is partitioned by a variant of K-means algorithm to allow the computation of large indoor environments on distributed machines.

In a recent work [11], a parallel algorithm that uses spatial partitioning of patches to processors to improve locality and asynchronous calculation to hide latencies is presented. They report almost linear speed-up upto 64 processors on CrayT3E. When the data needed on another processor, their algorithm initiates a request message to another processor to get the data. Achieving similar performances on a cluster of PC's with commodity networking would be difficult because of high startup costs of messages. We present a parallel algorithm more suited to cluster computing and our results show that good speed-ups are achievable.

3. A parallel algorithm for hierarchical radiosity utilizing collective communication

The major work done in hierarchical radiosity is calculating the interactions between pair of patches that are visible to each other. Therefore, interactions are the major units of computation to be distributed to processors. To calculate one interaction, a processor needs the current radiosity values and geometric information of the two patches involved in the interaction. A mapping of interactions to processors such that reduced communication and balanced work among processors, hence, is significant for performance. One way to map interactions to processors can be done by assigning patches to processors and associating interactions with the patches. The parallel algorithm presented here distributes patches to processors and each processor is responsible for handling interactions of its own patches (owner computes rule). If both patches of an interaction is on the same processor, the interaction can be calculated without any communication. Otherwise, the remote patch information

need to be communicated. Since the interaction pattern is quite irregular, whatever the mapping is, there will always be many across-processor interactions. If we let each interaction object to gather the data it needs, then there will be many messages communicated and possibly multiple messages for the same patch involved in more than one interaction. In order to reduce number of messages and simplify the design of interaction calculations, we represent a remote patch with a special type of patch, called proxy patch [21]. We utilize collective communication operations of MPI [5] to gather and scatter patches. The use of proxy patches and collective communication operations decreases the communication overhead. More importantly, the use of proxy patches allows the push-pull phase to be parallelized, since every processor performs the push-pull operation for only its local patches.

3.1. Representation of the patches

Unlike other earlier parallel algorithms such as [10], the scene data is not partitioned, instead every processor contains the information of the whole scene with all patches. The amount of memory used for keeping the patches is negligible compared with the amount required for holding the interactions. During the evaluations of interactions, a patch might be divided into smaller patches. This patch could be a local patch or a proxy patch. The newly created patches need to be given a handle that is consistent across processors (another processor might divide its proxy corresponding to the same patch). To globally identify the patches, each patch is given a global patch ID. Global patch IDs consist of two parts: root patch ID, and hierarchical information. The root patch IDs are assigned in a straight-forward manner, during the distribution of the initial geometry. However, the hierarchical information part is not so simple, since it requires an encoding scheme that describes the position of a patch in the hierarchy quad-tree. We have adopted a scheme where the hierarchical information part of an ID (hierarchy ID) consists of a sequence of two-bit groups. Each two-bit group is interpreted as a child number, from zero to three. The left-most bit with the value of one indicates the start of the hierarchy ID (An illustration of a 32-bit hierarchy ID is given in Figure 1). When read from left to right, a hierarchy ID for a patch effectively describes the

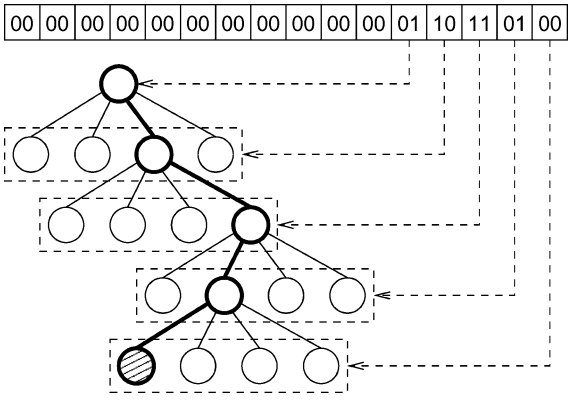

Figure 1. An illustration for a sample hierarchy ID. The nodes in the above tree denote patches.

path that must be followed starting from the root patch to find that patch. With this scheme, some of the ID space remains unusable. However, the scheme is easier to implement than most other schemes, and its simplicity results in higher performance.

3.2. The algorithm

The outline of the algorithm, which is carried out by all of the processors, is given below:

1. *Broadcasting of initial geometry.* The root processor reads the initial geometry from the input file, and broadcasts the data to all of the processors in the parallel system.
2. *Trivial patch-to-processor assignment and initial linking.* The initial linking operation consists of the computation of form factors for every possible interaction between the patches forming the initial geometry. Each processor is assigned an equal number of patches, and computes the form factors for the interactions of its own patches. It is possible that this phase be skipped if the user wishes to use form factors saved from a previous execution on the same scene.
3. *Re-assignment of patches to processors.* In this phase, the root processor gathers the form factors computed in the previous phase. The root processor evenly divides initial patches according to their area sum. Then, the processors are informed of which patches they are assigned, and receive the appropriate set of form factors.
4. *Computations and refinements for an iteration.* This phase is where hierarchical radiosity computation takes place. Each processor goes through its patches for evaluation of interactions. Evaluation of an interaction consists of the computation of the flux between two patches, and decision of whether to subdivide one of the patches or not. If refinement is to occur, then new interactions are created, and the form factors for those interactions are computed. This step takes the longest time to finish, and strong load imbalances occur at this step. To solve this problem, a new approach is proposed, which will be discussed in the next section.
5. *Sending back those proxies that were updated to their home processors.* Those proxies whose radiosity values were changed at the current iteration are sent back to their homes. The local patches for the proxies are created with the new value at their home processor if they were previously non-existent, or their values are updated if they were already there. This step features a total exchange between all processors.
6. *Push-and-pull phase.* At each node the push-and-pull phase is executed in order to bring the intensity values for local patches into a consistent state.
7. *Update of the proxy objects.* This step is the inverse of Step 5. To bring the system into a completely consistent state before the start of the new iteration, proxy patches should be updated with the intensity values that were computed in the push-and-pull phase. Similar to Step 5, the proxy patches for the local patches are created at their processors if necessary.
8. *Determine if there is a next iteration.* In this phase, all processors communicate to determine if there will be another iteration. If it is decided that the system has already converged, there will be no more iterations, and the algorithm stops.

3.3. Evaluating interactions

This is the most time consuming part of the hierarchical radiosity. In this step, each interaction is evaluated by computing the energy transfer between the patch pairs. Due to BF-refinement criteria, if the total exchanged energy exceeds a certain threshold, the patch with the bigger area is subdivided. In each subdivision, four new interactions are created, calculating the form-factors as necessary (Figure 2).

Since 16 rays are fired for each new interaction, subdividing the interactions dominate the evaluation step. For load balancing, we distribute the interactions among processors. Since each processor has the knowledge of all the patches, interactions can freely be exchanged between processors.

In standard hierarchical radiosity algorithms, interactions are evaluated in a depth first search (DFS) fashion; first the interactions of a patch are evaluated, and then the children's interactions are evaluated. However, this kind of approach has an inherent nature for load imbalance. An interaction, which is going to be subdivided to the maximum allowed level, will be assigned to one processor and a serious imbalance in the work-load will arise.

To overcome this problem, the interactions are evaluated in a breadth first search (BFS) fashion; first, only 0th level interactions are evaluated, and then 1st level interactions are evaluated, and so on. The algorithm for evaluating the interactions is given in Algorithm 1.

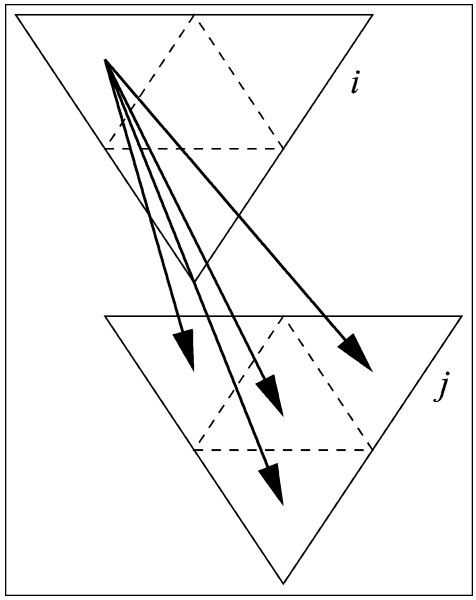

Figure 2. An illustration of the rays in the form factor calculation. Four of the sixteen rays fired from patch i to patch j for the calculation of F_{ij} are shown. For each subarea on patch i, four rays are to be fired.

Algorithm 1. Evaluating the interactions in *all Interactions* list, initialized with the interactions from initial linking.

```
curInteractions ← empty list
for all interaction in all Interactions do
  Evaluate interaction
  if interaction should be subdivided then
    Remove interaction from all Interactions
    Add interaction to prolificInteractions
  end if
end for
while prolificInteractions not empty do
  newInteractions ← empty list
  LoadBalance(prolificInteractions) {prolificInteractions contains the interactions
  that will be subdivided}
  for all interaction in prolificInteractions do
    Subdivide interaction and its associated patches
    for all newInteraction in children of interaction do
      Evaluate newInteraction
      if newInteraction should be subdivided then
        Add newInteraction to newInteractions
      else
        Add newInteraction to allInteractions
      end if
    end for
  end for
  prolificInteractions ← newInteractions
end while
```

In Algorithm 1, *LoadBalance* function distributes the interactions across processors to ensure equal workload.

3.3.1. Cost estimation. Although all interactions in the *newInteractions* are going to subdivide for only one level at a time, still not all interactions require the same processing power. However, predicting the cost of an interaction is very difficult. The time required for computing the form factor depends only on the geometry of scene—how many triangle and octree intersection tests are made, which is impossible prior to actually computing the form factor.

To predict the cost of subdividing an interaction, we can use the number of octree-node and triangle intersection tests performed for calculating that interaction's form factor, weighted with its parent interaction's cost by a certain factor, α_c.

3.3.2. Load balancing. In this section, we try to distribute the prolific interactions among the available processors such that the total amount of cost associated with each processor is equal to a certain constant. This is done by each processor in the following way:

1. *Compute the work load.* Each processor goes over its prolific interactions list, summing the cost of each interaction to determine its own work-load.
2. *Gather other processors' work loads.* A total exchange of associated work-loads between processors is done each processor so as to make every processor aware of others' work loads.
3. *Determine how much load will be sent to other processors.* A processor first computes the average load and declares each processor as a sender (should send some interactions to balance loads) or receiver. Then, the load each sender will send to each receiver is computed. Sender processors, sorted in descending order of their loads, send their loads to the first receiver with the smallest load.
4. *Distribute the interactions.* After having computed the load to send to each processor, sender processors distribute their interactions in a round-robin fashion to the receiver processors.

Although only an approximation for the load distribution is employed in Steps 3 and 4, generally the load tends to be distributed evenly, due to the high amount of interactions with varying costs.

4. Results

The Parallel Hierarchical Radiosity (PHR) system is implemented on a PC-cluster using C++ Programming Language and Message Passing Interface (MPI) for communication between processors.

4.1. Test scenes

For performance testing, three models from Soda Hall[1] are used. All models are augmented with ceiling lights. These models are;

– Room #320 (6350 triangles)
– Room #380 (31933 triangles)
– Room #420 (18510 triangles)

4.2. Measurements

The tests were carried on a Beowulf-cluster with 32 nodes. Each node is installed with 1 GBytes of RAM and an Intel P4 2.4 GHz CPU. Each node has a Fast Ethernet with a bandwidth of 100 Mbps. The nodes are linked together by a Fast Ethernet switch, which is a 48 port fast ethernet nonblocking switch with a 2.66 Gbps backplane and with gigabit uplinks to the master node.

4.3. Timings and speed-ups

Table 1 shows the total running time for each test scene. In Figure 3, the speed-ups obtained for each scene are shown according to the formulae $k_n = T_1/T_n$. For the *Room #380* test

Table 1. Timings for the test scenes (excluding initial linking stage).

	Total time (secs)		
Processors	Room 320	Room 380	Room 420
1	11506.4	7283.17	26871.6
4	2917.52	1382.74	6898.85
8	1548.05	737.04	3533
16	830.83	406.62	1894.23
24	583	302.43	1296.64
32	460	259.45	1012.3

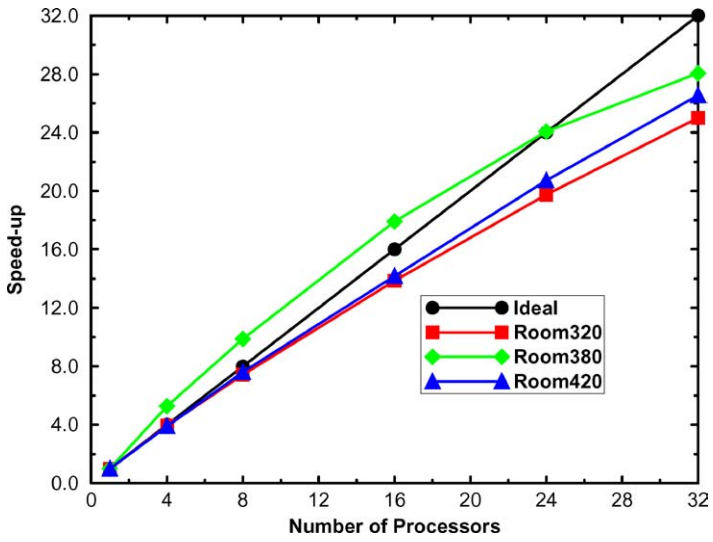

Figure 3. Speed-up measurements for the test scenes.

scene, we observe a super-linear speed-up behavior. Since the number of interactions are quite high, the virtual memory is heavily used in 1-processor case and this results in a huge slow down. When the number of processors increase, the interactions are divided among processors and therefore the memory of each processor becomes sufficient to hold all interactions assigned to them. This relieves processors the need to use virtual memory, causing a super-linear speed up to be measured.

In Table 2, detailed timings for the test scenes are given. It can be seen from Table 2 that the total time spent for migration is negligible ($<$2 secs for all scenes). Besides, duplicating the whole scene on every processor and making a total exchange in the *gather* and *scatter* phases do not take significant amount of time. However, the main issue is still the load balancing, and the idle times constitute a big portion of the time spent. The rendered images generated by our implementation for the test scenes Room320, Room380, and Room420 are given in Figures 4–6, respectively.

Table 2. Detailed timings for the test scenes.

		Detailed timings (in secs)					
Scene	Processors	Refinement	Idle	Migration	Gather	Push pull	Scatter
Room 320	1	11442.4	0	0.85	0.01	0.96	0
	4	2803.08	93.78	1.29	7.32	0.21	8.30
	8	1422.96	92.33	0.7	12.40	0.12	16.35
	16	710.12	86.36	0.66	14.14	0.06	16.86
	24	475.57	64.22	0.63	17.32	0.04	21.83
	32	350.83	54.23	0.61	20.44	0.03	30.68
Room 380	1	6061.09	0	6.43	0.24	0.76	0
	4	1266.34	79.51	1.09	6.8	0.16	7.5
	8	635.16	68.79	1.36	8.83	0.08	11.3
	16	319.55	51.68	0.8	11.97	0.04	16.45
	24	213.64	51.27	0.95	13.79	0.03	18.74
	32	160.54	50.23	0.81	17.78	0.02	27.06
Room 420	1	26377.1	0	1.12	0.1	1.99	0
	4	6551.15	275.22	1.94	7.1	0.22	8.46
	8	3283.25	199.18	1.19	9.38	0.11	12.08
	16	1646.79	200.07	0.83	12.87	0.05	19.52
	24	1103.52	145.3	0.64	15.31	0.04	22.54
	32	822.9	137.71	0.67	17.69	0.03	26.38

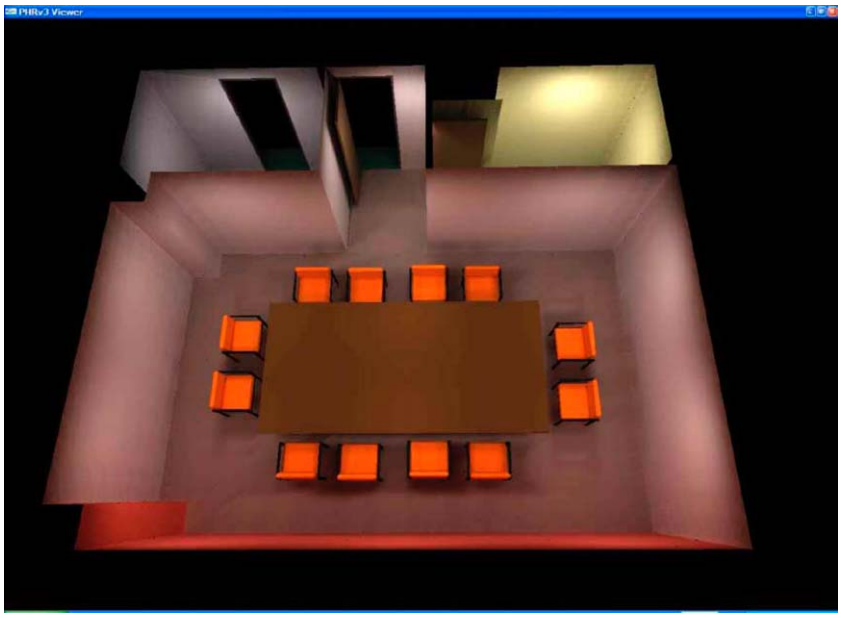

Figure 4. Rendering of the test scene Room320.

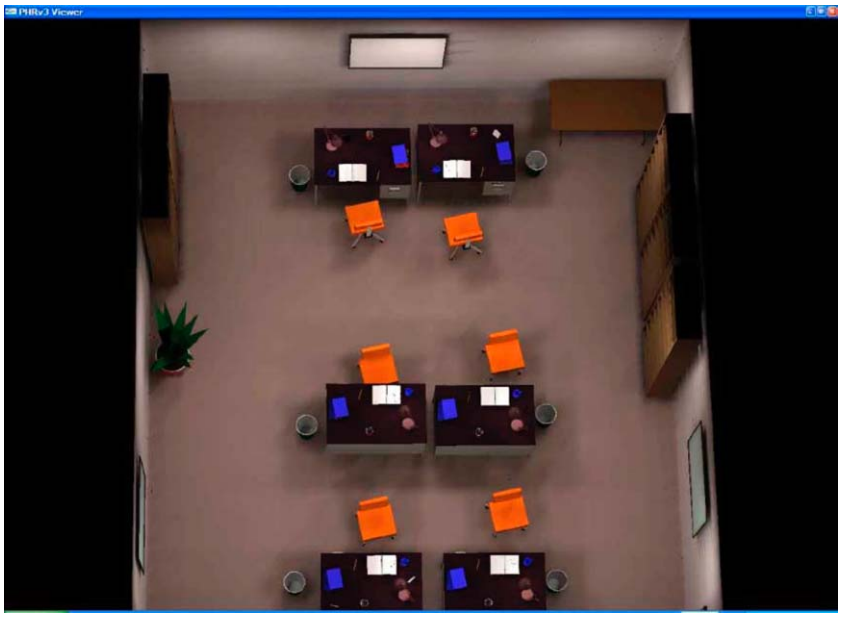

Figure 5. Rendering of the test scene Room380.

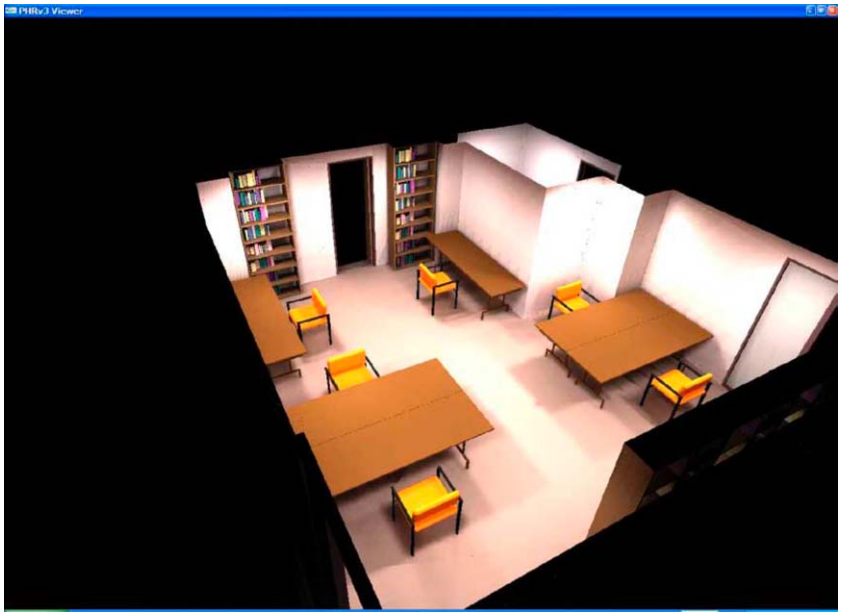

Figure 6. Rendering of the test scene Room420.

5. Conclusion

Since hierarchical radiosity works on a huge and dynamic tree structure, parallelizing it on distributed computers is a challenging problem. In order to obtain good speed-ups, dynamic load balancing techniques must be employed, but these techniques tend to require a lot of communication.

The most time consuming phase of hierarchical radiosity solution is the refinement phase, in which new interactions are formed. Due to the dynamic nature of algorithm, it is impossible to accurately predict how much time will be spent on subdividing a particular interaction without actually subdividing it. Hence, it is better to subdivide the interactions one level at a time in a BFS manner to equally distribute the loads. At the end of each level, the interactions, which are going to be subdivided, may be identified and distributed (migrated) over processors to keep load balanced.

Since the time needed to subdivide an interaction depends only on the time needed to compute the form factor, which in turn depends on the number of intersection tests, we can estimate the subdivision cost of a particular interaction by counting the intersection tests performed for calculating its own form factor.

Distributing interactions freely among processors require each processor to have the entire information of the scene geometry and the patches created so far. Our results show that doing this is not costly compared to the time spent for refinement. With 32 processors, a speed-up of 25 is achievable, making hierarchical radiosity practical for large scenes.

Duplicating the scene geometry on all processors may make this algorithm infeasible for complex scenes. As a future work, the algorithm can be extended to work on complex scenes by dividing the scene geometry into clusters according to some visibility factors, and then assigning every cluster to more than one processor so as to allow free interaction exchange.

Note

1. Available at http://www.cs.berkeley.edu/∼kofler.

References

1. M. Amor, E. Padron, J. Tourino, and R. Doallo. Scheduling of a hierarchical radiosity algorithm on a distributed-memory multiprocessor. In *Proc. of 4th International Meeting on Vector and Parallel Processing (VecPar'00)*, pp. 581–591, 2000.
2. C. Aykanat, T. Çapın, and B. Özgüç. A parallel progressive radiosity algorithm based on patch data circulation. *Computers & Graphics*, 20(2): 307–324, 1996.
3. J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324(4): 446–449, 1986.
4. C.-A. Bohn and R. Garmann. A parallel approach to hierarchical radiosity. In: V. Skala, ed., *Proc. of the Winter School of Computer Graphics and CAD Systems'95*. Plzen, Czech Republic, pp. 26–35, 1995.
5. G. Burns, R. Daoud, and J. Vaigl. LAM: An open cluster environment for MPI. In *Proc. of Supercomputing Symposium*, pp. 379–386, 1994.
6. M. Caballer, D. Guerrero, V. Hernandez, and J. Roman. A parallel rendering algorithm based on hierarchical radiosity. In *Lecture Notes in Computer Science*, vol. 2565, pp. 523–536, 2003.

7. M. Cohen, S. Chen, J. Wallace, and D. Greenberg. A progressive refinement approach to fast radiosity image generation. In *ACM Computer Graphics (Proc. of SIGGRAPH'88)*, vol. 22, pp. 75–84, 1988.
8. M. Cohen and J. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston, MA, 1993.
9. C. Feng and S. Yang. A parallel hierarchical radiosity algorithm for complex scenes. In: J. Painter, G. Stoll, and Kwan-Liu Ma, eds., *IEEE Parallel Rendering Symposium*, pp. 71–78, 1997, ISBN 1-58113-010-4.
10. T. Funkhouser. Coarse-grained parallelism for hierarchical radiosity using group iterative methods. In *ACM Computer Graphics (Proc. of SIGGRAPH'96)*, pp. 343–352, 1996.
11. R. Garmann. On the partitionability of hierarchical radiosity. In *Proc. of the 1999 IEEE Symp. on Parallel Visualization and Graphics*, pp. 69–78, 1999.
12. S. Gibson and R. Hubbold. A perceptually-driven parallel algorithm for efficient radiosity simulation. *IEEE Trans. on Visualisation and Computer Graphics*, 6(3):220–235, 2000.
13. C. Goral, K. Torrance, D. Greenberg, and B. Battaile. Modelling the interaction of light between diffuse surfaces. In *ACM Computer Graphics (Proc. of SIGGRAPH'84)*, vol. 18, pp. 213–222, 1984.
14. P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. *ACM Computer Graphics (Proc. of SIGGRAPH'91)*, 25(4):197–206, 1991.
15. P. Heckbert. Finite element methods for radiosity. In *ACM SIGGRAPH'92 Course Notes No. 18—Global Illumination*, Chapt. 1, pp. 1–11, 1992.
16. P. Heckbert and J. Winget. Finite element methods for global illumination. Technical Report CSD-91-643, University of California, Berkeley, 1991.
17. J. Kajiya. The rendering equation. In *ACM Computer Graphics (Proc. of SIGGRAPH'86)*, pp. 143–150, 1986.
18. D. Meneveaux and K. Bouatouch. Synchronisation and load balancing for parallel hierarchical radiosity of complex scenes on a heterogeneous computer network. *Computer Graphics Forum*, 18(4):201–212, 1999.
19. L. Renambot, B. Arnaldi, T. Priol, and X. Pueyo. Towards efficient parallel radiosity for DSM-based parallel computers using virtual interfaces. In *IEEE Parallel Rendering Symposium*, pp. 79–86, 1997.
20. F. Sillion and J.-M. Hasenfratz. Efficient parallel refinement for hierarchical radiosity on a DSM computer. In *Proc. of the Third Eurographics Workshop on Parallel Graphics & Visualisation*. Universitat de Girona, Spain, 2000.
21. R. Sireli and A. Gürsoy. Parallel hierarchical radiosity. In H.R. Arabnia, ed., *Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'99*, vol. III, pp. 1634–1640, 1999.
22. J. Wallace, K. Elmquist, and E. Haines. A ray tracing algorithm for progressive radiosity. In J. Lane, ed., *ACM Computer Graphics (Proc. of SIGGRAPH'89)*, vol. 23, pp. 315–324, 1989.