Parallel Progressive Radiosity with Adaptive Meshing

Yizhou Yu Oscar H. Ibarra Tao Yang
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

Progressive radiosity is widely used for realistic image synthesis in computer graphics applications. High-quality image generation usually requires radiosity with adaptive patch refinement to account for global illumination effects from irregular objects in a 3D space. Parallelizing such an algorithm with adaptive refinement is difficult since computation cost for each object varies from one iteration to another depending on the location of dynamically selected shooting patches. Dynamic load balancing is required but its overhead is high for distributed memory systems. This paper presents an efficient parallel algorithm for progressive radiosity, which adopts a static processor assignment strategy to take advantages of hierarchical computation structure in this problem, minimize communication and balance dynamic load. Our experiments on a Meiko CS-2 distributed memory machine show that this algorithm has achieved good performance for the tested cases.

Keywords: Parallel Progressive Radiosity, Adaptive Meshing, Load balancing, Octree, Processor assignment of irregular objects.

1 Introduction

The main application of realistic image synthesis is to create simulated scenes of photo realistic quality. Ray tracing or radiosity algorithms are widely used for computing global illumination effects in image synthesis. Recently radiosity method becomes increasingly popular for determining global illumination in architectural modeling and virtual reality [SD+93, KPC93] because it accurately portrays illumination effects such as shadows and interreflections in diffuse environments. Furthermore, once a radiosity solution is calculated for a fixed geometry, a user can interactively walk through this model to view 3D objects using virtual reality techniques on a graphics workstation.

An important application requirement is to provide real-time interaction when a user modifies a high-quality synthesized image in a computer aided design process. At the present time, both ray tracing and radiosity can be used for such cases but they are too time-consuming to accomplish real-time interaction. For example, the synthesis of an image with an average number of objects may take a few hours on a Sparc 10 workstation. A compromise solution to reduce the radiosity time is to use progressive illumination calculation [CC+88, LTG92]. The basic idea is to provide low-quality images at the beginning, then the algorithm conducts a sequence of refinement iterations to improve the image quality. The intermediate result after a few iterations can be used for displaying and a user can revise the design based on this intermediate solution without waiting for the entire computation to end. A further improvement to increase the image quality of radiosity in such an iterative algorithm is to incorporate adaptive subdivision [CG+86, PJ94].

It is important to speedup the radiosity time since it still takes a few hours to complete all iterations for progressive radiosity and even if a user wants to view intermediate results, it takes minutes to see the obvious improvement from one view to another. There have been several parallel algorithms proposed in [BW90, BP94, PC94, RGG90] but speedups for progressive radiosity with adaptive meshing are still low because the computation associated with patches in a 3D space is unstructured and the cost varies during iterations. Load balancing is the main challenge in achieving high speedups [PC94]. The previous research [BP94, PC94] improves parallel performance by concurrentizing several iterations on different processors. This is called to exploit control parallelism. In this scheme, synchronization is needed when several iterations modify the same patch object and communication overhead is high on distributed memory machines.

In this paper we propose a parallel progressive radiosity algorithm which achieves a good load balance with minimal communication overhead. Our approach is to identify the hierarchical computational structure in this problem to exploit data parallelism instead of control parallelism. We use a static assignment for all patches to minimize communication overhead. The processor assignment for irregular patch objects uses a novel object ordering with cyclic mapping to achieve load balance even if computation associated with patch objects changes. Our experiments show that the algorithm works efficiently on multi-processors and significantly reduces the illumination calculation time.

This paper is organized as follows. Section 2 gives an overview of the sequential algorithm. Section 3 discusses issues in designing parallel radiosity algorithms and related work. Section 4 gives an overview of our algorithm. Section 5 presents the processor assignment strategy of this algorithm. Section 6 gives the experimental results on a Meiko CS-2 distributed memory machine.

2 Progressive Radiosity with Adaptive Meshing

The radiosity method assumes that a 3D space contains a set of surfaces, which are perfectly diffuse, i.e. they reflect light with equal radiance in all directions. Each surface is further subdivided into a set of patches and this process is called meshing. Radiosity is assumed to be constant on each patch and we need to derive this value in order to generate a realistic image.

Let us recall the well-known radiosity equation, valid for each sample wavelength:

$$B_i = E_i + \rho_i \sum_{j=1}^{N} F_{ji} B_j. \tag{1}$$

The constants and unknown variables are:

- N is the number of patches after meshing.
- B_i is the emittance of patch i (radiosity) to be solved in the radiosity process.
- E_i is the given self-emitted radiosity of patch i. If $E_i > 0$, patch i is usually called a light source.
- ρ_i is the given reflectivity of patch i.

- F_{ji} is the constant form-factor which is fraction of energy leaving patch i that arrives at patch j. The computation of this form-factor is discussed in Section 2.3

This system of simultaneous equations represents the energy interchange via multiple interreflections and emissions in the environment. The solution of this system is the patch radiosities which provide a discrete representation of the diffuse shading of the scene. This solution is independent of the view direction. Once the system of equations has been solved, each vertex radiosity of a patch is evaluated by averaging the radiosities of the patches sharing it. The image of the scene is then computed by applying Gouraud shading or ray casting.

2.1 Progressive radiosity

We first discuss the motivation of progressive radiosity as follows. The computational cost for solving N equations in (1) is high. N^2 form-factors $\{F_{ji}\}$ must be computed from every patch to every other patch as discussed in Section 2.3 and the complexity is more than $O(N^2)$. The direct method for this linear system using Gaussian elimination takes $O(N^3)$. Thus it is time-consuming to generate a realistic image with a large number of patches. Since the radiosity matrix is diagonally dominating, the Gauss-Seidel method [CG+86] has been used to find an approximate solution, which converges in less than N iterations in practice. However the form-factor calculation is still expensive, which slows down the entire process since the partial result cannot be displayed until the form-factor calculation has completed.

This difficulty was addressed using the progressive radiosity approach[CC+88]. This method still uses a sequence of numerical iterative refinements but only N form-factors are needed to conduct the refinement at each iteration. Thus the complexity of the form-factor calculation is distributed among those iterations and the partial result can be displayed sooner than the Gauss-Seidel based method. Also some of the form-factor computation is avoided because dark patches with low shooting energy do not contribute light energy significantly in the 3D space and the form-factor computation related to these patches can be neglected.

This progressive method is shown in Figure 1 where N is the number of patches and symbol $\stackrel{s}{=}$ indicates that the operation is applied on vectors. At each iteration, the patch with maximum unshot light flux is chosen to shoot its energy. All patches will receive some part of this flux and update their radiosity values. After each iteration the accuracy of the radiosity for each patch increases, but the change is usually small. In practice, the partial result is usually displayed after several iterations. Note that only a column of the system matrix (N form-factors) is calculated, which also reduces the memory storage requirement.

2.2 Adaptive meshing

In the first paragraph of Section 2 we have assumed that each patch has a constant radiosity. That is actually not true for each patch with a large area. To achieve a good accuracy, adaptive meshing was integrated with radiosity in [CG+86, HSA91, LTG92, LTG93, PJ94, YP95]. In this approach, each patch derived from the initial subdivision (meshing) might be further refined during the course of the computation if the accuracy of each patch radiosity does not meet the standard. We call the patch derived in the initial meshing as a *superpatch* to distinguish other patches derived from the adaptive subdivision of the superpatch during each iteration.

real $F_i[N]$; /* *column of form-factors* */
typedef real spectrum$[Nb - wavelengths]$; /* *RGB tristimuli are usually used* */
spectrum ΔRad;
spectrum $B[N]$; /* *array of radiosities* */
spectrum $\Delta B[N]$; /* *array of delta radiosities* */
spectrum SpectrumNul$\stackrel{s}{=}$0.0; /* *spectrum equaling 0.0 for each sample wavelength* */
for all patches **do** /* *Initialization:delta radiosity= self-emittance* */
  $\Delta B[i] \stackrel{s}{=} E_i$;
  $B[i] \stackrel{s}{=} E_i$;
/* *iterative resolution process* */
while no convergence() **do** /* *emission loop* */
  i=patch-of-max-flux();
  compute form-factors $F_i[j]$; /* *form-factor loop* */
  for all patches j **do**{ /* *update loop* */
    $\Delta Rad \stackrel{s}{=} \rho_j \Delta B[i] \stackrel{s}{\times} F_i[j] A_j / A_i$;
    $\Delta B[j] \stackrel{s}{=} \Delta B[j] \stackrel{s}{+} \Delta Rad$;
    $B[j] \stackrel{s}{=} B[j] \stackrel{s}{+} \Delta Rad$;
  } /* *end of update loop* */
  $\Delta B[i] \stackrel{s}{=}$SpectrumNul;
} /* *end of emission loop* */

Figure 1: The progressive radiosity method.

There are various criteria for such adaptive meshing. We use the following method. We first select a few sample points in a patch. Because image rendering uses the Gouraud shading method based on linear interpolation, we compare the radiosity values at the selected sample points derived using linear interpolation with values derived using direct shooting. If the difference is great than a given threshold, then subdivision is needed. This subdivision can be applied recursively. This method performs well on parametric domain of curved or planar surfaces, and we discuss its detail as follows:

- 1) We first subdivide each surface into a triangular mesh. Each patch produced in this initial meshing is a *superpatch*. Each superpatch may be further adaptively subdivided into smaller patches during iterations using the method described in 2).

- 2) In each iteration, for each triangular patch in the current mesh as shown in Figure 2 , we compute the radiosity received from the current shooting patch at four points A, B, C and D which are centers of four smaller triangles obtained by connecting the midpoints of the edges of the larger one.

  Then we obtain the linearly interpolated radiosity at D from the radiosities at A, B and C, and compare it with the previously computed radiosity at D. If the difference is larger than a threshold, subdivide this triangle into smaller ones. Otherwise, test linearity at some sample points on AB, BC and CA. If it is nonlinear for any of them, subdivide this triangular patch. We apply the above process recursively on each newly generated superpatches.

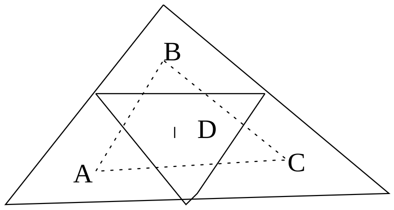


Figure 2: Testing and subdividing within a triangular patch.

2.3 Form-factor calculation

We discuss how each form-factor is computed for the above sequential algorithm. Recall the form-factor expression:

$$F_{ij} = \frac{1}{\pi A_i} \int_{A_i} \int_{A_j} \frac{cos\theta_i cos\theta_j}{r^2} VIS(P_i, P_j) dA_i dA_j \tag{2}$$

where P_i is a point on patch i, and A_i is the surface area of this patch. The term $VIS(P_i, P_j)$ expresses the visibility between a point of patch i and a point of patch j. θ_i is the angle between the normal of patch i at P_i and $\overrightarrow{P_iP_j}$, and θ_j is the angle between the normal of patch j at P_j and $\overrightarrow{P_jP_i}$. r is the distance between P_i and P_j. The cosine functions and visibility testing, $VIS(P_i, P_j)$, in (2) are very expensive to evaluate. That is why form-factor calculation is the main concern in radiosity algorithms.

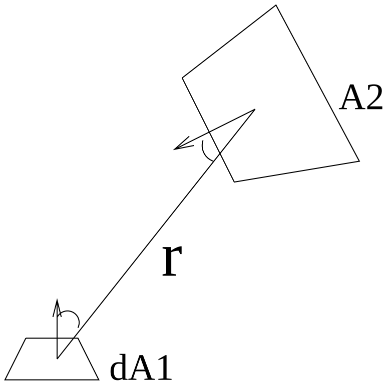


Figure 3: Form-factor from an area to a differential area.

Given area A_1 in the receiving patch and A_2 in the shooting patch, as shown in Figure 3, [WEH89] proposed a disk formula for approximating the form-factor from an area A_2 to a differential area dA_1:

$$dF_{dA_1 - A_2} = dA_1 \frac{cos\theta_1 cos\theta_2}{\pi r^2 + A_2}. \tag{3}$$

The accuracy can be improved by ray tracing more sample points on A_2. The resulting formula for radiosity at dA_1 due to illumination by A_2 is

$$B_1 = \rho_1 B_2 A_2 \frac{1}{l} \sum_{i=1}^{l} \delta_i \frac{cos\theta_{1i} cos\theta_{2i}}{\pi r_i^2 + A_2/l} \tag{4}$$

where

$$l = \text{ the number of the sample points on } A_2,$$

and

$$\delta_i = \begin{cases} 1 & \text{if the sample point is visible to } \quad dA_1; \\ 0 & \text{if occluded.} \end{cases}$$

We adopt this formula because it is desirable to obtain the radiosity at some sample point with a known normal on a curved patch rather than the radiosity received by the whole patch. In our algorithm, the radiosity of each triangular patch is approximated by the radiosity at its center. Thus, in the progressive radiosity algorithm, the first statement inside the innermost for loop should be modified to: ΔRad= the radiosity received by the center of patch j from patch i obtained by using (4).

3 Issues in Parallel Radiosity

The most important problem which arises when parallelizing radiosity is the data access. Data access determines the amount of communication involved, the type of parallelism that can be used, scalability and the necessity of static or dynamic load balancing. Another important point to be accounted for is the level at which the parallelism must be exploited in a parallel algorithm. Indeed, a coarse grain partitioning reduces the cost of managing the parallelism, but it generates less parallelism than a fine grain one. Moreover, data locality has to be exploited to keep the amount of communication between processors as low as possible. In the radiosity algorithm, choosing a coarse grain parallelism would not allow exploitation of data locality. Therefore, a tradeoff between selecting a grain size and exploiting the data locality has to be decided.

Several parallel radiosity algorithms have been proposed in the literature, e.g. [BW90, BP94, PC94, RGG90]. Baum et. al.[BW90] have obtained interesting results on a shared memory machine with 8 processors supplied with a hardware Z buffer; however, scalability results on a larger of numbers of processors have not been demonstrated. In this paper, we focus on parallel algorithms on distributed memory machines. In [RGG90], a master-slave approach is proposed. At each computation stage, each processor obtains a part of the work to process. Since patch distribution is not fixed, data re-distribution is needed. To avoid excessive communication cost for data-redistribution, they replicate all geometrical data to all processors. But still the master processor needs to collect a new update from each slave processor, selects the shooting patch and broadcast related new updates to all slave processors at each iteration. Communication overhead and memory requirement are high.

Another approach is to evenly distribute data among all nodes [PC94, BP94]. The question is how to exploit the parallelism. Since data structure in the 3D space is irregular, control-oriented parallelization is used in [PC94, BP94]. In this scheme, several iterations, corresponding the while loop in Figure 1, can be started concurrently in different processors. Notice that the parallel algorithm result is slightly different from the sequential algorithm since unsynchronized radiosity updating is used. Since a processor may need non-local data, managing communications between processors is important [PC94]. Also since concurrent iterations may update the radiosity of the same patch, these updates must be synchronized. The implementation in [BP94] relies on the shared virtual memory support support, which leads to excessive operating system level overhead in maintaining data consistency. The other problems which are not well addressed are the load balancing and adaptive meshing for high quality image synthesis. If adaptive meshing is considered, the cost of communication and keeping data coherent will be higher because the mesh refinement results also need to be communicated.

Our approach is to follow the sequential algorithm semantics, exploit irregular data parallelism instead of control parallelism and provide a carefully designed processor assignment strategy to map patches for load balancing. We only allow the owner processor of a patch modifying its radiosity value to avoid unnecessary synchronization overhead.

Adaptive meshing required at each iteration of radiosity creates an additional difficulty in designing an efficient parallel solution. This is because adaptive refinement of patches may increase the number of patches on some processors dynamically, consequently it worsens the load balance.

It is indicated in [BP94] that the form-factor calculation and the radiosity update require an access to the information regarding *all* patches. This is due to the use of hemicube-based algorithms. Actually, hemicube is seldomly used in recent sequential algorithms because the hemicube method has aliasing effects. It is not necessary to access all patches. One good method for the form-factor calculation is discussed in Section 2.3.

4 A Parallel Solution for Progressive Radiosity with Adaptive Meshing

Our approach is to parallelize the *for* loop in the algorithm of Figure 1. We distribute the superpatches evenly among processors and the processor mapping method is discussed in Section 5. After selecting a global shooting patch, each processor only has to work on those patches owned by that processor. The calculation for each patch needs all surface information but does not need knowledge about patches on other processors.

4.1 Initialization and data distribution

At the beginning, the albedo and geometrical information of all surfaces making up the objects in a scene are replicated to all processors. After each processor computes the bounding box for each surface, it sets up an octree-based space partition which will be used in the data distribution. The octree partition can accelerate ray casting in form-factor calculation which will be elaborated later. Then each processor subdivides each surface into a number of large triangular superpatches of approximately the same size and orders all superpatches used for the processor assignment. Notice that the result of this partitioning process is used for the rest of the iterations, and it takes less 1 second. There is no need to parallelize this process.

Octrees are the 3D counterpart of quadtrees. The octree-based space partition initially has only one node representing a large cube containing all the surfaces. Then we check the number of surfaces in the current tree node. If it is more than a threshold, the current node is subdivided into eight smaller nodes and all surfaces intersecting the current node are distributed into its child nodes. This process is recursively repeated for each of its children.

In the superpatch distribution stage, the algorithm distributes superpatches cyclically on the given p processors based on their ordering numbers. This is done by letting each processor scan all superpatches in all surfaces, preserve those superpatches distributed to itself and deleting all the others. We discuss the superpatch ordering and distribution strategy in Section 5.

Now we have two hierarchies of data. The upper level contains the octree partitioning and geometrical information about all surfaces. The lower level contains radiosity and geometrical information on all patches. After data distribution, each processor only owns a subset of superpatches and it keeps a copy of upper level information to reduce communication in form-factor calculation. Duplicating the data in upper level has little impact on memory requirement. This is because usually we have several hundred of surfaces in a scene but the number of patches after the initial and adaptive meshing is very large.

4.2 Iterative radiosity computation

At the beginning of each iteration, each processor picks up the superpatch with the maximum unshot light flux in its own dataset. Then a global MAX reduction operation which costs about $O(\log p)$ is conducted to select the superpatch with the global maximum unshot light flux. Then each processor only works on its own patches to receive radiosity from the shooting patch. It may subdivide a patch into smaller ones by using the adaptive meshing technique discussed in Section 2.2. Notice that a shooting patch is always a superpatch. At the end of each iteration, a barrier is called to synchronize the processors and start the next iteration.

We discuss a hierarchical strategy in reducing the computation cost associated with radiosity calculation in a patch subdivision using adaptive meshing. By observation, we find that the lightness of objects in a scene is contributed mainly by a small number of light sources. After they finish shooting, the changes of lightness in this environment are not substantial. On the other hand, the existence of these light sources causes the adaptive subdivision of other patches. But this adaptive meshing is only useful when these sources possess a large amount of unshot flux. After shooting from bright light sources completes, the adaptive subdivision may not be necessary because the dark light sources do not lead to significant changes of radiosity values in destination patches. Thus we maintain the hierarchical structure of subdivision. When a shooting patch is selected, we start to compute the impact of this shooting patch on radiosities of a set of patches represented hierarchically in a top-down fashion. We first start from the superpatch and apply the procedure to its child nodes recursively. When the computation at an internal node x shows that the radiosity distribution for subtree nodes rooted at x is smooth enough, the computation can stop at this point.

This hierarchical computing strategy can eliminate the amount of computation substantially because many iterations involve shooting patches with small unshot energies. It also improves the load balance because given a set of superpatches after initial meshing, uneven run-time subdivision of each superpatch is the major reason causing load imbalance. The hierarchical computing method eliminates unnecessary computation in a superpatch with a large subdivision tree, prevents unbalanced load from propagating to subsequent iterations, and thus smoothes cost difference between different patch subdivision trees. We will show the effectiveness of this hierarchical strategy in our experiments.

Now we show some quantitative analysis of this technique. Suppose we want to achieve a global relative error ϵ on each patch, and the radiosity contributed by those bright light sources is RAD, the radiosity contributed by all other patches is αRAD where α is usually less than 1. Instead of using the same error tolerance ϵ in numerical computation for each shooting patch, we use the error tolerance $\frac{\epsilon}{2}$ for those light sources and $\frac{\epsilon}{2\alpha}$ for the other shooting patches. Now the total accumulated error is

$$\frac{\epsilon}{2}RAD + \frac{\epsilon}{2\alpha}\alpha RAD = \epsilon RAD\ .$$

The total accumulated radiosity is

$$(1 + \alpha)RAD \ .$$

So the global relative error is

$$\frac{\epsilon RAD}{(1 + \alpha)RAD} = \frac{\epsilon}{1 + \alpha} \leq \epsilon \ .$$

Thus, we can achieve the same global error tolerance by using a smaller one for light sources and a much larger one for others. For example, if $\alpha = 0.1$, $\frac{\epsilon}{2\alpha}$ would be 5ϵ. Since there are only a small number of light sources which contribute most of the radiosity, the computation cost for other patches dominates the total cost. By using a larger error tolerance for those, we can make computation more efficient without losing necessary accuracy. A larger error tolerance also means smaller subdivision trees and smaller difference between different subdivision trees, which smoothes the computation and makes it more balanced.

4.3 Form-factor calculation

In [BP94], the parallel algorithm needs to access information regarding all patches. That is actually not necessary. We only need the radiosity and geometrical information of the shooting patch, the albedo and geometrical information of the receiving patches and occluding information implied in the octree partitioning. Thus each processor only accesses local patch information and the replicated surface data. No interprocessor communication is needed during the form-factor calculation.

For the method discussed in Section 2.3, occlusion test only uses surface geometry. It is done by casting rays from the current receiving patch to some sample points on the shooting patch.

This process can be accelerated by letting the rays go through some leaf nodes in the octree between the receiving patch and the shooting patch[MB90]. In this way, occlusion test is done only against those surfaces intersecting these leaf nodes.

Since a shooting patch is chosen from the initial mesh, it may contain many tiny patches. To accelerate form-factor calculation, we can fit a coarser mesh for the shooting patch by combining some of the tiny patches. And a larger patch takes the average radiosity of the smaller ones.

4.4 Rendering and display

The image rendering and display usually cost a few seconds in a workstation, which is acceptable with most of computer aided design applications. There is some research work on parallel rendering [MC+94, OHA93, W94] for larger data sets, which we have not considered in this paper.

5 Processor Assignment for Superpatches

Our goal is to distribute the amount of dynamic computation evenly among all processors. Since the amount of computation differs for different patches, it is not sufficient to only make each processor have an equal number of patches. We have not seen any publications on optimal static data distribution for parallel radiosity. We have not seen any publications using heuristics to do processor assignment for this problem, either.

The problem of partitioning irregular data space has been studied in other scientific application areas, for example n-body simulations [SHG95]. Usually the quadtree partitioning method is used and neighboring leafs are directly mapped to the same processor. This is because neighboring particles have intensive communications. We call this method the quadtree-based block mapping. This method can be generalized to a 3D space. We call it octree based block mapping. The octree based block mapping may not be effective for our problem because there is no significant difference in terms of communication volume between neighboring superpatches and non-neighboring superpatches. Also each superpatch may be subdivided dynamically, which leads to uneven patch distribution.

Dynamic load balancing that migrates patches between processors looks reasonable. But in practice it suffers high communication overhead on a distributed memory machine. We will use a static assignment for all superpatches to avoid patch migration overhead: each superpatch and its possible child patches generated during adaptive meshing are assigned to the same processor. We design a mapping strategy which considers the possible increase of computation associated with each superpatch. The *key idea* is to identify superpatches with similar cost even after adaptive meshing and distribute these superpatches evenly among different processors.

We study the computation requirement associated with each super or non-super patch. At an iteration, each patch receives radiosity from the shooting patch, and most of the computation is spent on the numerical integration for form-factor calculation (Equations (2) and (4)), whose complexity depends on many factors, such as the distance between the shooting and receiving patches, the relative orientation of the patches, and the area and unshot flux of the shooting patch. The following are some facts that we need to exploit in the design of processor assignment to achieve load balance.

- **F1:** Adjacent patches on the same surface have similar orientation because a surface is usually represented by a continuous mathematical function which has continuous normals;
- **F2:** Patches close to each other have approximately the same distance to the shooting patch;
- **F3:** Most subdivision happens at shadow boundaries. If a patch contains a shadow boundary, another patch nearby is also likely to contain a shadow boundary since shadow boundaries are continuous curves. So both these patches are likely to be subdivided into many smaller patches.

Based on these facts, in addition to make processors have the approximately equal number of superpatches, we propose the following strategies for processor mapping.

- **S1:** Adjacent superpatches on the same surface should be distributed to different processors based on F1 and F2. This will not incur additional communication because adjacent patches have no extra dependence on each other.
- **S2:** Superpatches in different surfaces close to each other should be distributed to different processors based on F2. This heuristic will be applied after S1 because S2 has less constraints.

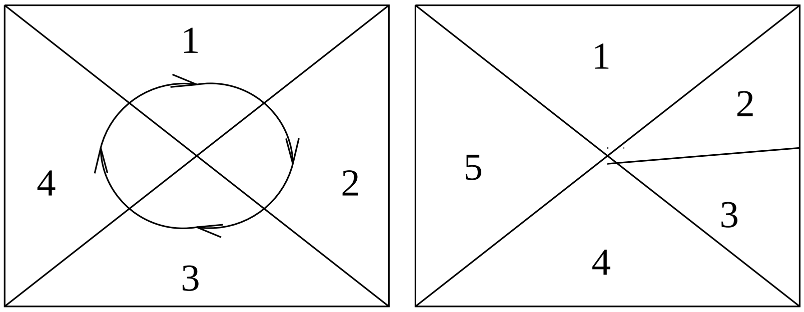
To achieve the goal of S1 and S2, we derive a linear order for all superpatches, and then cyclically distribute superpatches to processors based on this order. Thus the superpatch ordering plays an important role in achieving the load balance using S1 and S2.

Since all surfaces are distributed irregularly in a 3D space, there is no obvious ordering between them. We first introduce such an ordering for superpatches on the same surface, then for those on different surfaces.

5.1 Ordering of superpatches within the same surface

Each surface is assumed to be bivariate and its parametric domain is a square. A triangular superpatch is always subdivided into two smaller ones if the length of its longest edge is above a threshold during each step of the initial meshing. If another superpatch shares with it the edge where the subdivision happens, that superpatch is also divided into two at the same edge. The ordering of superpatches within a surface is defined recursively during initial meshing using the following three rules:

1. Initially a surface is subdivided into four superpatches. Their ordering is clockwise as depicted in Figure 4(a).
2. If a superpatch is subdivided, the two newly generated superpatches should be inserted into the ordering at the position occupied by the original superpatch. So if superpatch 2 in Figure 4(a) is subdivided, the ordering is shown in Figure 4(b).
3. Given the ordering for the current mesh, there exists a route traversing all superpatches following the ordering. As the directed cycle shown in Figure 4(a), each superpatch has an entry and an exit edge. An entry edge is the one where the route enters the superpatch. An exit edge is the one where the route leaves the superpatch. If a superpatch in the current mesh is subdivided, the relative order of the two new superpatches is defined as below,
   (a) If the subdivision happens at the entry edge, the new ordering is shown in Figure 5(a);
   (b) If the subdivision happens at the exit edge, the new ordering is shown in Figure 5(b);
   (c) If the subdivision happens at the third edge, the new ordering is shown in Figure 5(c).



(a) (b)

Figure 4: Ordering after subdivision.

Example:. The ordering for a surface using the above rule is shown in Figure 5(d).

Theorem 5.1.1 *A closed continuous route traversing all superpatches following the above ordering always exists after each step of the subdivision.*

Proof by induction. The route for the *basis* is shown in Figure 4(a).

Suppose at k-th step, a patch A is subdivided and the subdivision happens at its entry edge with respect to the current route. There must exist another patch B whose exit edge coincidents with A's entry edge.

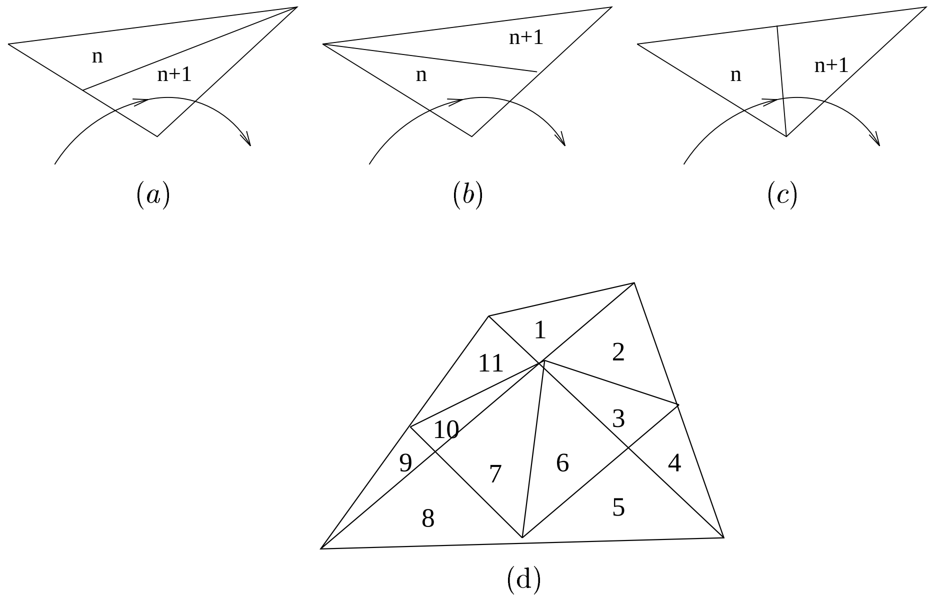


Figure 5: Ordering of superpatches.

So both these two patches should be subdivided, and there are four new smaller patches. Let the two new patches generated from A be A_1 and A_2, the two generated from B be B_1 and B_2. The ordering of A_1 and A_2 should observe Figure 5(a); The ordering of B_1 and B_2 should observe Figure 5(b). A new route can be set up. Only the part of the old route between A and B should be changed. B's entry edge becomes B_1's entry edge. The common edge between B_1 and B_2 becomes B_1's exit edge and B_2's entry edge. B_2's exit edge is a part of B's exit edge. A's exit edge becomes A_2's exit edge. The common edge between A_1 and A_2 becomes A_1's exit edge and A_2's entry edge. A_1's entry edge is a part of A's entry edge.

We can give similar proof for the cases where the subdivision happens at A's exit edge or the third edge.

We can see that two consecutive superpatches in the ordering must be adjacent to each other, which means they have a common edge. This is important because it is most likely that two adjacent superpatches share the same properties, such as there is one shadow boundary on both of them. However, since we are trying to fit a linear ordering onto a 2D or 3D space, we can not guarantee any two adjacent superpatches will be close to each other in the ordering. But this ordering scheme surely can increase this kind of possibility significantly. This recursively defined ordering has another appealing property— the superpatches within the same larger superpatch are close to each other in the ordering. This means that these superpatches in the same local region, which share the same local properties, are likely to be distributed to different processors.

5.2 Ordering of superpatches on different surfaces

Now we can define the ordering for superpatches on all surfaces using an octree partitioning defined in Section 4.3.

- Traverse the octree recursively. If a leaf node L_1 is met before L_2, all superpatches on surfaces intersecting L_1 are ordered before those on surfaces intersecting L_2. If a surface intersects more than one leaf, it is only considered the first time it is encountered.

- If more than one surfaces intersect the same leaf node, they are ordered randomly. But if one surface S_1 is put before S_2, all superpatches on S_1 are also before those on S_2 in the ordering.

Once an ordering of all superpatches is given, they can be distributed to the processors cyclically in the data distribution phase. Each processor has an approximately same number of superpatches.

5.3 Performance analysis

It is difficult to provide an analytic result on the effectiveness of load balancing for an arbitrary irregular object distribution in a 3D space and our strategy is a heuristic. We analyze the performance of our algorithm in the following two scenarios to demonstrate the effectiveness of our design strategy. We also compare our approach with the octree-based block mapping. The following parameters will be used in this analysis.

- p is the number of processors.
- T is the number of superpatches for each surface.
- n is the total number of sample points used for all numerical integrations using (4) for a receiving superpatch in one iteration.
- E is the average cost for each sample point when (4) is evaluated.
- γ is the ratio of the number of radiosity updates and patch subdivisions over the total number of sample points for numerical integrations. So $\gamma << 1$.
- C is the maximum amount of time needed for each radiosity update or patch subdivision. It is less expensive than each form-factor calculation.

Scenario 1: Loosely distributed light source-pairs. Suppose there are m pairs of object surfaces $\{S_1, S_1', S_2, S_2', \cdots, S_m, S_m'\}$ as shown in Figure 6. These surfaces are exactly the same except for their spatial positions and orientations. We assume there is no light occlusion between any two surfaces in the space. Surfaces are flat so that the patches on the same surface have no interaction with each other. The two surfaces in the same pair are placed face-to-face very close to each other.

Each surface is subdivided into the same number (T) of superpatches. We assume that $p << m$ and if the shooting superpatch and the receiving superpatch are on the same pair but different surfaces, we need at least n_2 sample points for the numerical integrations to achieve the desired accuracy. We also assume that the surface pairs are placed far away from each other so that if the shooting patch and the receiving patch are on different pairs, we need at most n_1 sample points for numerical integrations. Notice that $n_1 << n_2$.

For one iteration, let the shooting patch be on S_j. Based on Equation (4), the sequential time for form-factor calculation in one iteration is:

$$Seq_{form} = En_2T + En_1(2m-2)T.$$

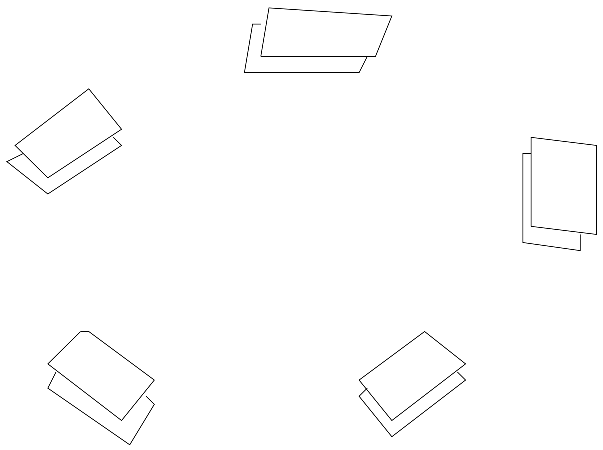


Figure 6: Surface configuration in Scenario 1 when $m = 5$.

where the first term is for those superpatches on $S_j^{'}$ and the second term is for those receiving superpatches on other surface pairs.

The sequential time for radiosity updates and patch subdivisions at each iteration is about

$$Seq_{update} = 2C\gamma n_2 T + 2C\gamma n_1 T(2m-2).$$

So the total sequential time for each iteration is approximately

$$Seq = Seq_{form} + Seq_{update} = (E + 2C\gamma)n_2 T + (E + 2C\gamma)n_1 T(2m-2).$$

When our method is used, we let α be the cost of communication for each iteration. Then the parallel time for each iteration is approximately equal to:

$$(E + 2C\gamma)n_2 \frac{T}{p} + (E + 2C\gamma)n_1 \frac{(2m-2)T}{p} + \alpha$$

So the speedup of K iterations is

$$\frac{K((E + 2C\gamma)n_2 T + (E + 2C\gamma)n_1(2m-2)T)}{K((E + 2C\gamma)n_2 \frac{T}{p} + (E + 2C\gamma)n_1 \frac{(2m-2)T}{p} + \alpha)} \approx p$$

if the cost of communication (α) is very low. Our experiments show that the communication overhead is small.

If we use octree-based block mapping, then superpatches on the same light source pair are assigned to the same processor. Each processor has $\frac{m}{p}$ pairs of surfaces. At each iteration, the processor in which the shooting patch is selected will have the heaviest CPU load which is:

$$(E + 2C\gamma)n_2 T + (E + 2C\gamma)n_1(\frac{2m}{p} - 2)T.$$

Since $n_1 << n_2$ and $p << m$, the speedup is at most

$$\frac{K((E + 2C\gamma)n_2 T + (E + 2C\gamma)n_1(2m-2)T)}{K((E + 2C\gamma)n_2 T + (E + 2C\gamma)n_1(\frac{2m}{p} - 2)T + \alpha)} \leq \frac{n_2 T + n_1(2m-2)T}{n_2 T + n_1(\frac{2m}{p} - 2)T} \leq \frac{2}{3} + \frac{2}{3}p.$$

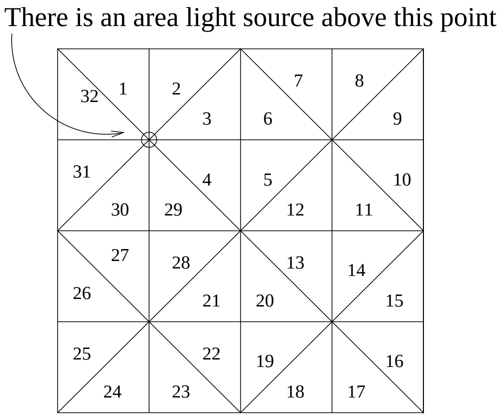


Figure 7: An example of surface configuration for Scenario 2 when $p = 4$ and $q = 2$.

Scenario 2: Single light source above a planar surface. The second case has one planar surface. There is a small area light source hanging above the center of the upper left quarter of the planar surface and the position of this light source is very close to that surface. We assume the reflectivity of the light source is close to zero, and that initial meshing divides the surface into p blocks. And each block is further divided into qp superpatches. Thus $T = qp^2$. An example with $q = 2$ and $p = 4$ is depicted in Figure 7. Notice that since there is no other surfaces, the reflected lights from the planar surface do not affect other patches.

Assume that we need n_2 sample points for superpatches directly under the light source. Denote the set of those superpatches by U. For Figure 7, U contains superpatches 1, 2, 3, 4, 29, 30, 31, and 32. Also assume that we need at most n_1 sample points for other superpatches to achieve a specified accuracy. Notice that $n_2 >> n_1$ because the distance between the light source and the patches in U is close to zero, which will make n_2 extremely large. There is a lower bound on the distance between the light source and those patches not in U, which means there is an upper bound on n_1. Thus $\frac{n_2}{n_1}$ is large.

The total sequential time for one iteration is about

$$(T - qp)(E + 2C\gamma)n_1 + qp(E + 2C\gamma)n_2.$$

When our mapping method is used, each processor has about q patches in U. The CPU time on each processor is approximately

$$(\frac{T}{p} - q)(E + 2C\gamma)n_1 + q(E + 2C\gamma)n_2$$

Again, if the cost of communication is low, we can get nearly perfect speedup for this case.

If octree-based block mapping is used, all patches in U are placed onto processor 0 and other patches are distributed to other processors. Processor 0 has the heaviest load

$$qp(E + 2C\gamma)n_2.$$

Thus the speedup for computing radiosity distribution contributed by the area light source is at most

$$\frac{(T - qp)(E + 2C\gamma)n_1 + qp(E + 2C\gamma)n_2}{qp(E + 2C\gamma)n_2} = (p - 1)\frac{n_1}{n_2} + 1 << p.$$

Through the above analysis, we can see that our method balances the load and obtains the near optimal solutions for the above scenarios. And it outperforms the octree-based block mapping.

6 Experimental Results

We implemented the parallel radiosity algorithm presented in the previous two sections on a Meiko CS-2 distributed memory machine. Each Meiko node is SUN Sparc Viking with a peak performance 40MFLOP per second and nodes are connected by a fat-tree like fast network with a peak bandwidth 40MB per second. An octree is set up on each processor for data distribution and the acceleration of ray-object intersection. We will report the speedups for each tested case and analyze the effectiveness of our processor mapping strategy for irregular objects and impact of our hierarchical calculation strategy discussed in Section 4.2.

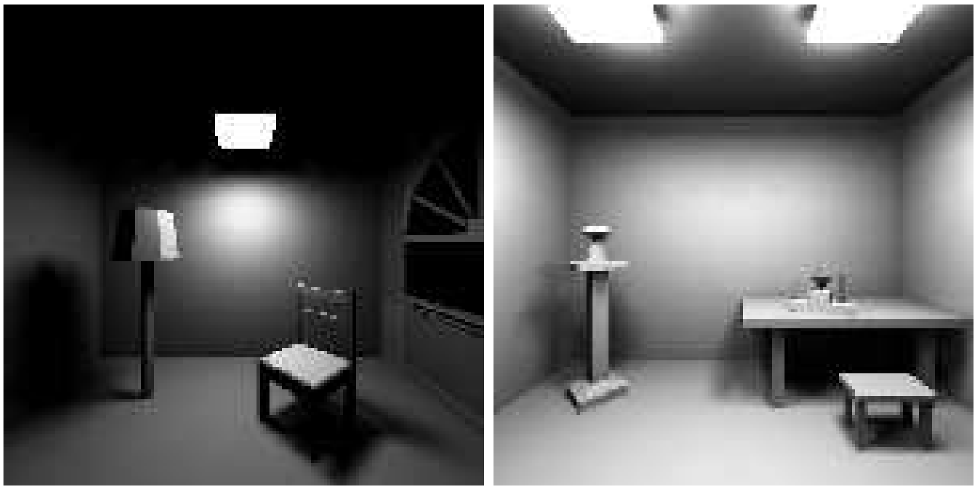


Figure 8: The left is a room with a lamp, chair and window. The right is a room with tables, stools and some other objects.

We tested our algorithm on three indoor scenes. Two scenes are shown in Figure 8. The number of surfaces in each scene is between 200 and 300. The number of superpatches after the initial meshing for each scene is around 10,000. 400 iterations were executed for each scene, which is reasonable for interactive design. The total sequential times for scenes 1, 2 and 3 are 4 hours 35 minutes, 3 hours 18 minutes and 2 hours 54 minutes, respectively.

Scalability. The parallel time speedup for each test scene is shown in Figure 9. This result is comparable to the recent results for progressive radiosity[BP94] *without* adaptive meshing, and is much better than previous results for both adaptive and non-adaptive progressive radiosity[PC94]. As we already mentioned in Section 3, some difficulties arise from adaptive meshing, the above experimental result shows that our algorithm deals with irregular and adaptive load variation very well.

Distribution of computation time, idle time and communication overhead. We will show that the load among processors is approximately the same during the radiosity computation. We measure the CPU time used for arithmetic computation for each processor and we call it workload. The workload distribution on 16 processors for 400 iterations in processing the first scene is shown in Figure 10. It can be seen that our algorithm has achieved a good load balance. The workload distribution for other scenes is similar.

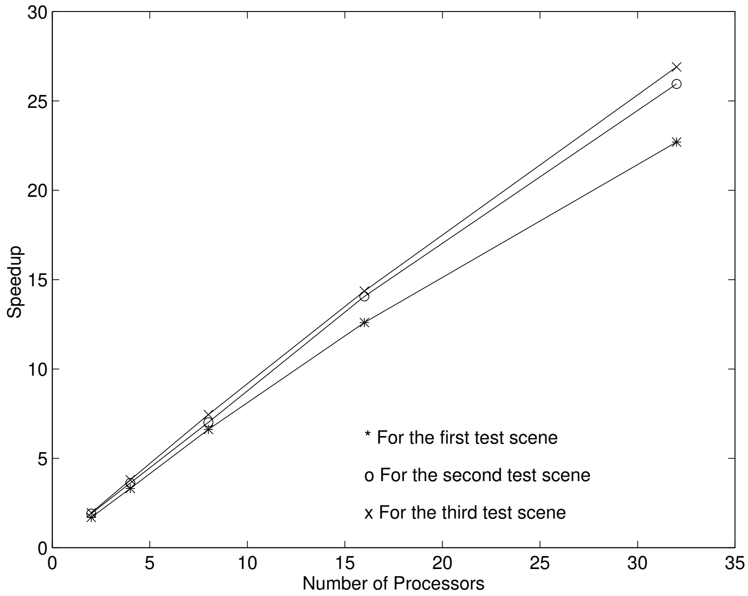


Figure 9: Speedup for three test scenes.

We have not reached the perfect speedup when $p = 32$ because we lose some performance due to communication overhead and some processor idles caused by small load imbalance. Table 1 lists the average time among all processors spent on communication and the standard deviation of total amount of workload on 32 processors. The communication cost is extremely low, used for selecting the shooting patch and broadcasting the data for the shooting patch. The load imbalance factor is within 10%. That is probably the best we can achieve for a static assignment in responding to the dynamically-changing workload. The previous work [PC94] using dynamic balancing does not have a good speedup because of high communication overhead.

Effectiveness of the hierarchical calculation strategy. We have examined the impact of our hierarchical calculation method which is discussed in Section 4.2. Table 2 shows the improvement ratio $(\frac{Speedup_{with}}{Speedup_{without}} - 1)$ when p=32. Here $Speedup_{without}$ is the speedup without using this strategy and

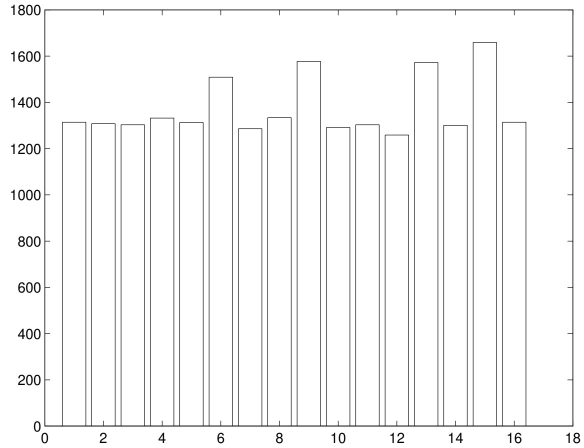


Figure 10: Workload distribution on 16 processors for the first scene.

	Scene 1	Scene 2	Scene 3
Communication	0.87%	1.21%	1.65%
SDev. of workload	9.94%	7.40%	9.78%

Table 1: Cost of communication and standard deviation of workload.

$Speedup_{with}$ is the speedup using our method. The improvement is around 20% and that is another reason we can obtain a much better performance compared to [PC94].

	Scene 1	Scene 2	Scene 3
Improvement ratio	22%	16%	24%
Speedup of NH	18.6	22.4	21.7

Table 2: Our hierarchical method vs. a non-hierarchical method (NH).

Effectiveness of the processor assignment strategy. We have also examined the impact of our processor assignment strategy. Table 3 shows the improvement ratio of our strategy over octree-based block mapping when p=32.

	Scene 1	Scene 2	Scene 3
Improvement ratio	28%	31%	36%
Speedup of OBM	16.6	17.9	17.3

Table 3: Our processor assignment strategy vs. octree-based block mapping (OBM).

7 Conclusions

We presented an efficient parallel progressive radiosity algorithm with adaptive patch refinement using a hierarchical computing and octree-based cyclic mapping scheme. This algorithm can significantly reduce the time in generating high-quality realistic images for real-time interaction. Our parallel algorithm efficiently exploits irregular and adaptive data parallelism using a carefully designed object assignment method. Our experiments show that it has low communication cost and promising scalability.

Our future work is to study the possibility of obtaining better speedup by incorporating limited dynamic load balancing. However, we believe that our static distribution should still play an important role in balancing since a fully dynamic strategy is too expensive in terms of communication overhead.

The stage of image rendering can also be parallelized to get a faster rate of interaction and currently we can make use of some of the existing parallel polygon rendering algorithms [MC+94, OHA93, W94].

References

[BW90] D.R.Baum, and J.M.Winget, "Real Time Radiosity Through Parallel Processing and Hardware Acceleration," Computer Graphics, 24(2), 1990, pp.67-75.

[BL+86] Bergman, Larry, H.Fuchs, E.Grant and S.Spach, "Image Rendering by Adaptive Refinement," Proceedings of SIGGRAPH'86, Computer Graphics, 20(4), 1986, pp.29-37.

[BP94] K.Bouatouch and T.Priol, "Data Management Scheme for Parallel Radiosity," Computer-Aided Design, Vol.26, No.12, 1994, pp.876-882.

[CG+86] M.F.Cohen, D.P.Greenberg, D.S.Immel and P.J.Brock, "An Efficient Radiosity Approach for Realistic Image Synthesis," IEEE CG&A, 6(2), 1986, pp.26–35.

[CC+88] M.F.Cohen, S.E.Chen, J.R.Wallace, D.P.Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation," Computer Graphics, 22(4), 1988, pp.75–84.

[HSA91] P.Hanrahan, D.Salzman and L.Aupperle, "A Rapid Hierarchical Radiosity Algorithm," Computer Graphics, 25(4), 1991, pp.197–206.

[KPC93] J.K.Kawai, J.S.Painter, and M.F.Cohen, " Radioptimization–Goal Based Rendering," Computer Graphics Proceedings, Annual Conference Series, 1993, pp.147-154.

[LTG92] D.Lischinski, F.Tampieri and D.P.Greenberg, "Discontinuity Meshing for Accurate Radiosity," IEEE CG&A, 12(6), 1992, pp.25–39.

[LTG93] D.Lischinski, F.Tampieri and D.P.Greenberg, "Combining Hierarchical Radiosity and Discontinuity Meshing," Computer Graphics Proceedings, Annual Conference Series, 1993, pp.199–208.

[MB90] J.D.MacDonald and K.S.Booth, "Heuristics for ray tracing using space subdivision," Visual Computer, (1990)6, pp.153–166.

[MC+94] S.Molnar, M.Cox, David Ellsworth and H.Fuchs, "A Sorting Classification of Parallel Rendering," IEEE Computer Graphics and Applications, No.4, 1994, pp.23-32.

[OHA93] F.A.Ortega, C.D.Hansen and J.P.Ahrens, "Fast Data Parallel Polygon Rendering," proceedings of Supercomputing'93, pp.709-718.

[PC94] D.Paddon and A.Chalmers, "Parallel processing of the radiosity method," Computer-Aided Design, Vol.26, No.12, 1994, pp.917-927.

[PJ94] M.Paulin and J.Jessel, "Adaptive mesh generation for progressive radiosity: A ray-tracing based algorithm," Eurographics'94, pp.C421-C432.

[RGG90] R.J.Recker, D.W.George and D.P.Greenberg, "Acceleration Techniques for Progressive Refinement Radiosity," Computer Graphics, 24(2), 1990, pp.59-66.

[SD+93] C.Schoeneman, J.Dorsey, B.Smits, J.Arvo, and D.Greenberg, " Painting with Light," Computer Graphics Proceedings, Annual Conference Series, 1993, pp.143-146.

[SHG95] J.P.Singh, J.L.Hennessy, and A. Gupta, "Implications of Hierarchical N-Body Methods for Multiprocessor Architectures," ACM Trans. on Computer Systems, Vol.13, No.2, 1995, pp.141-202.

[WEH89] J.R.Wallace, K.A.Elmquist and E.A.Haines, "A Ray Tracing Algorithm for Progressive Radiosity," Computer Graphics, 23(3), 1989, pp.315–324.

[W94] S.Whitman, "Dynamic Load Balancing for Parallel Polygon Rendering," IEEE Computer Graphics and Applications, No.4, 1994, pp.41-48.

[YP95] Y.Yu and Q.Peng, "Multiresolution B-spline Radiosity," Eurographics'95, pp.C285-C298.