Parallel Radiosity Image Generation on a Distributed Memory Machine

Katsuhiro Yamazaki, Akira Uejima, Tohru Watanabe and Hidekatsu Tokumaru
Department of Computer Science
Ritsumeikan University, Kusatsu 525–77, Japan
{yamazaki,uejima}@hpc.cs.ritsumei.ac.jp

Abstract. Although the radiosity method can generate very realistic images due to global illumination effects, it requires an enormous amount of computation to calculate the form factors. This paper describes how to parallelize the calculation of form factors in the progressive refinement method by subdividing hemicubes into multiple elements and equally allocating them to multiple processors. A static load balancing algorithm and a dynamic one are devised in order to equalize the processing time in each processor. Two sample scenes are executed using these two algorithms on a 64 transputers system connected by a torus network. The results demonstrate that the execution time is proportional to the number of hemicube subdivisions, and it decreases almost linearly as the number of processors increases provided that polygon data are copied to all processors, while patch data are distributed to each processor. This approach is very attractive in terms of space complexity, because data for patches including the form factor matrix are subdivided into multiple elements which are equally allocated to each processor. In addition, the dynamic load balancing algorithm is better than the static one when the number of hemicube subdivisions is not a multiple of the number of processors.

1. Introduction

The radiosity method models the correct interaction and object-to-object reflections between diffusely reflecting surfaces[1], thus it can generate very realistic images due to global illumination effects. In the radiosity method, polygons consisting of the surfaces of objects are divided into small elements, and then the intensity is computed for each element by considering incident and reflected rays. The hemicube method[2] was devised in order to numerically calculate the approximate value of the form factor integral. Here, form factors represent the fraction of energy leaving a patch which arrives at other patches. Computing form factors requires an enormous cost in terms of time and space complexities.

Meanwhile Cohen et al. proposed the progressive refinement approach[3] the goal of which is to display the results after each step of an iterative process, in order to provide immediate feedback[4]. The ability to view partial solutions allows the user to evaluate the design without waiting for the solution to be completed[5]. In addition, only one column of the form factor matrix is required for one shooting step, and thus the memory requirements for the form factor matrix are greatly reduced.

This paper describes how to parallelize the calculation of the form factors in radiosity image generation using static/dynamic load balancing algorithms on distributed memory parallel machines. In our approach as well as in the work by Baum et al.[5], hemicubes are subdivided into small elements so that they are computed by multiple processors.

In terms of data distribution, polygon data are copied to all processors, while patch data are distributed to each processor. Here, polygons are elements of an object and include coordinates. If a polygon is a rectangle, it is subdivided both vertically and horizontally at even intervals. The subdivided elements are called patches, which are units for calculating form factors. Although this approach is inadvantageous to memory requirements, the approach is advantageous to speed, because it is unnecessary to transfer coordinates among processors.

The processing time of each processor, however, may be different due to the imbalance in the placement of polygons if small elements of hemicubes are equally allocated to each processor. Thus, small elements of hemicubes should be allocated to each processor so that the processing time of each processor is balanced. We propose not only a static load balancing algorithm which equally allocates these elements to each processor, but a dynamic one which allocates these elements in run time in order to equalize loads in each processor. Two sample scenes, 192 polygons and 372 polygons, are executed on a 64-transputers system with a torus network using static/dynamic load balancing. The experimental results are discussed in terms of time complexity, space complexity and load balancing algorithms.

Section 2 explains the outline of the multi-transputer system. Section 3 describes the radiosity algorithm, particularly calculations of form factors using the hemicube method in detail. Section 4 explains the static and dynamic load balancing algorithms by subdividing the hemicubes into multiple elements. Section 5 shows the experimental results and discusses them in terms of time and space complexities. Section 6 compares our approach with related work. Finally, Section 7 presents the conclusions.

2. A Multi-Transputer System

A total of 64 transputers, T805, with 1MB main memory, the clock rate of which is 25MHz, is used in this experiment. Four INMOS serial links the communication rate of which is 10Mbps are used to communicate with the external processors. Four kinds of torus, that is 2×2, 4×4, 4×8, and 8×8, as well as 2^5 hypercube are provided in the interconnection networks. In order to construct a 2^5 hypercube, two transputers are used as a couple which offers six links[11]. This research uses the torus network, because it has the least communication overheads. Apart from these 64 transputers, a root transputer is provided on a host machine. Parallel C, the functions of which are extended for communication between the processors and multi-threads, is used in this research. Here, threads are light-weight processes which can be concurrently executed by sharing data and communicating with others using channels. Computation and communication are overlapped by using multi-threads [11].

3. The Radiosity Algorithm

3.1. The Computation Procedure

The procedure for developing the radiosity image generation is as follows[4].

1. Objects are defined by polygons which are divided into patches, discrete surface areas.

2. Assuming that the light energy is equilibrated, the following equation is given by considering the incidence and diffusion of lights.

$$B_i A_i = E_i A_i + \rho_i \sum_{j=1}^{n} B_j F_{ji} A_j \tag{1}$$

where

B_i = radiosity of patch i (energy/unit area/unit time),

E_i = emission of patch i (energy/unit area/unit time),

A_i = area of patch i,

A_j = area of patch j,

F_{ji} = form factor from j to i (fraction of energy leaving patch j which arrives at patch i),

ρ_i = reflectivity of patch i,

n = number of discrete patches.

Using the reciprocity relationship for form factors and dividing through by A_i, the equation (1) is expressed as follows,

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} \tag{2}$$

3. The following simultaneous equation is derived from equation (2). Solving the equation yields the radiosity for all patches.

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \tag{3}$$

4. Compute the intensity of the vertex for each patch, and then render the image using bilinear interpolation.

3.2. Form Factors

Form factors represent the fraction of radiant energy from one patch to another. If two patches i, j are placed as shown in Figure 1, the form factor from patch i to patch j is represented as follows,

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} dA_j dA_i \tag{4}$$

It is impractical to compute precise form factors using this equation when two patches are placed in complex situations where some obstacles are located between them, for instance. This is because the overwhelming cost of the radiosity method lies in the computation of the form factors[3]. Thus, the hemicube method was studied in order to compute approximate values at high speed.

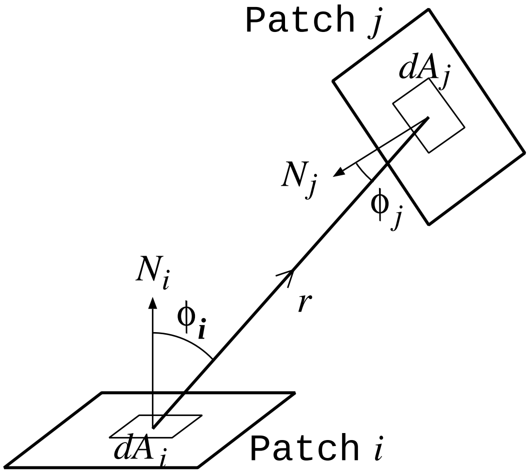


Figure 1: Form factors.

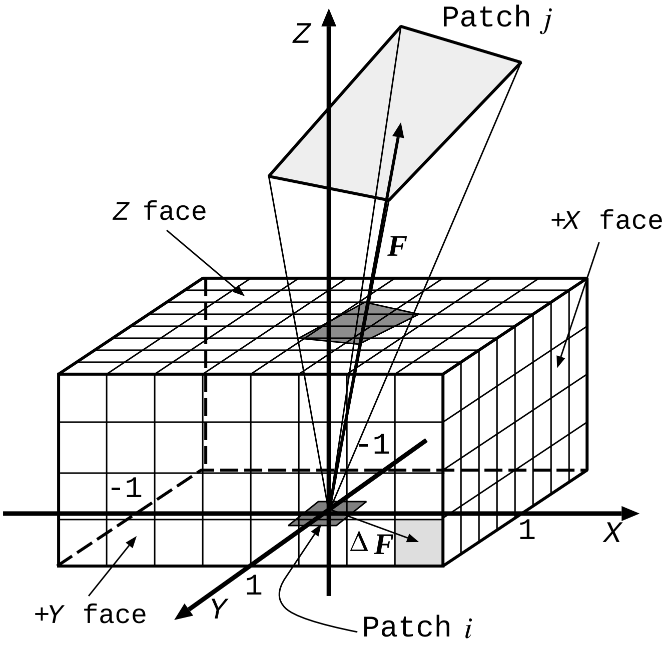


Figure 2: Hemicubes.

3.3. Computing the Form Factors Using the Hemicube Algorithm

An imaginary cube is created around the center of a patch, and every other patch in the environment is projected onto the cube[2]. Consider a hemicube the values in the x and y axes of which are -1 to 1, and those in the z axis are 0 to 1, placing patch i in the center, as shown in Figure 2.

First, divide the surface of the hemicube into square pixels. In Figure 2, the top surface is divided into 8 $\times$ 8 pixels. Calculate the delta form factor for each pixel using the following equations, and make a lookup table.

For the top surface,

$$\Delta F = \frac{1}{\pi(x^2 + y^2 + 1)^2}\Delta A_i \tag{5}$$

For the side surfaces,

$$\Delta F = \frac{z}{\pi(x^2 + z^2 + 1)^2}\Delta A_i \tag{6}$$

Then, project patch j onto the hemicube using the Z buffer algorithm. Finally, leave only the nearest objects by removing the hidden surfaces. After determining which patch j is visible at each pixel on the hemicube, a summation of the delta form factors for each pixel occupied by patch j determines the form factor from patch i at the center of the cube to patch j. This summation is performed for each patch j, and a complete row of N form factors is formed[2]. Thus, F_{ij} can be computed using the lookup table as the sum of delta form factors, ΔF, projected on the mesh, that is $\Sigma\Delta F$.

In our approach, polygons rather than patches are projected onto the hemicube, and the hidden surfaces are removed. Then it computes which patch in the polygons corresponds to each pixel. Hence, the number of projection is equal to the number of surfaces of polygons and is not dependent on the number of patches. Thus, projection and hidden surface removal can be done at high speed even when the number of divided patches is large.

3.4. The Progressive Refinement Method

The conventional radiosity algorithm requires a computation of $O(n^2)$, where n is the number of discrete surface patches, and $O(n^2)$ storage. In order to reduce the enormous cost of the algorithm, the progressive refinement method was devised, where a useful image is produced in $O(n)$ time and $O(n)$ storage[3]. In the progressive refinement method, the radiosity of all patches is simultaneously updated, and patches are processed in sorted order according to their energy contribution to the environment.

In equation (2), the light leaving patch i is determined by gathering in the light from the rest of environment. The progressive refinement method, however, focuses on shooting light out from a patch into the environment, and it repeats the shooting of the patch to other patches so that the radiosities of all patches reaches a state of equilibrium. The algorithm provides a useful solution which progresses gracefully and continuously to the complete radiosity solution[3].

Focusing on shooting yields the following assignment.

$$B_j \leftarrow B_j + B_i(\rho_j F_{ji}) \quad (j = 1, 2, 3, \ldots, n) \tag{7}$$

It is possible to progressively approximate to the solution by iteratively applying this assignment to patch i with the greatest radiosity one by one until the solution converge. In the previous methods which solve a simultaneous equation, all form factors in equation(3) are used, while this method uses only one column of matrices as in assignment(7).

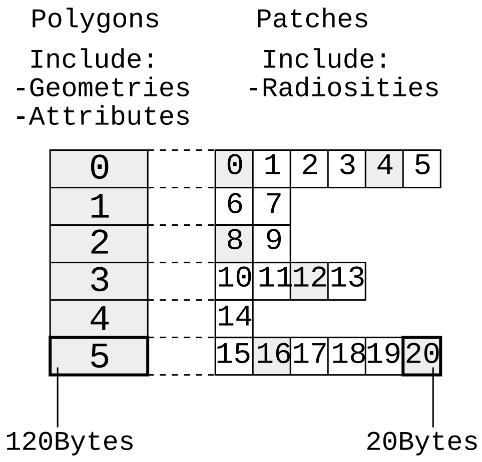


Figure 3: Allocation of scene descriptions.

4. Parallel Computation of Form Factors

Polygon data are copied to all processors, while patch data are distributed to each processor in an interleaved fashion. For example, when four processors are used, processor 0 holds the hatched parts in Figure 3. All processors hold all polygon data with coordinates so that no transfer of coordinates is required. Meanwhile patch data are distributed in order to reduce the total amount of memory requirements.

Computing form factors generally occupies most of the total computation time in the radiosity method, and thus parallelization of the calculation of form factors is most important. In this research, meshes of the five surfaces of a hemicube are grouped into columns, as shown in Figure 4.

The outline of a parallel progressive refinement method is shown in Figure 5. Assume that all patches are numbered from 0 and allocated to each processor in an interleaved method. First, a patch with the greatest radiosity in each processor is selected. These data are collected in one processor in order to determine the patch with the greatest radiosity, and then the patch is broadcast to all processors. After that, partial form factors are calculated in each processor using Z buffer rendering when polygons are projected onto the hemicube. The partial form factors are sent to other processors in order to compute the complete form factors.

Here, the partial form factor that is four bytes per patch is sent to the raw direction first and then to the column direction in the torus network, by repeatedly adding the result of the previous processor and transferring it to the next processor.

Note also that each processor has only those elements of the complete form factors which the processor has to compute. Finally, the contribution from the radiosity of the patch with the greatest radiosity to the radiosities of all other patches is simultaneously calculated in each processor.

Thus, the search for a patch with the greatest radiosity, form factor calculation, and shooting from the patch to all other patches are parallelized. Among those, form factor calculation occupies more than 99% of the total calculation, thus this must be simultaneously calculated by multiple processors with the load balanced. Synchronization is done twice in one iteration, as shown in Figure 5.

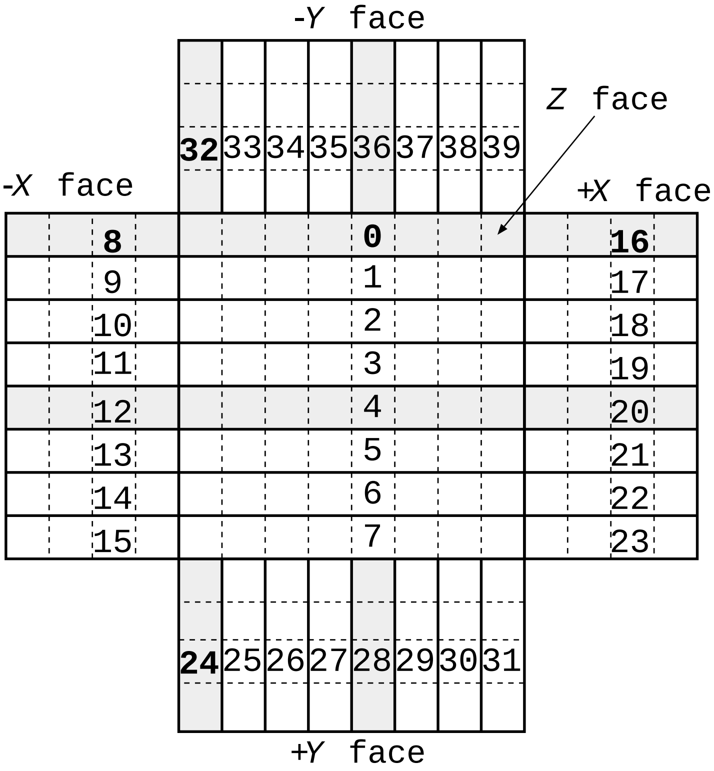


Figure 4: Division of hemicubes.

4.1. Static Load Balancing

Hemicubes are subdivided into mesh like elements by the product of the number of subdivision and five, and each element is numbered like those in Figure 4. Here, the number of subdivisions is usually a multiple of the number of processors. In Figure 4, since the number of subdivisions is eight, the top surface is numbered from 0 to 7, and side surfaces are numbered from 8 to 39. The top surface which requires a large amount of computation is allocated first, and then the side ones are allocated so that load balancing can be achieved.

In order to achieve load balancing among processors even when objects are unequally placed, elements of a hemicube are allocated to each processor by an interleaved method where column l of a hemicube is computed by a processor of (l mod Tn), assuming that the number of processors is Tn. In Figure 4, for instance, all hatched parts are computed by processor 0, assuming that the number of subdivisions is eight and they are computed by four processors. Since many polygons are projected onto the hemicube, and the surfaces of a hemicube are subdivided as in Figure 4, the load in each processor becomes close.

Since one mesh of the hemicube is 8 bytes, the total amount of memory for the hemicube is 864 Kbytes when the number of subdivisions is 192. The memory requirement for one processor, however, is 864/64= 13.5 Kbytes when 64 processors are used.

4.2. Dynamic Load Balancing

Two threads, an allocator and a calculator, are provided in each processor. Tokens which include a task number and a termination flag flow through the allocator in each processor. The simplest way to achieve dynamic load balancing is to find an idle processor out of all of

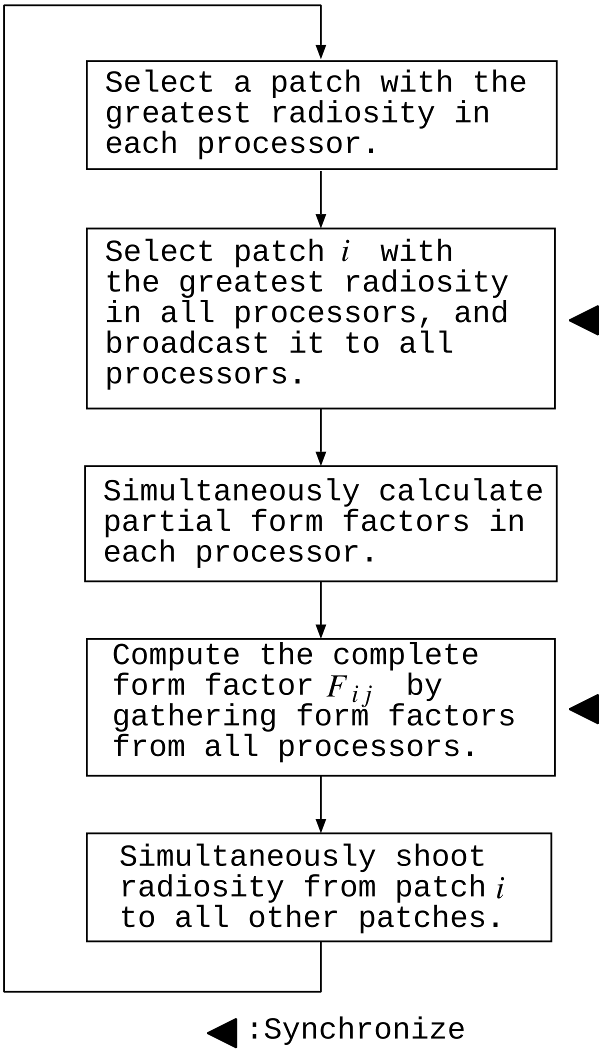


Figure 5: A parallel progressive refinement method.

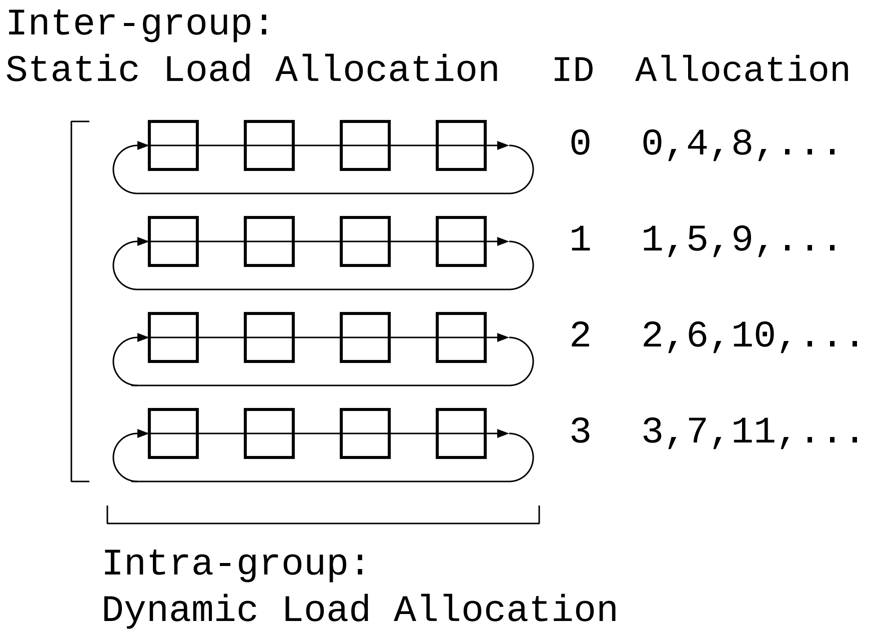
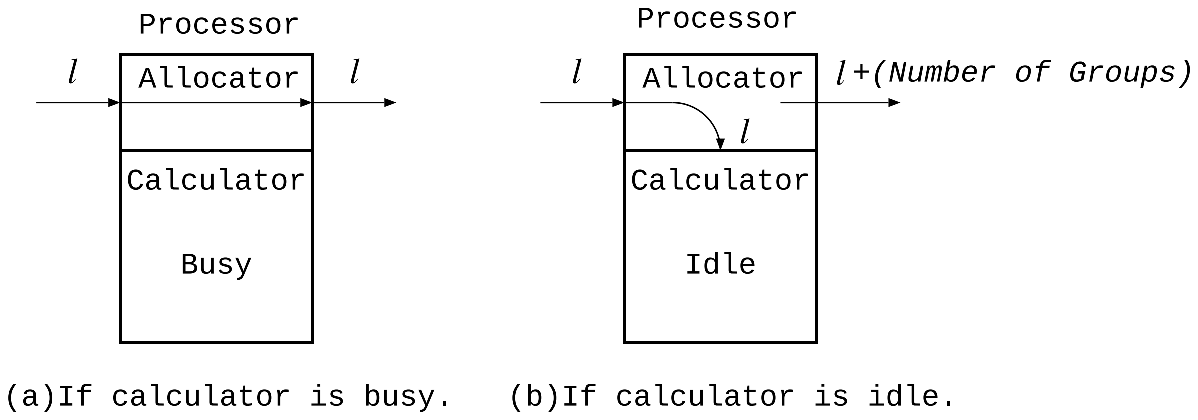


Figure 6: Dynamic load balancing by grouping.

the remaining idle processors.

One problem in such a method is that it takes some time for the next token to arrive after one token is calculated as the number of processors increases. In order to solve this problem, grouping of processors is used, where processors are grouped by each row and dynamic load balancing is done in each group, as shown in Figure 6. Since the whole load is divided and equally allocated to each group, this method can be considered as a mixture of static load balancing among each group and dynamic one in each group.

In Figure 6, the task number starts from the number of groups. As shown in Figure 7, if the processor is idle, the task number is transferred to the calculator, and the task number added by the number of groups is sent to the next processor in the group. Meanwhile, if the processor is busy, the task number is transmitted to the next processor. All of the



(a)If calculator is busy. (b)If calculator is idle.

Figure 7: Two threads in each processor.

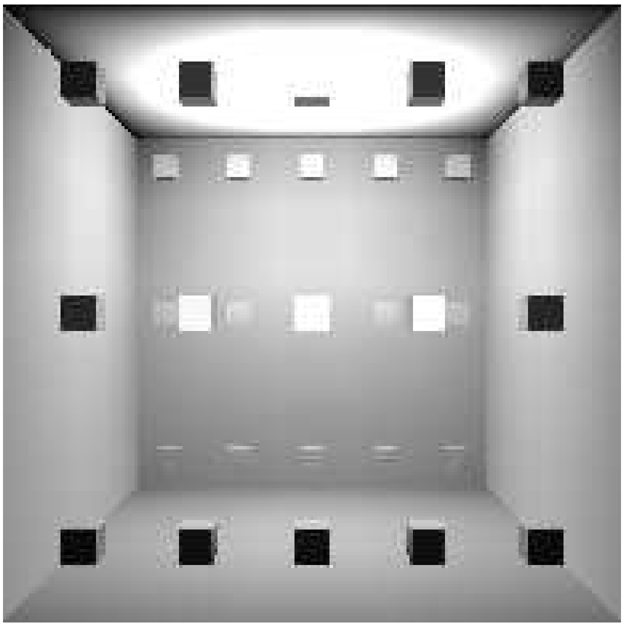


Figure 8: Scene 1 (192 polygons and 894 patches).

idle processors in a group are allocated tasks successively in this way. A termination flag represents that all elements of the hemicubes are allocated to processors, and it orders each processor to start communication to gather partial form factors.

Each processor keeps the total memory area for the hemicube, because the number of lines each processor calculates is variable in dynamic load balancing.

5. Experimentation of Parallelization

5.1. Experimental Method

Radiosity image generation is parallelized using static/dynamic load balancing algorithms. The time needed for each execution is measured on a multi-transputer system of 4, 16, 32 and 64, respectively, connected by torus networks. Here, the execution time is measured from the time when data are transferred to all processors until the results are collected by the root processor. Two sample scenes for benchmarking shown in Figures 8 and 9 are executed. Shooting of the progressive refinement method is done 100 times for two cases, that is the number of hemicube subdivisions is 128 and 192.

Scene 1 consists of 192 polygons divided into 894 patches, and scene 2 consists of 372 polygons divided into 1614 patches. In scene 1, 30 cubes, six walls and one light source are provided. Each cube consists of six surfaces which are divided into 2×2 patches. Each wall is divided into 5×5 patches, and the light source consists of six surfaces which are divided into 2×2 patches. Thus, the total number of patches is 894. Similarly, 60 cubes, that is 15×4, are provided in scene 2.

5.2. Experimental Results and Discussion

A comparison between static and dynamic load balancing is shown in Figure 10. It is based on the number of hemicube subdivisions as shown in Figure 11. Here, HC represents

Figure 9: Scene 2 (372 polygons and 1614 patches).

the number of hemicube subdivisions, and S, D shows static and dynamic load balancing, respectively. Figure 10 shows that the processing time of dynamic load balancing is slightly smaller than that of the static one.

5.2.1 Execution Time

In terms of the number of hemicube subdivisions, the execution time for HC192 divided by that for HC128 varies from 2.16 to 1.53 for scene 1, and 2.11 to 1.55 for scene 2. The quotient comes close to 1.5 as the number of processors increases. This means that the execution time is proportional to the number of hemicube subdivisions and the execution time decreases more linearly when the number is larger. Speedup is around 30 for these two scenes when 64 processors are used. The execution time using 64 T805s, 25MHz, is 39 seconds for scene 1, and 77 seconds for scene2, assuming that the number of hemicube subdivisions is 128.

The execution time is a function of mainly the number of hemicube subdivisions and the number of polygons[13], because polygons rather than patches are projected on hemicubes. It also relates to the target architecture and the data allocation method. Formalization of the execution time using these parameters is a future issue.

5.2.2 Memory Requirements

In terms of the space complexity, this method is very attractive for multi-transputer systems even when a small amount of memory is attached to each processor, because data for patches including the form factor matrix is subdivided into multiple elements which are equally allocated to each processor. Thus, more complex scenes can be processed by increasing the number of processors in this approach where 20 bytes are required for a patch and 120 bytes are required for a polygon. In scene 1, for example, 40.0 Kbytes are required for each processor if all polygons and patches are copied into every processors, while 22.8 Kbytes are required if the data for patches are subdivided and equally allocated to 64 processors.

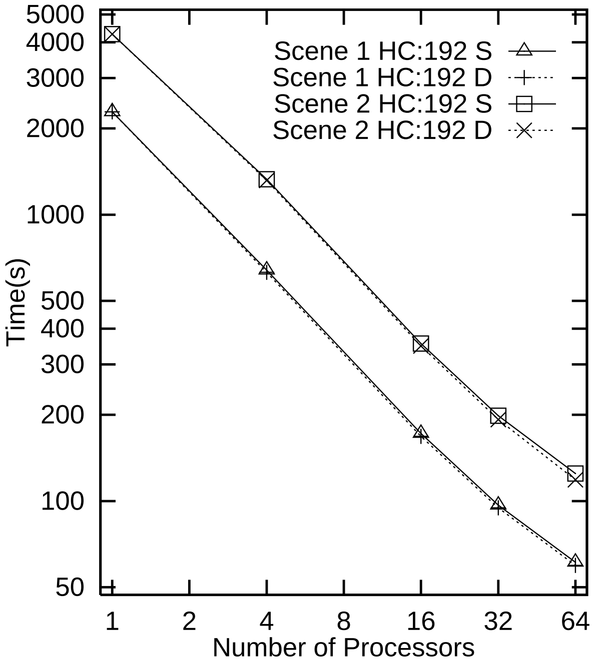


Figure 10: Comparison between static and dynamic load balancing.

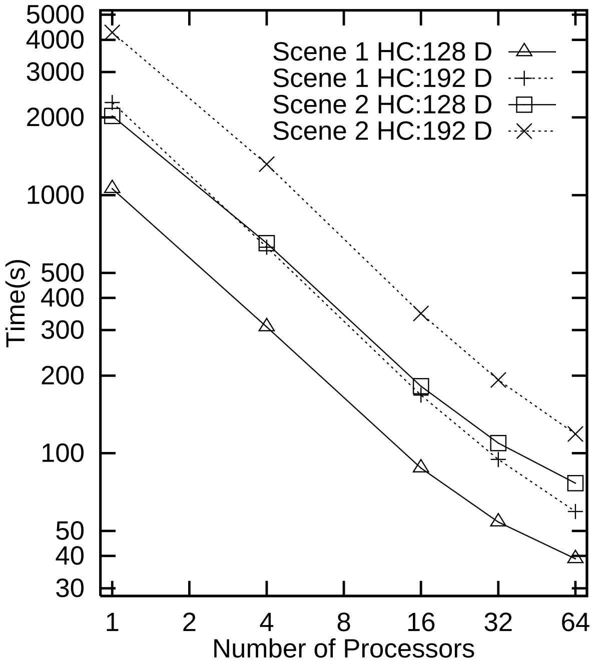


Figure 11: Comparison based on the number of subdivision of hemicubes.

Table 1 Related work

Reference	Parallel machine (Memory type)	Form factor calculation	Parallelization	Data allocation
5	SGI-WS(SM)	Hemicube	Hemicube	Centralized
6	iPSC/2(SM)	Hemisphere	Patch	Shared
7	DASH(SM)	Ray tracing	Patch	Shared
8	T800(DM)	Hemicube	Patch	Distributed
9	Paragon(DM)	Ray tracing	Patch	Distributed
10	nCube2S(DM)	Raycasting	Patch	Distributed
Ours	T805(DM)	Hemicube	Hemicube	Distributed and Copy

Let's consider a maximum sized scene which can be processed on the system. Assume that the number of polygons is n, each polygon is divided into four patches, and 64 transputers are used. Then, the memory requirements for polygons is $120n$ bytes and that for patches is $20 \times 4n \div 64 = 1.25n$ bytes. This means that the memory requirements for patches is just 1% of that for polygons, which can be negligible. In addition, an area for a hemicube can be distributed in static load balancing. Thus, around 7500 polygons and 30000 patches can be processed when 1Mbytes memory is provided in each processor and static load balancing is used.

5.2.3 Load Balancing

In terms of the load balancing algorithms, the processing time of the hemicubes is dependent on a processor the rendering time per one shooting of which is maximum. The maximum execution time in each processor of dynamic load balancing is 7% smaller than that of the static one for scene 2, and the difference is 4% for scene 1. Thus, the differences between dynamic and static load balancing algorithms are very small for these scenes. However, if the number of hemicube subdivisions is not a multiple of the number of processors, load imbalance increases for the interleaved partitioning of static load balancing. For example, if the number of subdivision is 160, the execution time for dynamic load balancing is 12% smaller than that of the static one, thus the dynamic load balancing is useful for such cases.

6. Related Work

Parallel machines have been applied to compute form factors which occupy almost all of the computation time. Table 1 shows the features of related work. Baum et al. described parallelization of the progressive refinement method using a producer-consumer model on shared memory multiprocessor workstations, and they developed an analytical performance model[5]. Bouatouch et al. described how to parallelize the radiosity method on shared virtual memory parallel machines and demonstrated the accelerations obtained with up to 32 processors[6]. Singh parallelized the hierarchical radiosity using dynamic task stealing on shared-address-space multiprocessors and showed the speedup obtained with up to 48 processors[7]. Feda et al. described a parallel implementation of the progressive refinement algorithm on a transputer network, where worker transputers compute the radiosity values of all patches using hemicubes, and rendering transputers generate images of actual results[8]. Guitton et al. proposed a parallel implementation for an extended stochastic progressive radiosity method, where form factors are computed with a ray tracing scheme, on a network of processors with a distributed memory and a message passing mechanism[9]. Sturzlinger et al. presented several load distribution schemes for such a parallel algorithm which includes progressive refinement and adaptive subdivision for fast solutions of high quality[10].

The criteria for evaluating parallel radiosity methods are target architectures, form factor calculation methods, the level at which the parallelism is exploited, and the data allocation of scenes. [5]-[7] were experimented with shared-memory parallel machines, while [8]-[10] and our research were done on distributed-memory machines. As for the form factor calculation, hemicubes are used in [5],[8] and in ours, hemispheres are used in [6], ray tracing or raycasting is used in [7], [9] and [10]. The most important issue is the level at which parallelism is exploited. In our approach and [5], hemicubes are subdivided into small elements that are equally allocated to each processor, while all patches are subdivided and equally allocated to each processor in other approaches. As for data allocation, the scene is shared in [5]-[7], while data are distributed in [8]-[10] and in our approach. In our approach, polygon data are copied into all processors, while patch data are distributed to each processor.

In shared-memory machines, parallel programming is easier and good performance is obtained provided that data locality is utilized on suitable sized cache memories. Meanwhile, in distributed-memory machines, hardware scalability is better, although communication overheads must be reduced with effort and parallel programming is more difficult.

7. Conclusions

This paper has described a parallelization method for the radiosity image generation on a multi-transputer system. The calculation of form factors, which occupies the overwhelming cost of the radiosity method, is parallelized by subdividing hemicubes into small elements and allocating them to each processor using static/dynamic load balancing algorithms. The experimental results demonstrated that the execution time is proportional to the number of hemicube subdivisions, and it decreases almost linearly as the number of transputers increases provided that polygon data are copied to all processors, while patch data are distributed to each processor. Dynamic load balancing is more effective than static one when the number of hemicube subdivisions is not a multiple of the number of processors.

More complex image generation on the multi-transputer system and similar experimentation on AP1000, another distirbuted memory parallel machine, are currently being done. The relationship between the number of iterations in the progressive refinement method and the reality of images should be investigated in the future.

Acknowledgements

The authors would like to thank Kobe Steel Ltd. for supportung part of this research.

References

[1] Goral, C.M., Torrance, K.E., Greenberg, D.P. and Battaile, B.: Modeling the Interaction of Light Between Diffuse Surfaces, *Computer Graphics(Proc. of SIGGRAPH '84)*, Vol. 18, No. 3, pp. 213–222, 1984.

[2] Cohen, M.F. and Greenberg, D.P.: The Hemi-cube : A Radiosity Solution for Complex Environments, *Computer Graphics(Proc. of SIGGRAPH '85)*, Vol. 19, No. 3, pp. 31–40, 1985.

[3] Cohen, M.F., Chen, S.E., Wallance, J.R. and Greenberg, D.P.: A Progressive Refinement Approach to Fast Radiosity Image Generation, *Computer Graphics(Proc. of SIGGRAPH '88)*, Vol. 22, No. 4, pp. 75–84, 1988.

[4] Cohen, M.F. and Wallance, J.R.: Radiosity and Realistic Image Synthesis, *Academic Press Professional*, 1993.

[5] Baum, D.R. and Winget, J.M.: Real Time Radiosity Through Parallel Processing and Hardware Acceleration, *Proceedings 1990 Symposium on Interactive 3D Graphics*, pp.67–76, 1990.

[6] Bouatouch K, and Priol T.: Data Managenent Scheme for Parallel Radiosity, *Computer–Aided Design*, pp.876–882, 1994.

[7] Singh, J.P. et al.: Parallel Visualization Algorithms, *IEEE Computer*, pp.45–55, 1994.

[8] Feda, M. and Rurgathofer, W.: Progressive Refinement Radiosity on a Transputer Network, *Photorealistic Rendering in Computer Graphics(Proc. of the Second Eurographics Workshop on Rendering*, pp.139–148, 1994.

[9] Guitton, P., Roman, J. and Subrenat, G.: Implementation Results and Analisis of a Parallel Progressive Radiosity, *Proc. 1995 Parallel Rendering Symposium*, pp.31–38, 1995.

[10] Sturzlinger, W., Schaufler, G. and Volkert, J.: Load Balancing for a Parallel Radiosity Algorithm, *Proc. 1995 Parallel Rendering Symposium*, pp.39–45, 1995.

[11] Yamazaki, K., Kanatani, T., Watanabe, T. and Tokumaru, H.: Parallelization of Back-Propagation Learning using Several Interconnection Networks on a Multi-Transputer System, *World Transputer Congress*, pp.668–680, 1993.

[12] Uejima, A., Yamazaki, K., Watanabe, T. and Tokumaru, H.: Parallelization of the Radiosity Method on a Multi-Transputer System, *IPSJ Joint Symposium of Parallel Processing*, Vol.95, No.2, pp.19–26, 1995.

[13] Uejima, A., Yamazaki, K., Watanabe, T. and Tokumaru, H.: Parallelization of the Radiosity Method on a Multi-Transputer System, *Journal of Information Processing Society of Japan*, Vol.37, No.7, 1996.