Parallel Radiosity on a Cluster of Workstations

Libor Šindlar, Josef Pelikán

Deptartment of Software and Computer Science Education
Faculty of Mathematics and Physics, Charles University,
Malostranské náměstí 25, 118 00 Praha 1 - Malá Strana
email: {sindlar|pepca}@sun1.ms.mff.cuni.cz

Abstract

A parallel version of progressive refinement radiosity with radiosity overshooting and adaptive surface subdivision is presented. A cluster of workstations under the operating system Unix was chosen as parallel hardware. The whole system was organized according to the master - worker paradigm. We investigated how to adjust algorithms for run-time adaptive surface subdivision and overshooting for such parallel hardware, especially if we use a relatively slow computer network or, moreover, a network with highly varying throughput. Variation in computer performance (including the possibility that some workstations might crash) was also considered.

Keywords: radiosity, parallel algorithms, overshooting, progressive refinement, PVM

1. Introduction

Radiosity has proved to be a suitable method for generating high-quality photorealistic images. Because radiosity describes the light transport in scenes globally, it is relatively slow. In addition, the original so-called full-matrix radiosity method required a lot of memory but fortunately progressive refinement [5] has eliminated this requirement.

At present time the radiosity method is being enhanced in several ways. If we want to enumerate some of them we must mention various ways to compute form factors (hemicube [3], ray tracing [14]), surface subdivision and mesh generation before and during computing - not to mention different methods for working with non-diffuse surfaces [11]. Finally we should mention that radiosity is now often parallelized.

One of the most time consuming parts of radiosity is form factor computation. That is why form factor computation is the most commonly parallelized part of radiosity. Recently several means of radiosity parallelization have been published. We shall mention client-server architecture [12], transputer networks [7], and the use of some special capabilities of graphic workstations [1]. In most of the previous articles some special parallel hardware (such as a transputer network, a multi-processor workstation or Connection Machine 5 [2]) was used.

To summarize these articles, we can say that authors usually had a lot of relatively slow processors with only a small amount of memory (several MB) connected by a very fast network. Such configurations allow parallel computation of form factors from one patch to the rest of a scene. But only a few people can work with special parallel computers - on the other hand almost everyone can use several workstations connected by a network. A great advantage of such hardware is its availability and generality. Another advantage is relatively high workstation's performance and a lot of memory. So we need not to divide large scenes across the network. Small network throughput and big differences between individual node performances are main disadvantages of the method. Computers' computing power and network throughput can also fluctuate in time.

2. Classical algorithm

The basic radiosity algorithm can be found for example in [6]. We will give only brief description of some features which are interesting in parallelization.

One of the most important enhancements is a progressive refinement radiosity. In the original full-matrix radiosity we had to compute the whole form factor matrix. Usually the Gauss-Seidel iteration was used to solve the linear system of equations, because the matrix was diagonally dominant.

Progressive refinement is based on the idea of ordered shooting. We don't gather the energy coming from the environment to the plane as the Gauss-Seidel iteration did. Instead, we shoot energy from the plane to the environment. The most luminous planes are the most important, so we first shoot radiosity from them. Doing that, we soon got acceptable results. As the method does not remember the computed form factors, they must be computed again if we need them once more.

The precision of light distribution description depends strongly on the amount of planes used to approximate object surfaces. As the number of planes grows, memory requirements increase and speed falls down. To solve this problem, an adaptive surface subdivision [4] is often used. At the beginning we start with a relatively small number of planes. These planes are called „patches“. If we find a patch with a great light variance (such as shadow boundaries), the patch is divided into smaller elements. If the variance is not reduced enough, the elements can be divided again and so on. Thus we use a plane hierarchy to achieve enough precision of the light description without enormous memory consumption. Originally, the adaptive surface subdivision was designed for full-matrix radiosity. In progressive refinement we shoot energy from the patches to the elements. Patch energy is computed as the sum of energy of all elements the patch consists of. A big effort was done to find some method for the patch division before the whole computing begins. But because we cannot do very good prediction of light distribution in a scene, planes must be divided at the run-time as well.

Progressive refinement radiosity can also be accelerated by overshooting [8]. Overshooting is based on the fact that we can *a priori* estimate the amount of radiosity received by the shooting patch in later iterations. This estimated radiosity is then shot together with actual unshot radiosity. If we were too optimistic and shot too much (overshot), we could correct it by shooting negative radiosity. The problem is how to predict radiosity that will be received in future iterations. Ambient term is often used for this estimation and some more sophisticated methods can be found in [9].

3. Division into modules

We chose the master-worker architecture to parallelize radiosity computation [13]. This architecture has one great advantage - we need not worry about dynamic load balancing because it is done automatically.

There are four types of processes in our system. The Master and Worker processes will be described later in detail, respecting the parallelization of the earlier-mentioned radiosity enhancements. The Monitor and Display processes will be mentioned only briefly, because they are not so important for radiosity parallelization. The next picture shows the relationships between modules.

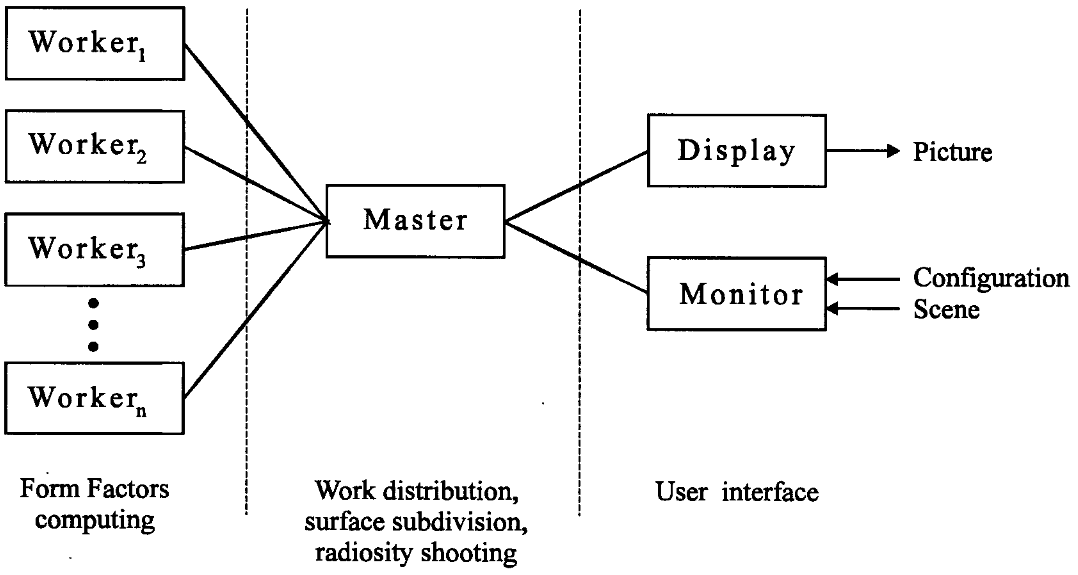


Figure 1: Process structure in parallel radiosity.

User starts the Monitor process and gives a configuration file to it. This file contains a list of available computers by their Internet addresses. Monitor analyzes the input scene as well and sends the result to Master. Finally, on user demand, Monitor stops the whole computing.

The system is directed by the Master process. Master is responsible for Worker processes initializing, selecting and distributing patches, shooting radiosity from patches to elements according to computed form factors. Master is also responsible for run-time adaptive surface subdivision. Finally, Master sends actual results to the Display process.

Form factors are computed by Worker processes. The hemicube algorithm is used here.

The Display process allows the user to see the actual results and to walk through the computed scene. Display process can also save images to disk.

4. Parallel algorithms

A progressive refinement radiosity with run-time adaptive surface subdivision and ambient overshooting was parallelized. All the processes in our method have geometric information of the entire scene. The whole algorithm may be briefly described as follows: Master selects patches for shooting. Identifiers of the patches are sent to Workers so that they can compute form factors. Each Worker computes the form factors from one patch to the rest of the scene (one column of the form factors matrix). The computed results are sent back to Master. Then, according to these form factors, Master shoots radiosity from the patch to the scene. Meanwhile, Master carries out the adaptive surface subdivision and broadcasts changes in the plane hierarchy to all Workers. Now these basic steps will be described in more detail.

4.1 Work distribution

The Master process maintains a list of active Workers. To decrease sensitivity to network throughput each Worker has a input queue (3-10 patches according to network size). Then, if the network throughput is low for a while, Workers have still enough to do to ensure full utilization. This queue also ensures full utilization when Master adaptively divides planes. The size of these queues must be selected very carefully - large queues may degrade the dynamic load balancing efficiency especially when running on workstations with different speed. Large queues can also decrease the overshooting efficiency because it will take a long time from selection of a shooting patch to the shot itself (this may cause a non-optimal shooting order). On the other hand, if we work on a slow or fluctuating network, the queue size should be increased. In conclusion, the size of the queues should be set to the smallest value that ensures full utilization of the Workers. In our implementation this value must be set manually before the computing begins.

Let's describe the structure of Master's list of active Workers in detail. Master maintains an array for each Worker containing patch identifiers actually sent to the Worker. Master also has a queue of patches prepared to be send to the Worker in future. If any patch is either in the array or in the queue it is owned by the Worker and cannot be given to another Worker. The array containing patches from a Worker's queue is used only when the Worker crashes and patches must be released so that another Worker can process them. On the other hand the queue of patches is used very intensively. As soon as the Worker has done its computing it sends results (form factors) back to Master. Form factors are packed to minimize network load. One of Master's main tasks is to ensure full utilization of the Workers. That's why sending patches to Workers has higher priority than shooting radiosity.

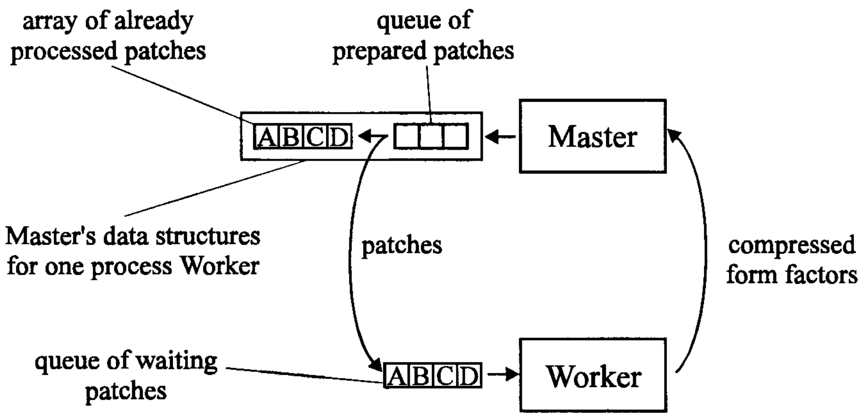

array of already processed patches
queue of prepared patches
A B C D
Master
Master's data structures for one process Worker
patches
compressed form factors
queue of waiting patches
A B C D
Worker

Figure 2: Data flow between Master and Worker.

Now we shall describe how the queue of pre-prepared patches is filled. We must note that the filling of this queue is very important for the overall efficiency. Ideally, patches should be processed according to their unshot radiosity. If we had only one Worker it would not be so difficult. But when Master must distribute patches across more Workers we cannot ensure the ideal shooting order. This must be considered when comparing parallel and sequential versions of the algorithm and will be discussed later.

A high / low watermark technique was used so that Master need not fill queues after each radiosity shot. At the beginning two numbers are computed - the high and low mark. Then at the run-time Master watches if the number of pre-prepared patches in some of the Worker's queue drops under the low mark. When that happens Master starts a patch distribution. The distribution continues until the number of pre-prepared patches in all queues reaches the high mark.

When Master starts the distribution of patches to its queues it first makes a cyclic list of Workers that are not fully loaded (number of prepared patches do not reach the high mark). Master wants to distribute the load to all Workers equally. So it would not be a good idea to completely fill the first queue, then the second and so on. Master rather goes through its cyclic list and in each step adds the patch with the biggest unshot radiosity and marks this patch as 'used'. Then Master continues with the next queue. When some queue is filled it is deleted from the list. Master continues the distribution of the patches until the list is empty. This distribution technique is important mainly at the beginning because it ensures that the primary light sources are distributed to all Workers and not only to the first ones.

4.2 Shooting and overshooting

The ambient overshooting technique was implemented to improve convergence. Selection of shooting patches is based on the radiosity prediction (based on actual ambient radiosity). Patches which have been selected and distributed are waiting in Master's queues. Patches are sent to Workers where they wait again before they are processed. It may take a long time from a patch selection to the arrival of appropriate form factors from Worker. Meanwhile, ambient radiosity might be changed or the patch (in particular the elements the patch consists of) might get some radiosity. Therefore new radiosity estimation is done before it is shot.

4.3 Adaptive subdivision of elements and patches

To achieve good approximation of the light distribution in the scene we used an adaptive subdivision of planes (patches or elements) in places of big radiosity change (light and shadow borders). Radiosity distribution in the scene is available only to the Master process - the subdivision is done sequentially by Master. Because the run-time adaptive surface subdivision is a time consuming task some restrictive conditions must hold before it starts. For example it makes no sense to start the division if radiosity has not been significantly changed - Master will not start the division unless 10 shots are made after the last division. At the beginning we have very inaccurate information on light distribution, so we should wait for a while (20-50 shots) before the first division starts. To ensure better Workers utilization Master will start division only if all Workers have enough patches in their input queues. The last condition is lack of any messages for Master (no computed form factors are waiting for shooting).

Master uses following strategy while dividing planes: For each undivided plane „dividing factor“ is computed - this factor says how much the plane needs to be divided. It includes radiosity gradient, plane size and maximum division level for neighbour planes. The factor is compared to some threshold value and if it exceeds the plane is divided. Then Master broadcasts changes in the plane hierarchy to all Workers. As the whole radiosity algorithm continues the threshold goes down so more and more planes are divided. Finally the threshold stops at some minimal level.

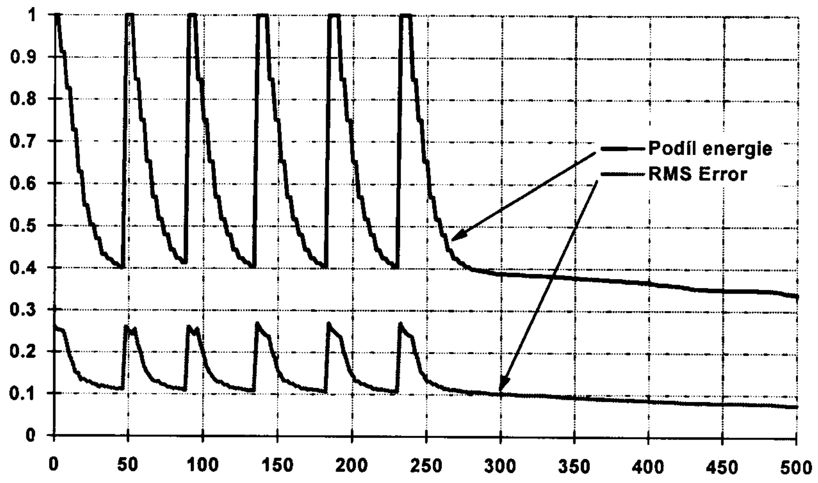
4.4 Wave computing

Adaptive run-time subdivision is connected to problem of radiosity distribution to divided elements. In the classical full-matrix radiosity iteration took only a small part of the total time. It was not difficult to solve the whole linear system again after a matrix change. But the progressive refinement method does not remember computed form factors - adaptive surface subdivision must be done in the time of form factors computation. There are several ways of distributing the radiosity received from an environment.

The easiest way is to give the same amount of received radiosity to all new elements. But why did we divide the plane then? We divide planes with big radiosity differences. A better idea is to divide the radiosity according to its gradient. But both these methods are very inaccurate.

A more accurate but very time expensive method is based on a form factors reciprocity. This method is similar to the full matrix radiosity solution of the surface subdivision. We can compute the form factors from the new elements to the whole scene and then use the equation $A_iF_{ij}=A_jF_{ji}$ for computing form factors from the scene to our new elements. These form factors can be used for reshooting from patches that have been already shot. Unfortunately this method spends a lot of time especially when we divide planes very often.

We introduce a method of „computing in waves“ for progressive refinement radiosity: in the first phase we use some easy method of radiosity distribution. After dividing about 20% of elements we throw away all computed radiosities and start from scratch (but with finer element mesh) and so on... Thus we can split the computing process into several phases („waves“). At the end of each phase a division process is done and radiosity values are re-initialized:



Graph 0: Unshot energy and RMS error in „wave computing“

Advantage of this method is high accuracy with acceptable time cost. If we plan to divide very often (sometimes the number of the elements increases up to ten times) this method seems to be very good. The main disadvantage is that the output is not improved continuously - the error of the light description jumps up and down.

The „wave“ method was used in our parallel implementation. Radiosity distribution by its gradient is very fast and can be done sequentially by the Master process. When we start again from the beginning we have to compute form factors of shooting patches again.

5. Implementation and results

The whole system was designed to run on a cluster of Unix workstations connected by a local-area network. A PVM (Parallel Virtual Machine) package [10] was used for inter-process message passing. Because of user-unfriendly features of Unix an emulator WinPVM was developed to enable working under the MS-Windows operating system. Only the final testing and tuning was done on a cluster of workstations.

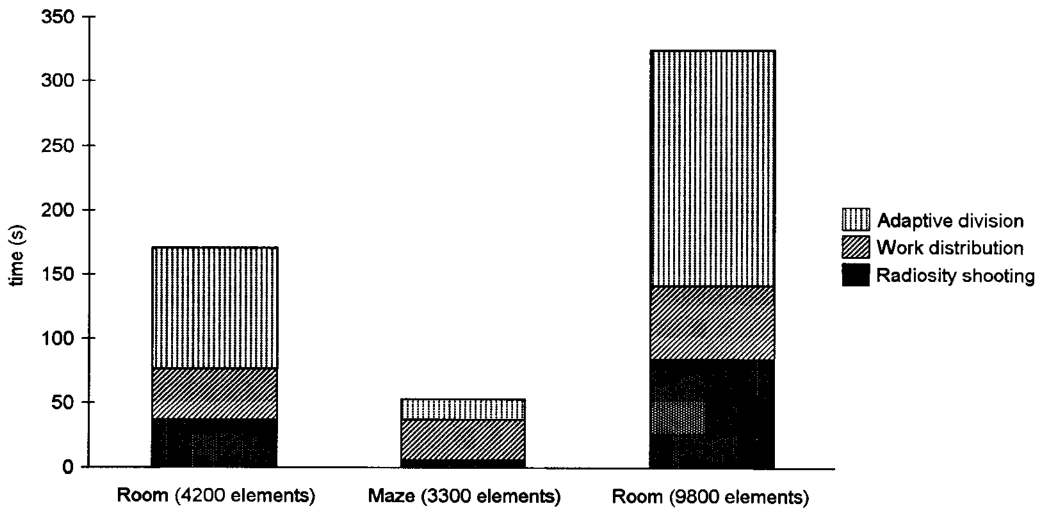
Two different test scenes were used. First one was a „Room“ with several pieces of furniture. There was only one light source and most of plane couples could see each other. The second scene was a „Maze“ consisting of several corridors. There were four light sources in the maze. The maze was relatively complicated and relatively small number of plane couples could see each other.

Next part of this paper is divided into two paragraphs. The first one shows the behavior of our system when it ran on a single computer. This is useful when we want to know what is the maximal possible speedup. The second paragraph shows results achieved when our system ran on a cluster of workstations.

5.1 Single computer

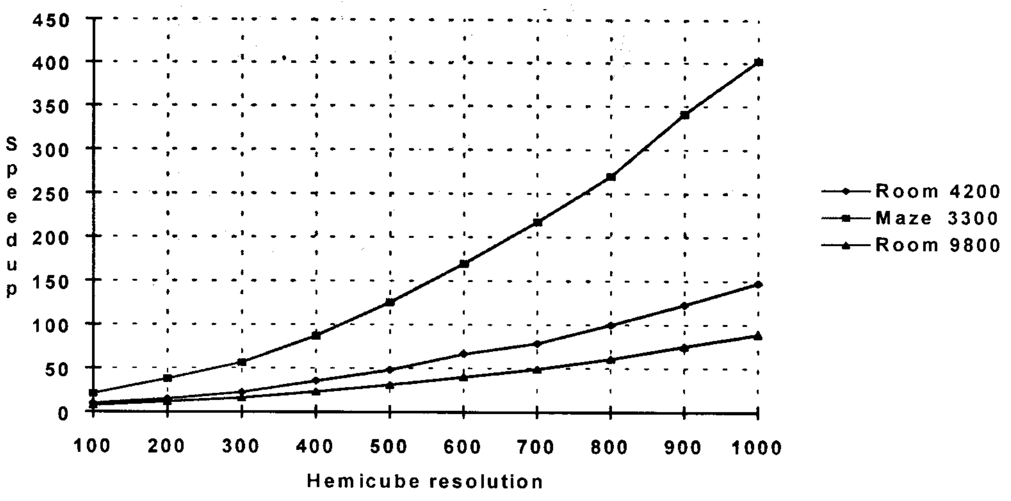
In sequential tests all processes were run on a single PC under the MS-Windows operating system (in particular in the Win32s subsystem). The computer was equipped with the AMD 486DX2-66 processor and 24 MB of RAM.

Time spent by the Worker process depends strongly on a hemicube resolution. On the other side, time spent by the Master process depends mainly on scene character. The next graph shows how much time Master process spends in three different computations. The time of run-time adaptive surface subdivision, shooting patches distribution and radiosity shooting for the first 250 shots was measured. Note that the shooting patches (or "work") distribution depends on the scene character very weakly. It depends only on the number of elements the scene is divided into. On the other side, adaptive surface subdivision and radiosity overshooting depends on scene character very strongly. An example: scene „Maze" is relatively complicated and as most of the plane pairs cannot see each other only 10% of the form factors are non-zero. This fact of course affects the shooting time. Similarly, because there are four primary light sources in „Maze", adaptive surface subdivision started later and took less time. (We had to carry out a lot of shots to be able to make a good radiosity approximation which is needed for surface subdivision.)



Graph 1: Time spent by Master process for various tasks

To be able to forecast real parallel behavior of the whole system we are not interested only in the time consumed by the Master process. We would rather like to know Master to Worker spent-time ratio because this ratio determines the maximum possible speedup. Next graph shows how this ratio depends on the hemicube resolution. Note that more complicated scenes with more primary light sources are more suitable for parallelization. A slightly disturbing fact is that with growing number of elements the maximum possible speedup decreases (compare the maximum possible speedup in „Room" scene for different numbers of elements):



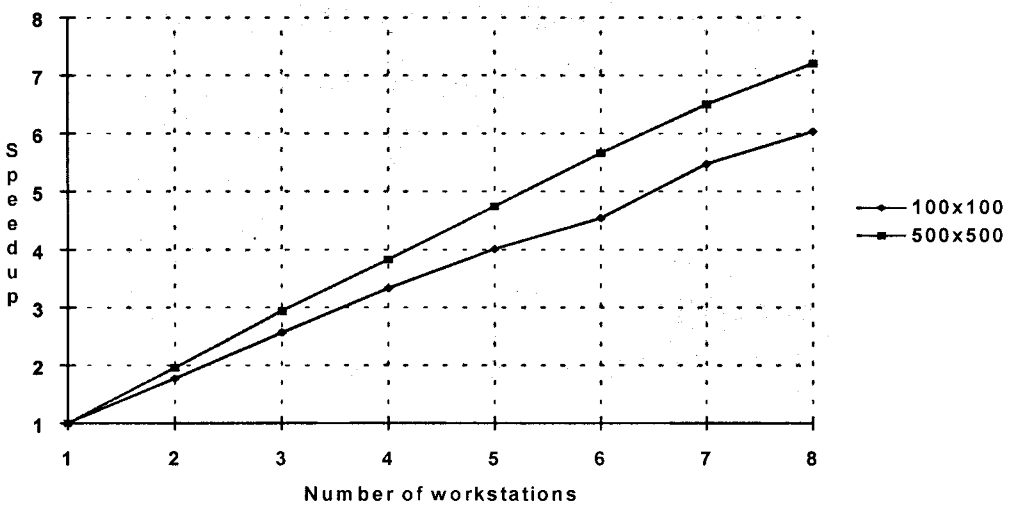
Graph 2: Maximum possible speedup for different scenes

5.2 Real parallel computation

Parallel tests were done on a network with 8 Sun-4 workstations running Solaris 2.3. These computers had 16 MB of RAM, only a computer where both Master and Worker were running was equipped with 32 MB of RAM. Only „Room“ scene was used; 500 shots were carried out.

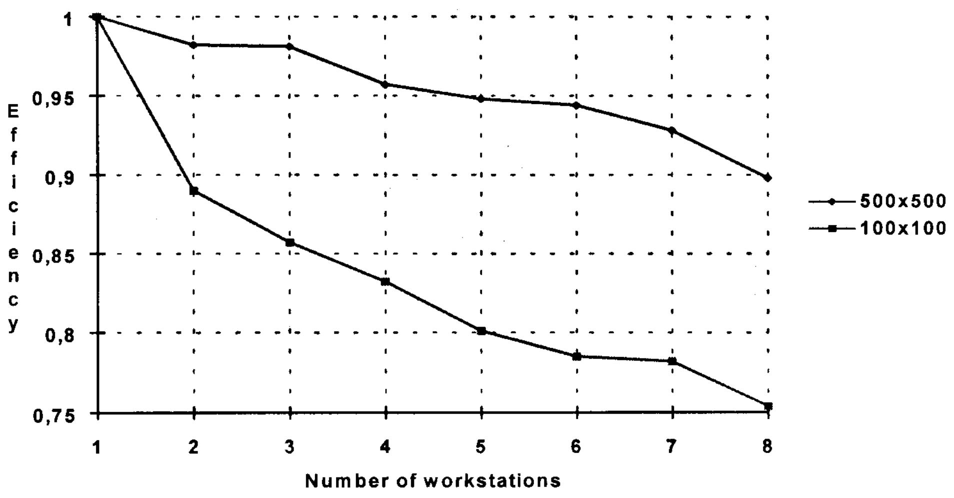
Workstations were connected by a local-area network with maximum theoretical throughput of 10 Mbits per second. When our system was running on all 8 workstations, it used to transfer less than 10 KB per second. Therefore (considering also maximum possible speedup from graph 2) one can expect that this model is scaleable up to 50-100 workstations according to their speed.

Workers’ speed strongly depends on hemicube resolution but Master’s speed does not. This difference can be used in testing. We can use small hemicube resolution to simulate a whole system on a larger number of computers than is available. Next graph shows the speedup achieved on the Room scene for two different hemicube resolutions.



Graph 3: Speedup achieved for two different hemicube resolutions

Another interesting parameter is overall efficiency. It is defined as the speedup divided by the number of processors. The efficiency has smaller range of values (from 0 to 1) which makes the graph more readable.



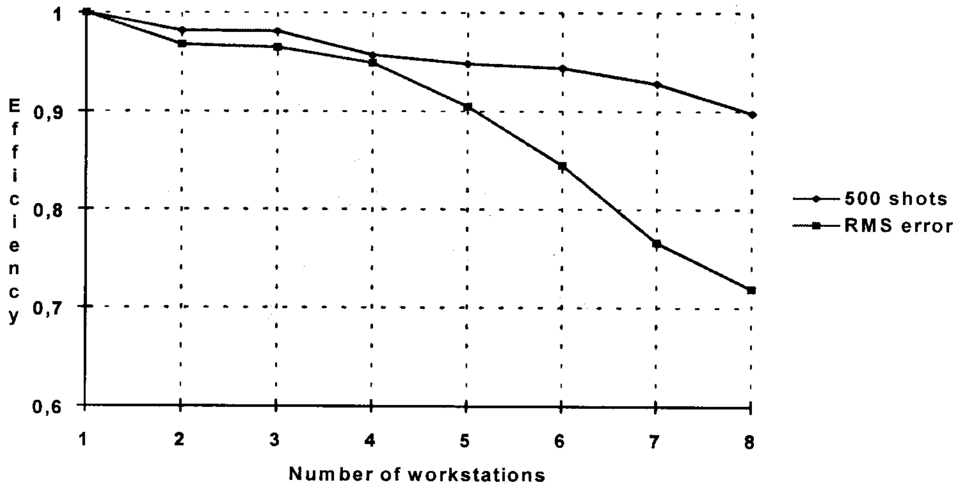
Graph 4: Efficiency achieved for different hemicube resolutions

These results were based on the time needed to perform the first 500 radiosity shots. But computing light distribution we are not usually interested in the number of shots. We would rather like to have good pictures as soon as possible. But what does that mean, "good pictures"? Picture quality is a subjective term and it is difficult to measure. So we must use something else. To measure the distance from the converged result the RMS Error is often used. It is defined by

$$RMS\,Error = \sqrt{\frac{\sum_{i=1}^{N}\left(\left(B_i^* - B_i\right)^2 A_i\right)}{\sum_{i=1}^{N} A_i}},$$

where B_i^* denotes converged radiosity of i-th patch, B_i denotes the actual estimated radiosity, A_i stands for the area of i-th patch and N is the total number of elements.

To get the next graph, we first computed the converged radiosity B_i^*. Then the whole system was run on one computer and after 500 shots the RMS error was computed. Finally, the time spent by the system running on more computers to achieve the same RMS error was measured. The next graph shows the results. The efficiency based on 500 shots is also plotted, so these two approaches can be compared.



Graph 5: Efficiency based on the number of shots and the RMS error

But one must remember that the same RMS error does not ensure the same picture quality. A similar problem was also discussed with overshooting technique [8]. Because the parallel version must do more radiosity shots to achieve the same RMS error, it produces a pictures with better visual quality. Thus "real" efficiency lies somewhere between these two curves.

6. Conclusions

A progressive refinement radiosity method with overshooting and run-time adaptive surface subdivision has been parallelized. The whole system was tested on a cluster of 8 workstations. For such parallel hardware there was achieved speedup up to 7.2 according to the scene character and used hemicube resolution when speedup was based on the number of radiosity shots. Speedup based on the RMS error decreases to 5.8.

7. References

[1] Baum D.R., Winget J.M.: *Real Time Radiosity Though Parallel Processing and Hardware Acceleration*, Computer Graphic (Proccedings 1990 Symposium on Interactive 3D Graphics), Vol.24, No.2, pp. 67-75, March 1990

[2] Bohn Ch.A., Garmann R.: *A Parallel Approach to Hierarchical Radiosity*, Proceeding of The Third International Conference in Central Europe on Computer Graphics and Visualisation 95, Plzeň February 1995, pp. 26-35

[3] Cohen F.M., Greenberg D.P.: *The Hemicube: A radiosity solution for complex environment*, Computer Graphic (SIGGRAPH '85 Proceedings), Vol. 19, No.3, pp. 31-40, August 1985

[4] Cohen F.M., Greenberg D.P., Immel D.S., Brock P.J.: *An efficient radiosity approach for realistic image sybthesis*. IEEE Computer Graphics and Applications 6(2), pp.26-35, March 1986

[5] Cohen M.F., Chen S.E., Wallace J.R., Greenberg D.P.: *A Progressive Refinement Approach to Fast Radiosity Image Generation*. Computer Graphics (SIGGRAPH '88 Proceedings), Vol. 22, No.4, pp. 75-84, August 1988

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

[7] Feda M., Purgathofer W.: *Progressive Refinement Radiosity on a Transputer Network*, Proceedings of the Eurographics Workshop on Rendering, Barcelona 1991

[8] Feda M., Purgathofer W.: *Accelerating Radiosity by Overshooting*, Proceeding of the 3rd Eurographics Workshop on Rendering, Bristol May 1992, UK, pp. 21-32

[9] Feda M.: *Speeding Up Progressive Radiosity by Overshooting*, Proceeding of The Third International Conference in Central Europe on Computer Graphics and Visualisation 95, Plzeň February 1995, pp. 87-92

[10] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V.: *PVM 3 User's guide and reference manual*, pvm@msr.epm.ornl.gov, May 1993

[11] Immel D.S., Cohen M.F., Greenberg D.P.: *A Radiosity for Non-Diffuse environments*, Computer Graphics (SIGGRAPH '86 Proceedings), Vol.20, No.4, August 1986, pp. 133-142

[12] Recker R.J., George D.W., Greenberg D.P.: *Acceleration Techniques for Progressive Refinement Radiosity*. Computer Graphics (Proccedings 1990 Symposium on Interactive 3D Graphics), Vol.24, No.2, pp. 59-64, March 1990

[13] Šindlar L.: *Paralelní algoritmy počítačové grafiky*, diploma thesis, MFF UK Praha, 1995, Internet WWW http://sun3.ms.mff.cuni.cz/thesis/sindlar

[14] Wallace J.R., Elmquist K.A, Haines E.A.: *A Ray Tracing Algorithm for Progressive Radiosity*, Computer Graphics (SIGGRAPH '89 Proceedings). Vol.23, No.3, pp. 315-324, July 1989