ELSEVIER

Parallel Computing 23 (1997) 899–913

PARALLEL COMPUTING

Fast massively parallel progressive radiosity on the MP-1

Christophe Renaud *, François Rousselle [1]

Laboratoire d'Informatique du Littoral, BP 719, 62228 Calais Cedex, France

Received 1 June 1996; revised 25 November 1996

Abstract

Radiosity is a powerful method for solving the global illumination problem in the case of purely diffuse light reflexions. The progressive refinement algorithm provides interactivity during computation by displaying intermediate images, and overshooting methods increases the convergence rate of progressive radiosity. However, computation times remain very important. Parallelising these algorithms is a good way to significantly improve interactivity by reducing computation time. The aim of this paper is to present a method for the parallelisation of the progressive refinement radiosity algorithm on a massively parallel SIMD machine. We took care of both the SIMD machine nature and the high number of available processors on studying the several ways to efficiently implement the algorithm. The parallel scheme we propose uses a disk projection area for form factors estimate and decreases dramatically the computation times.

Keywords: Progressive radiosity; Form factor calculation; Parallel rendering

1. Introduction

Since it has been introduced in 1984 [10], the radiosity method has been proved to be an efficient rendering technique, solving the global illumination problem in the case of perfectly diffuse surfaces. But it required to compute and to store a large set of form factors before rendering could occur. Cohen introduced interactivity and low cost storage by using a progressive refinement radiosity approach [5], reducing the waiting period between illumination and rendering. However this algorithm requires much more

* Corresponding author. E-mail: renaud@lil.univ-littoral.fr.

[1] E-mail: roussell@lil.univ-littoral.fr.

iterations in order to converge to the final illumination solution. Recently, overshooting methods [8,25] have been introduced in order to accelerate the resolution convergence. But even with these new improvements, radiosity is still computationally demanding, especially because form factors estimate requires a lot of computation. These computational requirements are not available on sequential machines. Several works have thus been performed in order to implement the radiosity algorithms on parallel machines. In this paper, we are interested in implementing a massively parallel approach for the progressive refinement radiosity algorithm on a SIMD mesh-connected machine, the MP-1. We focused our attention on the parallelisation of the form factors computation step. Form factors are computed using a disk projection area, directly based on the Nusselt equivalent, and uniformly discretised into square proxels. Furthermore, our approach takes advantage of overshooting too, as the existing overshooting methods can be efficiently implemented as a part of our SIMD algorithm. As the MP-1 computer uses a SIMD control, we take care to use optimally all the processors. This allows us to achieve well balance of each stage of the algorithm.

In the next part, we present the disk projection algorithm we used for the form factors estimate. Then we recall and analyse the main works in radiosity parallelisation in part 3, and we present the architecture of the MP-1 machine we used for our studies in part 4. In the last two parts, we present our parallel algorithm, and some results we have obtained.

2. Disk projection area

Form factors estimate requires to compute visibility between each pair of elements. Several algorithms have been developed for this purpose. They use either projective (e.g. hemicube [4], single projection plane [14,19]) or ray casting approaches [24]. Projective approaches have several advantages with regard to ray-casting for a SIMD parallelisation. They involve identical computation for each element (and each proxel of the projection area), when computation are different for rays according to the intersected surface type. Efficiency can only be obtained on SIMD architectures when processors are well load balanced and when they apply simultaneously the same simple computation. In order to achieve efficiency, we chose to implement a projective algorithm, based on the Nusselt equivalent.

2.1. Principle

The disk projection area algorithm is based on the Nusselt equivalent, and has been first proposed by Spencer [21]. A unit disk is applied on an element, tessellated into several square proxels, and scene elements are projected onto these proxels through theoretically 2 projections: the first one projects an element onto the hemisphere sustained by the disk; this projection is then projected orthogonally onto the disk surface (see Fig. 1a).

The disk projection algorithm has several advantages as compared to the other projective algorithms:

- Only one projection surface is used compared to the hemicube for which 5 projection surfaces (and consequently 5 projection steps) are required.

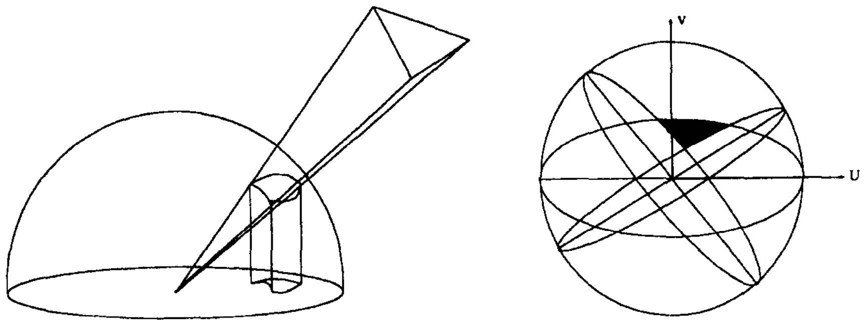

Fig. 1. (a) Nusselt equivalent, (b) elliptic arcs based outlines

· The disk projection surface lies onto the element, and it does not 'forget' the grazing directions like the single plane projection approaches do.

· The elementary form-factor connected to each proxel is the same for each one. This allows the disk to reduce the number of proxels to use, and to cancel out the oversampling that appears in each other projective approaches (for more details about projective approaches comparisons, see [16]).

One problem with this algorithm is that the projection outlines are not linear: the image of a 3D edge on the disk is an elliptic arc (see Fig. 1b). Commonly used outline filling algorithms are thus not available. In [9], Goldfeather proposed a method to fill such elliptic arcs based boundary, for the Pixel-Planes 5 machine. We derived a simpler solution from his work in [15], which is summarized below.

2.2. The filling algorithm

We suppose first that the proxels coordinates are expressed in an (u, v) orthonormal coordinate system, like the one that appears in Fig. 1b. In this coordinate system, each ellipse equation can be rewritten as

$$Au + Bv + C\sqrt{1 - u^2 - v^2} = 0.$$

An N-edges element projection is thus represented as an intersection of the N ellipses. By appropriately orienting each ellipse equation, it is then easy to know whether a proxel is inside or outside the projection. In order to reduce the number of proxels to test, the projection bounding rectangle of the projected outline is computed and only the proxels inside this disk subarea are taken into account (see [15] for more details).

3. Previous work

3.1. Parallel progressive radiosity

The progressive radiosity approach offers three different levels for parallelisation:

· Several emissions are computed in the same time, each one from a different emitting element, according to the number of available processors [2,3,7,12,14].

· Only one emission is computed, by distributing the involved computations between the processors [20].

· The processors collaborate for computing the form factor between the shooting element and a given element [1]. Note however that this approach only occurs for projective methods.

All these levels have been exploited both on MIMD and SIMD architecture. MIMD parallelisation has mainly focused on the first two levels. Differences between implementations proceed from the form factor algorithm, the network topology and the duplication/distribution/share schemes of the database. It appears however that as the database size grows, the elements have to be distributed between the processors. This distribution involves a very high number of communications, and the efficiency of these approaches decreases rapidly when the number of processors increases [13].

3.2. Overview of SIMD parallel radiosity

SIMD machines require all the processors to perform the same instruction at the same time. This centralized control requires to use data parallelism, and to take care of the load-balancing between the processors. The processors are generally organized as an array, which is close to the projection plane organisation used in projective methods. Consequently, several approaches have been proposed for this kind of form factors computation algorithms. In [16] we proposed to exploit the third level of parallelism we described, by projecting successively each element onto the sampling surface. Each proxel is handled by a processor, which computes whether the projected element is inside or outside the proxel, and applies the depth-buffer operations. However, a large number of proxels are not covered by the projected element, and the efficiency of such an approach is small. Varshney [23] proposed to simultaneously compute several element projections. Each PE manages a block of neighbouring proxels, and a part of the element database. The first step of its algorithm distributes the elements to the processor managing a part of the sampling area where the elements have to be projected. Then all the processors determine in which of their proxels the elements they received are visible, and apply the depth comparisons. However, the projection work is not distributed equally between the processors, as the number of elements that are visible in each sampling direction can be very different, and imbalances occurs.

Parallel ray casting for form factor computation has been implemented too. Drucker [6] used a processor allocation technique for computing all the possible intersections between a ray and the voxels that lie on the ray way. Several rays are treated simultaneously, according to the number of available processors. However, unnecessary work is performed as all the intersections are computed along the ray path, even if an intersection exists in the first voxels. Then, the SIMD nature of the approach quickly decreases the performances when several objects types are intersected onto different processors.

4. The MP-1 architecture

The MP-1 computer is made up of a host workstation, a large number of simple processing elements (PEs), and a specialized processor, so called the array control unit

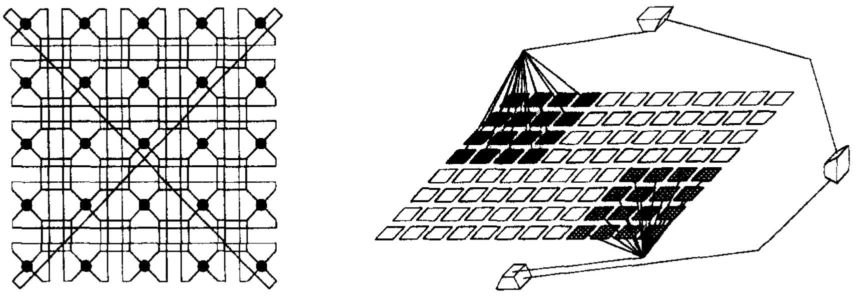
Fig. 2. (a) The Xnet, (b) the global router.

(ACU), which controls the set of PEs. The processing elements have their own local memory and run following a SIMD scheme: each PE performs simultaneously the same instruction onto its private data. The ACU broadcasts instructions and global data, while the local data are fetched from the PE memory. PEs can be disabled temporally, when they do not have to perform some instructions (for example when a *if* statement is broadcast).

The processing elements, which we will simply call the processors in the following of the paper, are build around a simple 4 bits architecture; they dispose of thirty two 32 bits registers, and up to 64 KB RAM. They are connected together according to a rectangular grid, its size varying from 1,024 (32×32 array) up to 16,384 (128×128 array) processors.

Two communication networks are provided, either for neighbouring or distant communications. The Xnet network connects each processor to its 8 nearest neighbours (see Fig. 2a). A processor lying on the edge of the grid is connected to the opposite edge processor. Communication through this network are SIMD controlled: all the processors must send their message in the same direction, or wait for an other communication step.

The Global Router is the second communication network of the MP-1. It is able to perform communications between every pair of processors, through a three stage hierarchy of crossbars (see Fig. 2b). However, only one link to the GR is available per cluster of 4x4 processors. Communication through the GR from or to a cluster are thus necessarily sequentialised. Consequently, conflicts (and then inefficiency) occur when several processors of a cluster have to send or to receive a message.

We have developed our parallel radiosity approach on a 16,384 processors MP-1, each processor managing 64 Kb RAM (1 Gb global memory).

5. A massively parallel approach

Our goal is to develop an efficient SIMD implementation of the disk projection algorithm as the heart of a massively parallel progressive radiosity approach. SIMD architectures are often characterized by a high number of very simple processors: the power is provided by the number of processors rather than by they computation capabilities. In the same way, processors have their own memory; this one is often of small size (a few Kbytes), but the use of several thousands of processors provides generally a very large amount of global memory. In order to benefit from this two unusual features, it is necessary on one hand to apply simultaneously the same operations onto all the processors and to assure that all these operations are really useful. On the other hand, the distribution of the data is necessary, but it allows the approach to process very large scenes.

5.1. Data 'distribution'

Two kinds of data are mainly handled by a radiosity algorithm, when using a projective approach: the elements (with both geometric and photometric features) and the proxels used for approximating the form factors. All these data have to be distributed over the processors, in order both to be directly available for the computation (no communication required for fetching the data) and to take advantage of the large amount of global memory available on the target machine.

Element distribution scheme is very simple, as no particular property is required for the form factor computation step: elements are extracted from the database file and send one by one to a different processor. When the number of elements is greater than the number of processors, the process is cyclically repeated, in order for each processor to finally manages N or $N-1$ elements.

Proxels distribution scheme is more subordinate to the MP-1 architecture. The proxel array is mapped cyclically onto the processor array, so that 2 neighbouring proxels (both along u and v axis) are managed by 2 neighbouring processors. This will allow some steps of the algorithm to be well load balanced, as described further.

5.2. The filling principle

Each element needs to be projected onto the disk, and a depth-buffer operation has to be performed in each inner proxel. However both the elements and proxels are distributed across the processor array. It is also impossible for a processor to apply the depth-buffer computation in each proxel covered by its element projection, since these proxels are managed by some other unknown processors.

We investigated a way to solve this problem by communicating a projected element to all the processors [15]. Thus each one is able to compute whether the element appears in one or more of the proxels it manages. But this solution involves a lot of useless computation, because projected elements are generally small and cover a few part of the entire proxel set. The main part of the processors is consequently used for useless computation (as compared to a sequential approach), and provides an inefficient approach.

In order to provide useful computation, it is necessary to compute, locally to an element handling processor, which proxels this element covers. This information can then be sent only to the processors that handle the covered proxels. However such an approach would provide an important drawback: a large amount of communication is required, as one message is necessary for each covered proxel. We propose to reduce this amount of messages by computing first only the covered spans (we call span a

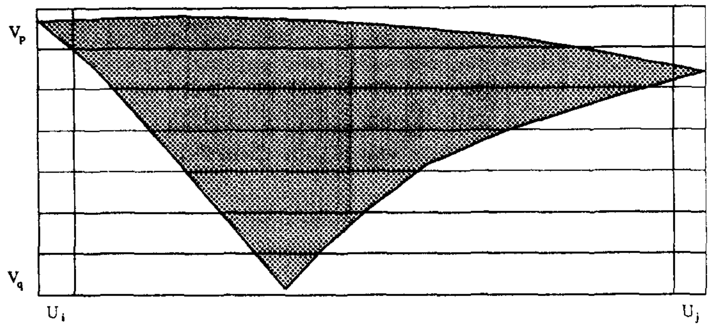

Fig. 3. Span conversion of the projected outlines.

continuous set of proxels lying on the same projection plane line). Each of this span is then sent to the processor handling its first proxel, which can apply the depth comparison. The depth-buffer for the other span proxels is then easy to perform, by remembering that proxels have been distributed cyclically. That means that the second proxel of a span is managed by the neighbouring processor of the one that manages the first span proxel, and so on. Communications are of course required, but they are efficiently performed by using the Xnet network, which is very well designed for proximity communications.

Computing spans is easy for linear convex outlines, but is more computationally demanding for elliptic arc based boundaries. For this reason the exact covered spans are not directly computed. Rather one can compute more easily the spans covered by the outline bounding rectangle, cutting this rectangle in equal length size spans (see Fig. 3). The inner test is then performed proxel per proxel just before the depth-buffer comparison, on the processor managing the proxel.

5.3. The parallel algorithm

By developing our approach on a SIMD architecture, we have to take care of each step of the algorithm in order to achieve load balancing and efficiency. For this purpose, the entire form factors computation process has been carefully studied and cut into several successive steps, each one being designed for a SIMD implementation.

5.3.1. Geometric transformation

The geometric transformations are first applied on the elements stored on each processor. These transformations include coordinate transformations, back-face culling and clipping. Note that the first two ones require exactly the same amount of computation, when clipping involves small unbalancing.

5.3.2. Spans conversion

As described before, projected elements are cut into spans: the bounding rectangle is first computed, and each of the covered row of the projection disk is then deduced. For each one, the first span proxel coordinates is stored into a local span list. Some other informations are stored too, like the length of the span and various parameters for the z-buffer step.

When a processor detects that its current element span conversion has been completed, it dynamically loads a new element from its local element list. This avoids the unbalance that would occur if it had to wait for the completion of some other processor.

5.3.3. Span passing

Computing the spans locally to each processor involves communications before applying the z-buffer. As explained in Section 5.2, the proxel covered by a span are generally managed by some other processor than the one managing the span itself. The span is thus sent to the processor managing its first proxel. This communication is performed through the Global Router, which is better-suited for connecting any pair of processor than the Xnet.

5.3.4. Z-buffer

After the previous step, each processor has a new span list, each of them beginning in a locally managed proxel. These spans are z-buffered by taking each one successively, and by applying the following operations:

- the first proxel of the span is extracted from the span;
- if this proxel is inside the projected outline, it is depth-buffered;
- the span length is decreased, and the span is sent to the right-neighbour processor;
- a span is received from the left-neighbour. If its length is null, a new span is fetch from the span list;
- the steps are applied again for the current span, until no more spans are stored on any processor.

Remember that this algorithm is applied simultaneously for all the processors. This allows us to depth-buffer a very high number of spans simultaneously. As proxels are distributed cyclically over the processor array, communications are very efficiently performed through the Xnet.

5.3.5. Form factors update

For the same reason than previously (covered proxels and elements are not managed by the same processor), communications are required in order for an element to known its form factor with the emitting element. These communications are performed through the Global Router, by sending each proxel associated elementary form factor to the corresponding element. As many proxels are generally covered by the same element, this involves both a lot of communications (one per covered proxel) and a large amount of conflicts (because several proxels are sent to the same processor). One can however reduce the communication time by studying proxel coherency properties:

- Several conflicts appear when many processors attempt to communicate with the same one. This problem appears frequently in projective approach, because the probability for 2 neighbouring proxels to be covered by the same element is very high. A simple 'blending' function has been implemented in order for 2 neighbouring processors to choose the proxels they have to send simultaneously. This choice is performed in such a way that the two processors do not send neighbouring proxels.

· The previous approach reduces the conflicts, but does not reduce the number of proxels that have to be sent. Using the horizontal coherency between the proxels (neighbour proxels are often covered by the same element), a collect step is performed along each disk line. After this step, the line form factor of each visible element has been accumulated in only one processor (the one managing the first visible proxel of the element). Only one communication is then performed for this new form factor.

These two solutions have been implemented in [17], and have dramatically reduced the communication time for this step of the algorithm.

5.4. Parallel overshooting

At each step of the progressive radiosity algorithm, when some radiosity is shot from an element i to the environment, a part of this energy returns to the emitter due to single or multiple reflections by elements j ($j \neq i$). The aim of overshooting's methods is to provide an estimate of the amount of this returned radiosity and to add it to the current unshot radiosity of the emitter before it shoots this one. The overall number of iterations required by the progressive radiosity algorithm is thus reduced.

Several overshooting methods have been proposed recently. Methods by Gortler et al. [11] and Shao et al. [18] compute precisely the radiosity reflected directly by other elements but do not take care of the multiple reflections. On the other hand, Feda and Purgathofer's method [8] estimates the radiosity due to all reflections (single and multiple) using a global ambient term.

In environments with large reflectivities, Feda and Purgathofer's method converges faster than the others. But in the case of small reflectivities, it might not converge as fast as Shao et al.'s and Gortler et al.'s methods because the returned radiosity due to multiple reflections is less significant. In such environments, Feda and Purgathofer's rough overshooting's estimate might be less accurate than Shao et al.'s and Gortler et al.'s computation.

Another approach described by Xu and Fussell [25] takes advantage of the latest ones. This method not only computes the radiosity reflected directly but estimates the radiosity due to multiple reflections too. This algorithm provides better results in environments with any reflectivities.

We have implemented Xu and Fussell's algorithm on the MP-1 taking advantages of its architecture. The amount of radiosity to shoot at each step of the algorithm $\Delta B'_i$ can be described as the sum of the unshot radiosity of the emitter ΔB_i with the radiosity due to the two kinds of reflection [25]

$$\Delta B'_i = \Delta B_i + \rho_i \left(\sum_{j=1}^{n} F_{ji} \Delta B_j + \rho_{\mathrm{ave}} \, Ambient \right).$$

In order to compute the radiosity due to direct reflections, $\sum_{j=1}^{n} F_{ji} \Delta B_j$, each processor first computes the product $F_{ji} \Delta B_j$ locally for each of the elements it manages. The sum of these products is then performed locally too. Finally, these partial results are quickly summed through a reduction operation.

The radiosity due to multiple reflections is expressed as $\rho_{\text{ave}} \mathit{Ambient}$, the *Ambient* term being described by

$$\mathit{Ambient} = R \sum_{j=1}^{n} \Delta B_j F_{*j},$$

where R is the overall interreflection factor

$$R = 1 + \rho_{\text{ave}} + \rho_{\text{ave}}^2 + \ldots = \frac{1}{1 - \rho_{\text{ave}}}$$

and ρ_{ave} is the average reflectivity of the environment

$$\rho_{\text{ave}} = \sum_{k=1}^{n} \rho_k A_k \bigg/ \sum_{k=1}^{n} A_k,$$

where A_k is the area of element k. The overall interreflection R and the average reflectivity ρ_{ave} are computed once at the beginning of the algorithm using reduction functions.

Finally, the form-factor F_{*j} is approximated by

$$F_{*j} \cong A_j \bigg/ \sum_{k=1}^{n} A_k$$

It is thus obvious that implementing Xu and Fussell's overshooting approach, which has been shown to reduce considerably the number of shooting steps, in our parallel algorithm does not have any significant overcost using the MP-1 machine.

6. Some results

We have applied our algorithm onto 4 different scenes, with increasing complexities and various illumination properties. Theoretically, the MP-1 is able to store about 10,000,000 elements (assuming about 100 bytes data per element), but the more complex scene we present (scene 4) is restrained to only 100,000 elements. A close-to 1 million elements scene is currently being designed. We present in Figs. 4 and 5 a view of these four scenes.

· In Fig. 4 top, a simple deskroom with few objects and only one source; in Fig. 4 bottom, a more complex deskroom including several objects and 4 sources. The first scene has been tessellated into about 8,000 elements and the second one into about 25,000 elements.

· Fig. 5 presents in the top part a classroom with a very large number of sources (windows, ceiling lighting sources); in the bottom part, a sports hall where sources are windows and ceil lighting. The classroom contains about 39,000 elements and the sports hall about 108,000 ones.

The average computation time per shooting step is presented in Table 1, for 3 different projection plane resolutions. Resolution N means that the bounding square of

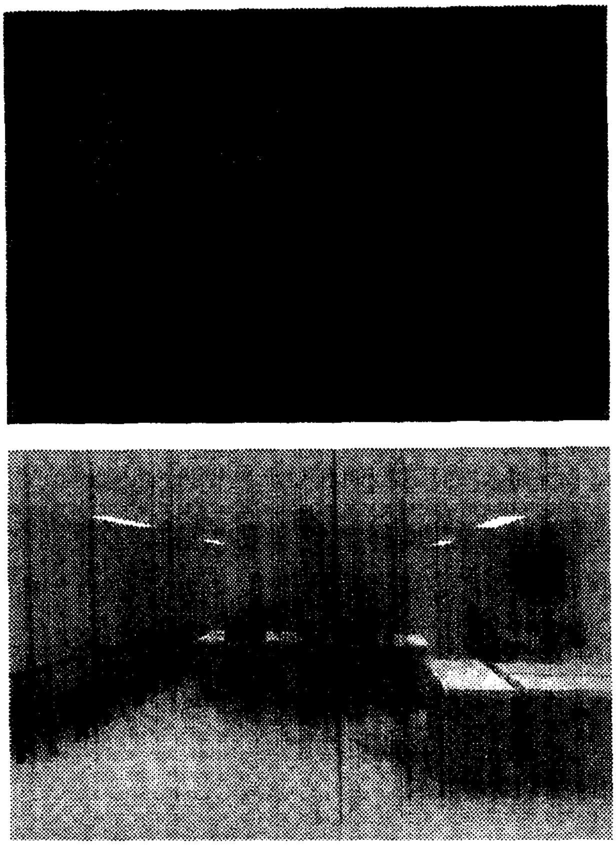

Fig. 4. Scenes 1 and 2: two deskrooms.

the projection disk has been cut into $N \times N$ proxels. Obviously only the proxels inside the disk are used for form factors estimate.

These results highlight the great performances of our approach, either for small or large databases. Note however that computation times are surprisingly similar both for simple or complex scenes, especially for high resolutions. This can be explained by taking into account the size of each scene, and the fact that the size of each element is almost the same for all the scenes. When elements of scene 1 are projected, they covers a large amount of proxels, because distances between objects are relatively small. Consequently, a lot of spans per element have to be computed, distributed and z-buffered. In case of scene 4, distances between the objects are large; the projected elements cover thus less proxels than for scene 1. Less spans per element have to be computed. But as the number of elements is greater in the more complex scene, global computation times are almost the same for each scene. It will be interesting to see what will happen in case of very large database.

It is uneasy to compute efficiency of the approach using the MP-1 architecture. Obviously it is impossible to run the algorithm on one processor. Furthermore, it is

Table 1
Average computation times per shooting step (s)

Resolution	Scene 1 8,000 elts	Scene 2 25,000 elts	Scene 3 39,000 elts	Scene 4 108,000 elts
256×256	0.296	0.431	0.551	0.673
512×512	0.524	0.583	0.692	0.821
1024×1024	1.183	1.125	1.175	1.293

uneasy to modify the architecture in order to have a smaller processor array. Thus we have simulated smaller processor grids, by disabling some PEs of the MP-1 during the computations. Two configurations have been tested: 1,024 processors (32 × 32) and 4,096 processors (64 × 64). The results of the simulation have shown a quiet linear speedup for the four scenes.

It is interesting to analyse the computation/communication cost of each step of the

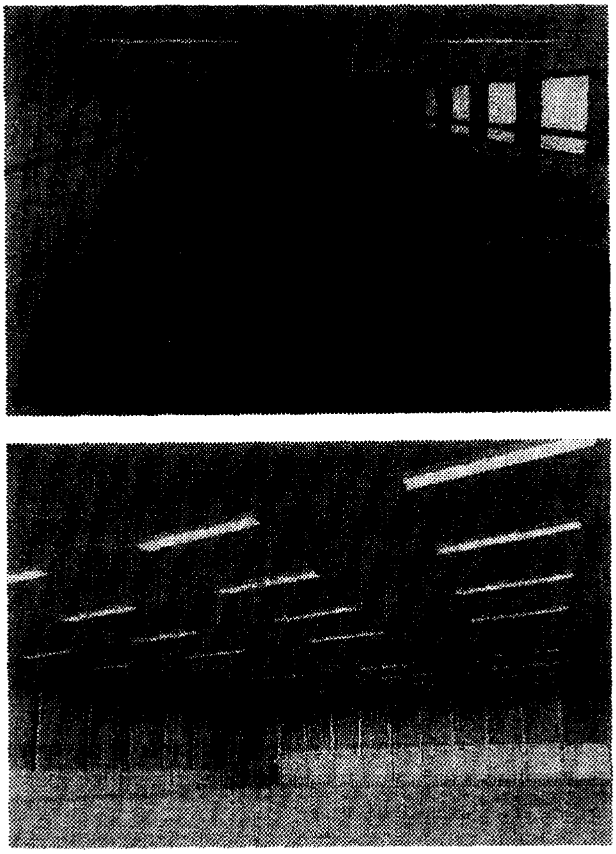
Fig. 5. Scenes 3 and 4: a classroom and a sports hall.

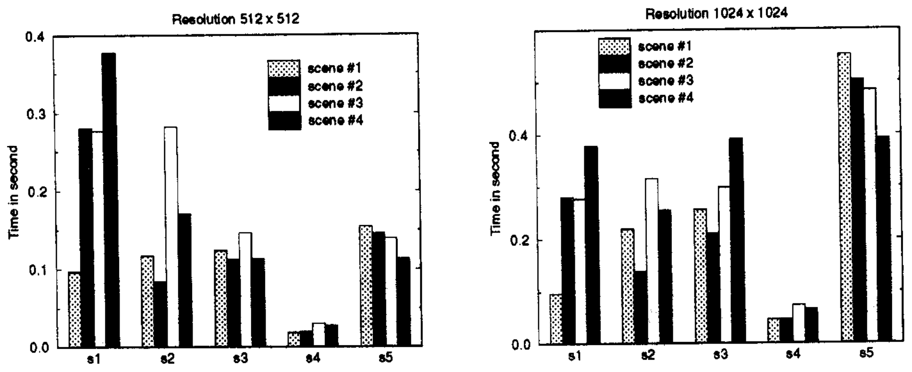

Fig. 6. Computation times distribution for two disk resolutions.

approach. For each of the five steps that we have previously described, we have measured their average computation time for each of the four scenes. These measures are presented in the two following histograms (see Fig. 6).

The steps appear with the notation S_1 for the geometric tranformations, S_2 for the spans computation, S_3 for the spans passing, S_4 for the depth-buffer and S_5 for the radiosity update. Obviously the time of step S_1 is the same, as it involves only computations on the elements. Both computation time for step S_2 and S_4, and communication time for step S_3 increase, as the number of spans increases when resolution grows. It appears that the form factor update time is smaller for complex scenes than for large ones. This can be explained by the use of the Global Router. When the scene is small, several proxels have to be sent to the same element. This generates many conflicts, because access are sequential. When the scene is very large, these conflicts decreases, in the same way the number of covered proxels per element decreases.

7. Conclusion and perspectives

We have presented in this paper a massively parallel approach for progressive radiosity. Form factors are estimated through the use of a disk sampling area, that reduces the number of proxels required for sampling, and avoids the loss of energy through the grazing directions. The parallel implementation of this algorithm has taken care of the SIMD features of the target architecture. Each algorithm step has been studied in order to provide SIMD-like code, providing a very high efficiency through the large set of processors. Furthermore, we have shown that recent overshooting methods can be easily added to our approach. Parallel form factors estimate and parallel overshooting allow our approach to provide fast progressive solving of the radiosity equations system, even for large databases. Despite our algorithm has been developed by intensively using MP-1 features, especially the communication networks, it seems to be made suitable to other SIMD architecture, subject to take into account their own communication ways.

Projective algorithms are prone to errors during form factors estimate. One of this error source, known as aliasing, is today investigated within the framework of our parallel approach. Aliasing occurs because of the discontinuities that appears during the sampling process. It can be reduced by increasing the number of samples (here the number of proxels), but this increases the computation time too. We are currently studying the adequacy of antialiasing techniques other than oversampling, that could be applied on the disk sampling method.

In our current approach, spans are computed from the projected outline bounding rectangle. This simplify the span conversion process, but involves more computation during the z-buffer step. We are studying whether computing the exact span would provide better results than those that have been presented.

Finally it seems important to still reduce the communication times. This can be achieved by considering coherency properties during span distribution and form factors computation.

References

[1] D.R. Baum, J.M. Winget, Real time radiosity through parallel processing and hardware acceleration, Comput. Graph. 25 (4) (1991) 51–60.
[2] A.G. Chalmers, D.J. Paddon, Parallel processing of progressive refinement radiosity methods, in: 2nd Eurographics Workshop on Rendering, Barcelona, May 1991.
[3] S.E. Chen, A progressive radiosity method and its implementation in a distributed processing environment, M.Sc. thesis, Cornell University, Ithaca, 1989.
[4] M.F. Cohen, D.P. Greenberg, The hemicube: a radiosity solution for complex environments, SIGGRAPH 85, pp. 31-40.
[5] M.F. Cohen, S.E. Chen, J.R. Wallace, D.P. Greenberg, A progressive refinement approach to fast radiosity image generation, SIGGRAPH 88, pp. 75–84.
[6] S.M. Drucker, P. Schröder, Fast radiosity using a data parallel architecture, in: 3rd Eurographics Workshop on Rendering, Bristol, 1992, pp. 247–258.
[7] M. Feda, W. Purgathofer, Progressive refinement radiosity on a transputer network, in: 2nd Eurographics Workshop on Rendering, Barcelona, 1991.
[8] M. Feda, W Purgathofer, Accelerating radiosity by overshooting, in: Proc. 3rd Eurographics Workshop on Rendering, 1992, pp. 21–32.
[9] J. Goldfeather, Progressive radiosity using hemispheres, Technical report TR 89-002, University of North Carolina at Chapel Hill.
[10] C.M. Goral, K.E. Torrance, D.P. Greenberg, Modeling the interaction of light between diffuse surfaces, SIGGRAPH 84, pp. 213–222.
[11] S. Gortler, M. Cohen and P. Slusallek, Radiosity and Relaxation Methods, Technical Report, Computer Science Department, Princeton University, 1993.
[12] P. Guitton, J. Roman, C. Schlick, Two parallel approaches for progressive radiosity, in: 2nd Eurographics Workshop on Rendering, Barcelona, 1991.
[13] P. Guitton, J. Roman, G. Subrenat, Implementation results and analysis of a parallel progressive radiosity, in: Proc. of IEEE/ACM 1995 Parallel Rendering Symp., 1995, pp. 31–38.
[14] R.J. Recker, D.W. George, D.P. Greenberg, Acceleration techniques for progressive refinement radiosity, Comput. Graph. 24 (2) (1990) 59–66.
[15] C. Renaud, F. Bricout, E. Leprêtre, Massively parallel hemispherical projection for progressive radiosity, Comput. Graph. 19 (2) (1995) 273–279.
[16] C. Renaud, Approches parallèles pour la radiosité, Ph.D. thesis, University of Lille, 1993.
[17] C. Renaud, F. Bricout, E. Leprêtre, An object parallel approach for radiosity on the MP-1, in: Int. Conf. on Massively Parallel Processing Applications and Development, Delft, 1994, pp. 887–894.

[18] M.Z. Shao, N.I. Badler, Analysis and acceleration of progressive refinement radiosity method, in: Proc. 4th Eurographics Workshop on Rendering, 1993.
[19] F. Sillion, C. Puech, A general two-pass method integrating specular and diffuse reflection, Comput. Graph. 23 (3) (1989) 335–344.
[20] D.B. Singh, S.G. Abraham, F.H. Westervelt, Computing radiosity solution on a high performances workstation LAN, in: 1st High Performance Distributed Computing, N.Y., 1992, pp. 248–257.
[21] S.N. Spencer, The hemisphere radiosity method: a tale of two algorithms, in: Eurographics Workshop on Photosimulation, realism and Physics in Computer Graphics, Rennes, 1990, pp. 127–135.
[22] T. Théoharis, I. Page, Parallel incremental polygon rendering on a SIMD processor array, Parallel Processing for Computer Vision and Display, pp. 329–337.
[23] A. Varshney, J.F. Prins, An environment-projection approach to radiosity for mesh connected computers, in: 3rd Eurographics Workshop on Rendering, Bristol, 1992, pp. 271–281.
[24] J.R. Wallace, K.A. Elmquist, E.A. Haines, A ray tracing algorithm for progressive radiosity, Comput. Graph. 23 (3) (1989) 315–324.
[25] W. Xu, D.S. Fussell, Constructing solvers for radiosity equation systems, in: Proc. of 5th Eurographics Workshop on Rendering.