Parallel Visibility Computations for Parallel Radiosity

by

W. Stürzlinger and C. Wild

Johannes Kepler University of Linz
Institute for Computer Science
Department for Graphics and parallel Processing
Altenbergerstraße 69, A-4040 Linz, Austria, Europe

Tel.: +43(732)2468-884, Fax : +43(732)2468-10
e-mail: wrzl@gup.uni-linz.ac.at

Parallel Visibility Computations for Parallel Radiosity

W. Stürzlinger and C. Wild

Institute for Computer Science, Johannes Kepler University of Linz, Austria

Abstract

The radiosity method models the interaction of light between diffuse reflecting surfaces, thereby accurately predicting global illumination effects. Due to the high computational effort to calculate the transfer of light between surfaces and the memory requirements for the scene description, a distributed, parallelized version of the algorithm is needed for scenes consisting of thousands of surfaces. We present a distributed, parallel progressive radiosity algorithm. Then we parallelize the visibility calculations and analyze the results.

1 Introduction

Radiosity has become a popular method for image synthesis due to its ability to generate images of high realism. It was first introduced to computer graphics by Goral et al. [GORA84]. Further research resulted in the progressive refinement method, which quickly produces good approximations of the final solution [COHE88]. The radiosity method was extended to include specular reflection through the so called two-pass approach. For recent developements see [MALL88, SILL89, SILL91].

Common to all these methods is the representation of the surfaces of the environment by a mesh of quadrilaterals and triangles. These "patches" are used to store the radiosity on the respective part of the surface.

A formfactor is a value describing the "influence" of two patches onto each other. These formfactors were first calculated by the use of a hemicube [COHE85]. A hemicube is placed around the center of a patch and all other patches are projected onto its surfaces. The projected area gives an estimate for the formfactor between the patches.

Because this estimate of the formfactors may be inexact even for simple cases [BAUM89], other methods for computing the formfactors were suggested and/or implemented. Baum used a hybrid method involving both numerical and analytic methods [BAUM89] and other methods use raytracing to compute the formfactors [WALL89, SILL89, MALL88, TAMP91]. Wallace subdivides the shooting patch (i.e. the lightsource) until an analytic solution can be used to approximate the (delta-)formfactors of the respective delta-areas. Then the formfactors for all visible delta-areas are summed up giving a good approximation to the formfactor of the shooting patch.

The calculation of the formfactors accounts for most of the computation time of the radiosity method (e.g. 70 - 95 %). Also the memory requirement is clearly a function of the number of patches. These problems led to the developement of parallel implementations of the progressive refinement radiosity method [BAUM90, RECK90, FEDA91, CHAL91].

1.1 Progressive Refinement

The radiosity method partitions the surfaces of the scene in small, flat patches and computes the illumination for each of those patches. The radiosity of a patch is determined by the radiosity it emits directly plus all light that is reflected. This is described by the radiosity equation:

$$B_i = E_i + \rho_i \sum_{j=0}^{n-1} F_{i,j} B_j \tag{1}$$

where

- n is the number of the patches.
- B_i is the radiosity of the i-th patch.
- E_i is the emitted radiosity of the i-th patch.
- ρ_i is the reflectivity of the i-th patch.
- $F_{i,j}$ is the formfactor from patch i to patch j.

The linear equation system defined above is diagonally dominant, not symetric and not sparse. It can be solved with e.g. an iterative Gauss-Seidel solution method and this method converges quickly in practice. As the memory reqirement for the formfactor matrix is proportional to n^2 this method becomes impractical for larger n. Also the computational effort to compute all $F_{i,j}$ becomes prohibitively large.

The progressive refinement method solves this equation system iteratively also. But due to a reordering of the solution process, it is only neccesary to calculate (and store) one column of the matrix per iteration step. This method has the following illiustrative interpretation. An iteration step distributes ("shoots") the radiosity of the patch with the maximum unshot radioisity to all other patches in the environment. The advantage of this method is that we can display the result after each iteration step, and therefore give the user feedback how the resulting picture is going to look approximatively.

2 Parallelization of the Progressive Refinement Method

2.1 Previous Research

Parallelization of the Progressive Refinement Method has been attempted in several ways. Baum [BAUM90] used a multiprocessor workstation calculating the hemicubes using a hardware z-buffer. Recker [RECK90] used a cluster of workstations.

Feda [FEDA91] presented an implementation on a transputer network, where each processor has local memory. The formfactors were calculated using the hemicube method. Chalmers [CHAL91] also presented an implementation on a transputer network. He improved the accuracy of the formfactor calculation by using the analytical approch described by [BAUM89].

The use of the hemicube or the analytical method still suffers from the problem that the formfactors and the visibilities are determined using the "shooting"-patch as projection center. This leads to noticeable artifacts. A better method is to calculate this information directly for each receiver.

In the following sections we assume that we have a number of processors with local memory and an interconnecting network.

2.2 Parallel Progressive Refinement

This paper presents an approach based on the calculation of formfactors by raytracing as described by Wallace [WALL89]. Raytracing is used to determine the visible parts of the "shooting"-patch as seen from each patch. The formfactor of these visible parts is then calculated using the analytical solution to the contour integtral. The visibility is determined by subdividing the "shooting"-patch regularily into M parts and tracing a ray from the receiver to each of these parts.

We distribute the patches evenly among the N processors, and each processor computes the radiosity for it's $\frac{n}{N}$ patches. But for the visibility computation we have to store the complete set of patch geometries on each processor also. Clearly the maximum number of patches this algorithm is able to handle is then limited by the available memory on each processor. In section 3 we will show how this algorithm can be extended for larger numbers of patches.

The following steps are performed until the solution has converged:

- All processors send their "best" patch to the master processor, i.e. the patch which has the most unshot radiosity.
- The master processor chooses the globally best patch as "shooter" and sends this information to all processors. Note that the "shooter"-geometry and the associated radiosity values are distributed to all processors in this step also.
- The following steps are performed on all processors in parallel:
  - For each receiving patch we determine the visibility of the "shooting"-patch by tracing rays to the shooter. Each ray is intersected with all other patches. For the visibile parts we calculate the geometric formfactor and add the resulting contribution to the receiver radiosity.

In contrast to many previously presented algorithms, we do not have to update radiosity values on other processors. The only communication overhear is generated by the selection of the "shooter"-patch once every iteration step.

2.3 Rendering

After the solution has converged we render the scene. This can be done e.g. by sending all patches to the workstation and using its hardware z-buffer to render the picture. We did not address the problem of parallel rendering or rendering the solution after each iteration step, as we feel that the communication costs are too high.

Parallel rendering could be done as follows: We partition the picture into N parts and assign each part to one of the N processors, then we distribute all patches to all processors and every processor renders the patches into it's z-buffer. The load distribution will be quite uneven for this method, as different parts of the picture will "contain" significantly different numbers of patches. This has been reported by Feda [FEDA91] but they also propose a scheme which assigns every N-th scanline to a processor thus distributing the load much more evenly.

We used the following scheme: After every p-th iteration (p is user selectable), we send all patches to the workstation, which uses it's hardware z-buffer to render the picture.

3 Parallel Visibility Calculation

In section 2.2 we presented a parallel progressive radiosity algorithm. The major shortcoming of this method is that the number of patches is limited by the available memory on each processor. This is due to the fact, that we have to intersect each visibility ray with all other patches to determine the visibility of the "shooter"-patch. Therefore all patches have to be stored on all processors. When there is not enough memory to store all patches on a processor one possible strategy is to retrieve the patch geometries from other processors (e.g. [FEDA91], [CHAL91]). But the results are not satisfactory due to the big amount of data, which has to be transfered.

Instead of transfering patches to compute visibility, we transfer the visibility information instead. We assume in the following, that we use a fixed subdivision of the "shooter"-patch, into M (e.g. 16 or 64) parts. For every part we determine the visibility by tracing a ray to it's center. The ratio of all visibile parts to the total number of parts gives an approximation to the visiblity of the "shooter"-patch from a certain point. Assume further that we store the visibility information for the "shooter"-patch in a bit vector of length M.

If we distribute the patches of the scene evenly onto two processors we can calculate the visibility information of the "shooter"-patch in parallel. Every processor calculates the visibility of the "shooter"-patch using its part of the patches of the scene. The binary AND operation of two visibility vectors gives then the combined visibility information for the combination of the respective patches. I.e. if a certain part of the "shooter"-patch is visibile on both processors, it is also visible with respect to the union of the patch sets. If a part is invisible on one or both processors, it is invisible with respect to the union too.

Using this method we are able to compute the visibility information for the algorithm given in section 2.2 in parallel. Assume again that all patches are distributed evenly among the N processors and that the processors are arranged in a ring topology which will be used as a pipeline for computing visibility information.

A processor which wants to compute the visibility information of the current "shooter"-patch from a sampling point computes its visibility vector and transfers the point and its associated visibility vector to the next processor. This processor generates the appropriate rays and intersects them with its set of patches and computes the local visibility vector. Then it performs a binary AND of the local and the received visibility vector and sends the result on to the next processor.

In the beginning each processor generates its packets of sampling points and sends them to the next processor in the ring. Then it waits for incoming sample packets from the ring pipeline and processes these in turn.

When a packet returns to the original processor we count the bits, which are set, and use this to compute the (approximate) visibility of the "shooter"-patch and also the respective formfactor. To reduce the communication overhead, we transfer the points and the associated visibility vectors, in blocks of e.g. 200 between processors.

3.1 Optimized Parallel Visibility Calculations

The method presented in the previous chapter has the drawback that one packet has to be transfered around the complete ring of N processors to compute the visibility information for the respective points.

As the patches are distributed evenly among the N processors every processor has (approximately) the same amount of free memory. To use this memory we organize the processors

into R seperate rings and distribute the patch geometries in each ring. Then a processor holds the patches it computes radiosities for and additionally a set of patch geometries for visibility computations. I.e. with N processors organized into R rings we store the scene $R+1$ times. Once distributed across all processors (for the radiosity computations) and R times distributed across each ring (for the visibility computations).

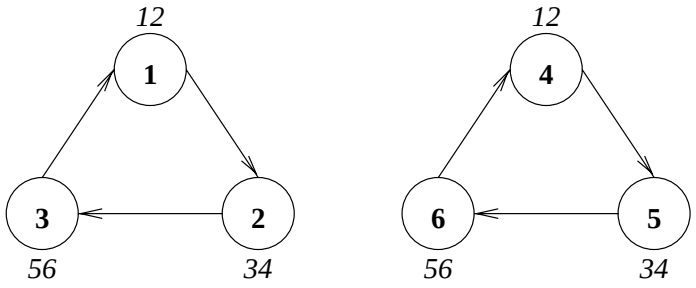

Figure 1: Sample patch assignements for 2 rings with 3 processors each.

In figure 1 an example patch assignment for a scene consisting of 6 patches onto 2 rings with 3 processors each is shown. The bold numbers denote patches for which radiosity is calculated and the italic numbers denote patch geometries stored for visibility caluclations.

Using this method each packet of sampling points has to be transfered only around a ring of $\frac{N}{R}$ processors. This speeds the algorithm significantly up.

Clearly the maximum number of rings R is dependent on the number of patches in the scene. If we can store all patches on one processor the scheme degenerates to the algorithm of section 2.2. If we have a huge number of patches, and are only able to store $\frac{n}{N}$ patches on one processor, we degenerate to the algorithm of section 3. In practice we will use the maximum number of rings we are able to allocate for a given scene, as this gives the best performance.

3.2 Adapting the Precision of the Visibility Computation

Using a fixed subdivision of the "shooter"-patch has the disadvantage that all visibilities are computed with the same precision. When e.g. the maximum of the energy which may be transfered between "shooter"-patch and sampling point is small there is no need for a precise formfactor.

If we calculate the geometric formfactor and the maximum energy transfered between the sampling point and the "shooter"-patch in advance we obtain a measure on how precise we need to calculate the visibility. Using this we can calculate how many rays we have to send to the "shooter"-patch and therefore deceide which subdivsion of the "shooter"-patch to use. This information is added easiliy to the packets of sampling points.

3.3 Dynamic Load Balancing

One problem of the parallel visibility computation method is that the load will be distributed unevenly among the processor, due to the different "visibilities" of different parts of the scene. When using an efficiency scheme (e.g. bounding volumes) to speed up the ray casting this effect will be much more noticeable. E.g. the walls of a room will be tested more often against

a ray compared to a small complicated object inside the room. A lot of rays will miss the bounding volume of the small object and therefore potential timeconsuming intersections will not be performed.

In each ring we measure how much time each of the processors took to calculate visibilities. When one processor has a significant higher load than the other processors we can redistribute a part of its patch geometries to other, less loaded, processors in the ring, therefore "lightening" it's load. This redistribution is done before the next "shooter"-patch is selected.

4 Patches and Elements

If the patches are too big, we will see jagged shadow boundaries and other artifacts. One solution is to use smaller patches, therefore increasing their number, which increases the memory and cpu time consumption. But for the "shooting" of radiosity we can safely use bigger patches.

The two-level hierarchy of patches proposed by [COHE86] is a very good means to overcome this problem. Each patch is subdivided into elements. The radiosity is then computed for all elements, and these elements are used also for display purposes. The average of the element radiosities is used as the patch radiosity for the "shooting" operation. This method proved to give satisfactory results in practice.

As the formfactor of the "shooting" patch is now needed for every element we have to compute more visibilities, therefore we will have to communicate more frequently. For the visibility computation we still have to store only the patch geometry for the intersection tests.

Additional communication overhead will also appear during the rendering phase, as all elements have to be transfered (and rendered).

For a better load distribution we distribute the patches so that an approximately equal number of elements has to be handled by each radiosity processor. Also we distribute the patches randomly among the processors to avoid situations where a processor has only patches of e.g. an unlit part of the scene.

5 Implementation and Results

Our approach was implemented on an nCube2S with 256 processors, which is a distributed memory computer where the nodes are connected with a hypercube topology network. The performance of a single processor is around 3 MFLOPS.

As a basis we used a progressive radiosity program for patches and elements [STÜ93] (sections 1.1, 4). We parallelized the algorithm as desribed in section 2.2 and implemented the parallel visibility calculation method (section 3). For practical reasons we used the first radiosity processor as master processor. The ray casting was done using a BSP-tree to speed the intersection test (e.g. [SUNG92]).

In the following tables N denotes the number of processors used and R denotes the number of rings used for the visibility calculations. We used a scene describing a simple room consisting of 4738 patches with a total of 17251 elements. Due to the memory constraints at least 4 processors are needed to store the scene. All times are given in seconds for the first iteration of the algorithm to complete (i.e. one "shooting"-operation).

Initially we implemented the primitive parallel visibility computation method (section 3) which uses only one ring.

N	R	t
4	1	158
8	1	152
16	1	105
32	1	54
64	1	45
128	1	38
256	1	37

The variations in the timings are due to the different distributions of the patches for the visibility calculations.

Then we tested the program using the maximum number of rings possible. I.e. for this particular scene every ring consisted of 4 processors.

N	R	t
4	1	158
8	2	78
16	4	65
32	8	36
64	16	17
128	32	13
256	64	7

The last result (256 processors: 7 seconds) shows that we can perform radiosity iterations at near interactive speed. As the processors of the nCube are quite slow we believe that this method will provide interactive speed on a machine of the latest generation.

Analyzing detailed timings further we found that the time a processor is busy depends also on the dimension of the rings. The values given in the next table describe how long the slowest and fastest processor were busy, and the ratio.

N	R	slowest	fastest	ratio
64	1	45	29	0.644
64	2	32	15	0.468
64	4	27	11	0.407
64	8	24	8	0.333
64	16	17	5	0.294

This behaviour result from slow visibility computations on one processor blocking the whole ring.

Analyzing the processor loads for a complete picture we found that the pattern of uneven load distributions is consistent for almost all iterations. I.e. the slowest processor for one iteration is also almost certainly the slowest processor for all other iterations, and so forth. The reason for this behaviour was outlined in section 3.3.

Preliminary results with the load balancing method presented in section 3.3 showed that we can increase the ratio between the slowest and fastest processor to about 0.8 for subsequent iterations. The improved load balance results also in further speedups for the total completion time.

6 Conclusion and Further Extensions

The newly presented method has a number of advantages.

- Formfactor calculation is done by raytracing, which proved to deliver much more accurate results than the hemicube method.
- There is no need to distribute radiosity values after formfactor calculation, as in previous parallelization attempts. All needed calculations are done locally, except for the visibility tests, and the shooter selection.
- The parallel visibility calculation method allows us to render scenes where the memory to store the patches and elements exceeds the available memory per processor node at reasonable times.
- The method allows adaptive subdivision of the surfaces. Each radiosity processor can decide locally if an element is not uniformly lit and subdivide it accordingly. The uneven load which may result from the differing number of element on the processors can be redistributed by transfering some patches and their elements and radiosity values to other processors. The strategy of transfering patches and the associated elements to other processors can also be used if we run out of memory while subdividing adaptively.

Also we are looking into applying the parallel visibility computation method to other problems.

Finally the incooperation of a discontinuity meshing algorithm look feasible and should improve the quality of the results further.

Acknowledgements

The authors thank J. Volkert for helpful discussions about parallel algorithms, and E. Spiegl for her good suggestions.

References

[BAUM89] Daniel R. Baum, Holly E. Rushmeier, James M. Winget, *"Improving Radiosity Solutions through the Use of Analytically Determined Form-Factors"*, Computer Graphics (SIGGRAPH '89 Proceedings), July 1989.

[BAUM90] Daniel R. Baum, James M. Winget, *"Real Time Radiosity Through Parllel Processing and Hardware Acceleration"*, Computer Graphics (SIGGRAPH '90), July 1990.

[CHAL91] Alan G. Chalmers, Derek J. Paddon, *"Parallel Processing of Progressive Refinement Radiosity Methods"*, in Proceedings of the Second Eurographics Workshop on Rendering, May 1991.

[COHE85] Michael Cohen, Donald P. Greenberg, *"The Hemi-Cube: A Radiosity Solution for Complex Environments"*, Computer Graphics (SIGGRAPH '85 Proceedings), August 1985.

[COHE86] Michael Cohen, Donald P. Greenberg, Dave S. Immel, Phillip J. Brock, *"An Efficient Radiosity Approach for Realistic Image Synthesis"*, IEEE Computer Graphics and Applications, March 1986.

[COHE88] Michael Cohen, Shenchang Eric Chen, John R. Wallace, Donald P. Greenberg, *"A Progressive Refinement Approach to Fast Radiosity Image Generation"*, Computer Graphics (SIGGRAPH '88 Proceedings), August 1988.

[FEDA91] Martin Feda, Werner Purgathofer, *"Progressive Refinement Radiosity on a Transputer Network"*, in Proceedings of the Second Eurographics Workshop on Rendering, May 1991.

[GORA84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, Bennett Battaile, *"Modelling the Interaction of Light Between Diffuse Surfaces"*, Computer Graphics (SIGGRAPH '84 Proceedings), July 1984.

[MALL88] Thomas J.V. Malley, *"A Shading Method for Computer Generated Images"*, Master's Thesis, University of Utah, June 1988.

[RECK90] Rodney J. Recker, David W. George, Donald P. Greenberg, *"Acceleration technique for Progressive Refinement Radiosity"*, Computer Graphics (SIGGRAPH '90), July 1990.

[SILL89] Francois Sillion, Claude Puech, *"A General Two-Pass Method Integrating Specular and Diffuse Reflection"*, Computer Graphics (SIGGRAPH '89 Proceedings), July 1989.

[SILL91] Francois X. Sillion, James R. Arvo, Stephen H. Westin, Donald P. Greenberg, *"A Global Illumination Solution for General Reflectance Distributions"*, Computer Graphics (SIGGRAPH '91 Proceedings), July 1991.

[SUNG92] Kelvin Sung, Peter Shirley, *"Ray Tracing with the BSP Tree"*, Graphics Gems III, Academic Press, 1992.

[STÜ93] W. Stürzlinger, *"FIRE - Fast Illumination Rendering Environment"*, Technical Report, Institute for Computer Science, University of Linz, Austria, December 1993.

[TAMP91] F. Tampieri, D. Lischinski, *"The Constant Radiosity Assumption Syndrome"*, in Proceedings of the Second Eurographics Workshop on Rendering, May 1991.

[WALL89] John R. Wallace, Kells A. Elmquist, Eric A. Haines, *"A Ray Tracing Algorithm for Progressive Radiosity"*, Computer Graphics (SIGGRAPH '89 Proceedings), July 1989.