1 An Environment-Projection Approach to Radiosity for Mesh-Connected Computers

A. Varshney
J.F. Prins

ABSTRACT
We describe a progressive refinement radiosity algorithm for highly-parallel mesh-connected SIMD or MIMD computers. The technique is based on environment-projection and scales easily to large machines and datasets. Form-factor computations can be performed using local communication by mapping the single-plane across the processor mesh. We report on the performance of an implementation on the MasPar MP-1, and discuss some potential improvements related to load balancing.

1.1 Introduction

In recent years, the radiosity method has gained widespread acceptance in the computation of view-independent global illumination for (primarily) architectural datasets. One of its important applications has been to provide realistic lighting for datasets of proposed buildings generated from architectural drawings. By navigating through one of these virtual buildings, the architectural design can be evaluated for usability, traffic, and aesthetic appeal [3]. Any necessary modifications, to remove shortcomings or to test new ideas, can be made and the design re-evaluated. One of the major bottlenecks in this design cycle is the computation-intensive radiosity process. With the increasing availability of parallel computers, attempts are being made to parallelize this process to achieve interactive rates on non-trivial datasets (of the order of thousands of polygons), thereby making the whole design process a more meaningful one.

In this paper we describe a radiosity algorithm for highly-parallel mesh-connected computers that can provide rapid progressive estimations of global illumination of large datasets. The organization of the remainder of this paper is as follows. We start with a brief overview of radiosity that introduces the terminology we will be using. Next we describe the parallelism available in the computation of radiosity and classify previous work done in parallel radiosity. Next we describe our approach and its implementation on a 4K processor MasPar MP-1. We conclude with a discussion of (as-yet unimplemented) potential improvements of our approach.

1.2 Radiosity Overview

Radiosity B_j for a surface j is defined as the total rate at which radiant energy leaves that surface in terms of energy per unit time and per unit area [15]. In an environment composed of n diffuse surfaces the radiosity of a surface j is given by

$$B_j = E_j + \rho_j \sum_{i=1}^{n} B_i F_{ji} \tag{1.1}$$

where E_j is the rate of direct energy emission from the surface j per unit time and per unit area, and ρ_j is the reflectivity of the surface j. The summation in the equation 1.1 represents the total flux incident on surface j from all other surfaces in the environment.

energy leaving surface i that is incident on surface j.

Considering all n surfaces, this yields a system of n linear equations in the n unknowns $B_1, .., B_n$ shown in Figure 1.1. The F_{ij} are determined from the geometry of the environment and the E_j, the emittances of the light sources in the environment, are assumed given.

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

FIGURE 1.1. The Radiosity System of Equations

The form-factor F_{ij} is equal to the fraction of the base of a unit-radius hemisphere centered on surface i that is covered by the projection of surface j on that hemisphere. The *projection-surface*, whether a hemisphere or its approximation - the hemi-cube [9], is subdivided into smaller elements, say pixels, for the purposes of sampling the environment. The *delta form-factor* Δf_q for a pixel q on the projection-surface is defined as the contribution of that pixel to the form-factor value. The magnitude of the contribution depends on pixel location. The total form-factor F_{ij} for surface i with respect to the surface j is the sum of all delta form-factors that are covered by the visible portion of surface j on the projection-surface.

Since the coefficient matrix is diagonally dominant[1], iterative methods for solving the system of equations in Figure 1.1, are preferred over direct methods such as the Gaussian elimination.

In an iterative method, a given variable B_i is updated based on the current approximation to the solution. This corresponds to a "gathering" of all the energy incident on the surface i. The method of progressive refinement [8] is a variant of the Gauss-Seidel iterative method that computes an update to *all* variables based on the current approximation to B_i. This corresponds to "shooting" all the energy from surface i. By choosing surfaces with high emittance first, this method provides good approximations for radiosity of all surfaces at early stages.

1.3 Parallel Approaches and Classification of Previous Work

The radiosity problem contains many opportunities to employ parallelism in the computation of its solution.

First, at the level of updating the approximate solution, we may employ a method other than the sequential Gauss-Seidel method of successive displacements. A simultaneous displacement method for solving the radiosity system of equations, such as the Jacobi method, permits all variables' updates to be computed in parallel. In a hybrid method a set of k out of a total of n variables can be chosen as the variables to be solved for in the current iteration. Each of these k variables is solved for in parallel using the values of the previous iteration for all variables. Then in the next iteration, these new values of k

[1]the participating surfaces are assumed to be planar so that $F_{ii} = 0$ for all i

To perform an update of the solution using surface i, either in the shooting or in the gathering approach, all form-factors F_{ij} are needed and may be computed in parallel. Since form-factor F_{ij} is in turn the sum of those delta form-factors where j is visible, this computation may also be performed in parallel for all j.

Figure 1.2 summarizes the parallelism available in the problem. The levels of parallelism are nested. One can go down these levels with increasing availability of processors to exploit increasing parallelism.

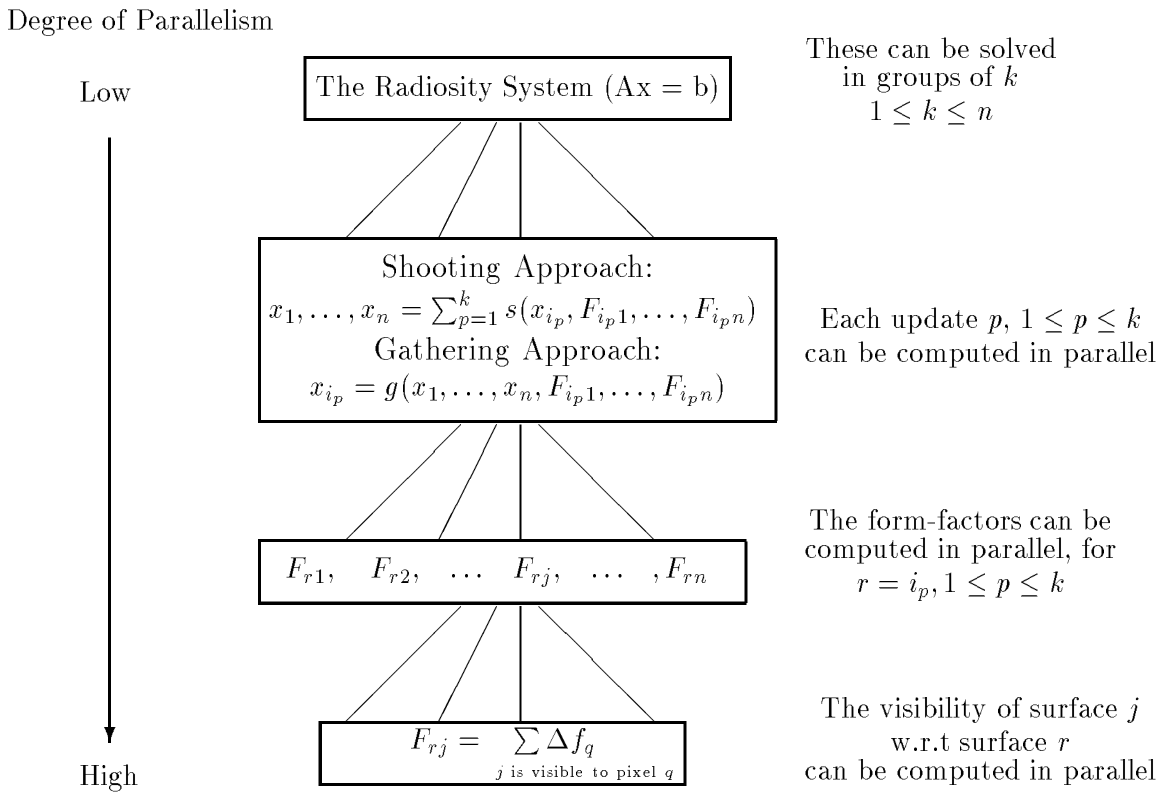
Degree of Parallelism
Low
High
The Radiosity System (Ax = b)
These can be solved in groups of k $1 \leq k \leq n$
Shooting Approach: $x_1, \ldots, x_n = \sum_{p=1}^{k} s(x_{i_p}, F_{i_p 1}, \ldots, F_{i_p n})$
Gathering Approach: $x_{i_p} = g(x_1, \ldots, x_n, F_{i_p 1}, \ldots, F_{i_p n})$
Each update p, $1 \leq p \leq k$ can be computed in parallel
$F_{r1}, \quad F_{r2}, \quad \ldots \quad F_{rj}, \quad \ldots \quad , F_{rn}$
The form-factors can be computed in parallel, for $r = i_p, 1 \leq p \leq k$
$F_{rj} = \sum_{j \text{ is visible to pixel } q} \Delta f_q$
The visibility of surface j w.r.t surface r can be computed in parallel

FIGURE 1.2. Parallelism in Radiosity

The existing work in parallel radiosity can be classified into three non-disjoint groups based on the level at which parallelism is exploited.

- *Level 1 Parallelism:* This involves computing several updates in parallel, each using one row of the form-factor matrix. This group includes [6, 7, 11, 16].

- *Level 2 Parallelism:* Form-factors within a row are calculated in parallel. This group includes [4, 13].

- *Level 3 Parallelism:* Delta form-factors are computed in parallel. Baum and Winget [4] achieve this by using the parallel rasterization hardware on the Silicon Graphics workstations.

The level at which radiosity is parallelized in these implementations is based on the extent to which a particular level of parallelization fits the underlying architecture.

Another architecture-dependent design decision is the computation of F_{ij}. In a *ray-casting* approach, rays are fired through points on the hemi-cube to determine the visible and reaching a visible surface j contributes Δf_q to F_{ij} [2, 16]. Alternatively, by firing a number of rays from surface i directly to sample points on surface j, the delta form-factors for successful rays may be added to give F_{ij} [13, 17].

The *environment-projection* approach involves projection of all surfaces in the environment onto the hemisphere surface or an approximation of it (the hemi-cube or a single-plane), and determining the visible surfaces at each pixel. The form-factor F_{ij} is the sum of all delta form-factors Δf_q where surface j is visible [2, 4, 6, 7, 11]. The determination of surface visibility at a particular pixel is done by depth-buffering the surfaces whose projections cover that pixel. The visible surface-id at each pixel is stored in an *item-buffer* for that pixel.

Unlike a ray-casting approach, where the sampling is done *from* the surface i, the environment-projection approach samples the environment *to* the surface i. This guarantees that within an error tolerance equal to the resolution of the projection-plane, no surfaces will be missed in form-factor calculations.

Our objective in this work was to choose an approach for a highly-parallel mesh-connected SIMD computer. In the ray-casting approach, the critical component is the efficiency of the ray-polygon intersection technique. An exhaustive approach that intersects a ray with all polygons works very well [13, 14, 21], but is limited to small datasets. Delany [10] proposes a more efficient space-subdivision ray-tracing method, but it has high cost and limited applicability in our setting. Consequently, we investigated an environment-projection approach. By mapping the projection-surface onto the processor mesh we can exploit coherence in the projection of a polygon onto the projection-surface. We use a shooting method as opposed to a gathering method to get progressive approximations for the entire scene.

Our current implementation parallelizes the radiosity computation at the second and third levels described in Figure 1.2, but could be extended to obtain concurrency at all levels.

1.4 MasPar MP-1 Overview

Even though the algorithm described in this paper is applicable to all mesh-connected computers, its implementation was carried out on the MasPar MP-1. This section provides a brief overview of the MasPar MP-1 before describing the implementation. A more detailed description of the MasPar MP-1 can be found in [18].

The MasPar MP-1 is a SIMD computer with a scalar execution unit and a data-parallel unit (DPU) which is an array of processing elements (PEs). Each PE has access to 16KB of local memory and can perform floating-point operations at a rate of about 75KFlops.

The PEs are organized in a 2D mesh with toroidal wrap-around on all edges. Each PE has direct connections to its 8 nearest neighbors and these are used for providing fast local communication (*xnet*). Relatively slower global communication is provided by a separate multistage crossbar network (*router*). This provides the lower-bandwidth general communication between arbitrary PEs.

The machine on which the implementation was carried out has 4096 PEs.

1.5 The Algorithm

The polygons in the model are subdivided into patches and spread out equally over the PE-array. Each patch has a *shooting energy* which initially equals its emittance. The

the radiosity of the patches has been approximated to within a predefined threshold.

Shooting patch selection phase

Since the progressive refinement approach is used, at each iteration we simultaneously distribute the energy from the k patches with highest energies. In the current implementation $k = 1$, yielding a single *shooting patch* i that is easily found using a global parallel *reduceMax* operation over the shooting energies of all the patches.

Projection phase

During an initialization phase, a matrix that transforms the world-coordinate system to the patch's projection-surface is computed and stored with every patch. The transformation matrix stored with a shooting patch is used by all the other patches to compute their projection in parallel onto the projection-surface of the shooting patch. The projection-surface in our implementation is a single plane that is able to catch 90% of the light energy emanating from the energy shooting patch. The idea of approximating a *hemi-cube* by a single-plane has been described in [19].

The single-plane is mapped across the processor mesh to maintain an orthogonal and monotonic correspondence between x and y on the plane and the processor indices such that the single-plane covers the entire mesh. Thus, neighboring pixels on the single-plane fall onto either the same processor (if the resolution of the single-plane is greater than the number of processors available) or onto an immediately neighboring processor. This mapping was chosen to exploit the coherency expected among the nearby pixels on the single-plane.

Each transformed patch determines its upper left corner on the single-plane and sends a description of the patch to the corresponding processor on the mesh using the *router* network.

Scan-conversion phase

Patch descriptions are spread-out downwards on the single-plane through the *xnet* to give single-processor thick strips. All strips (each corresponding to one patch) are then spread-out to the right in parallel, again using the *xnet*. Hence z-buffering at a processor is a matter of finding the patch description with minimum z value in the processor. The pixels that fall within the bounding-box but outside of the actual projection are not considered for z-buffering. To conserve space and improve performance, accumulation of the patch-descriptions is done only during the downward spread. During the horizontal spread, z-buffering is done on the fly as the patch descriptions travel across the processor mesh.

Form-factor calculation phase

Once the z-buffering is done, a surviving description of patch j at a pixel q on the single-plane must add Δf_q to F_{ij}. This is accomplished with a send to patch j of Δf_q using addition as a combining function. This operation is implemented efficiently, and in a way that is independent of the distribution of patches on the single-plane, using a single sort operation on the mesh followed by parallel-prefix sum and a send on the *router* network.

Radiosity update phase

The shooting energy of patch i now updates the radiosity of each patch j using

$$B_j = B_j + B_i \rho_j F_{ij} A_i / A_j \tag{1.2}$$

where A_i and A_j are the areas of patches i and j respectively. The shooting energy of patch i is set to zero. After this, the next iteration begins. This continues till the total shooting energy in the environment drops below a specified threshold.

efficient fashion. Rather than retaining F_{ij} with each patch j, we may save the delta form-factors for a particular shooting patch as a single layer across the PE-mesh. Then when a shooting patch is reused, one can get the form-factors by just adding these delta form-factors in parallel as described before. The advantage of using this strategy is that it takes the same amount of memory (of the order of the resolution of the single-plane) regardless of the dataset size and serves as a useful way to capitalize on the sparsity of the form-factor matrix and the finiteness of sampling.

1.6 Results

The results of our implementation for the *Sitterson 365 office* model with 3959 patches are summarized graphically in Fig 1.3 up to a 95% convergence of the solution. We define convergence C in iteration i as: $C_i = 1 - \frac{s_i}{s_1}$ where s_i is the energy of the shooting patch in the i^{th} iteration. The single-plane used had a side-to-height ratio of 3, allowing 91.7% of the shooting-patch's energy to be shot out per iteration. The patches for this model were derived from the initial polygons by slicing them to a global grid of 15×15 in^2 along each face. The resultant patches were then output in the form of triangles or convex quadrilaterals.

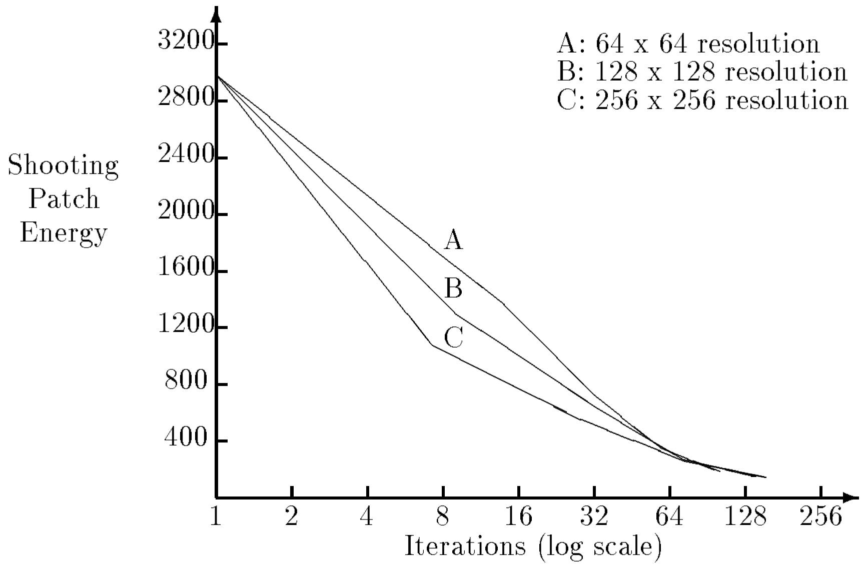

FIGURE 1.3. Convergence for different Single-Plane resolutions

The times for each iteration varied depending upon the resolution of the single-plane being used. The average times per iteration for different single-plane resolutions for the *Sitterson 365 office* model with 3959 patches are summarized in Table 1.1.

TABLE 1.1. Average Iteration Times

Single-Plane Resolution	Time (seconds)
64×64	0.24
128×128	0.45
256×256	2.15

to decrease faster than it does with a coarser resolution. The reason is that with a finer resolution single-plane, the distribution of the energy is more accurate and this leads to a faster convergence. Thus, if the resolution is coarse enough, the small patches that are in front of other patches would be covering whole pixels on the single-plane whereas they should have been covering only fractions of these. Thus, these patches get a higher share of energy than is due to them. Similarly, there would be some other patches that would be receiving lesser energy than their actual share. This energy imbalance is corrected as the resolution becomes finer.

1.6.1 Load-balancing

One of the important issues in any parallel implementation is that of load-balance. In our implementation, load-balance is an issue in the scan-conversion phase. This is because each processor on the mesh performs z-buffering for some contiguous region of pixels on the single-plane, and the projection of patches onto this plane may not be homogeneously distributed. Consequently some processor may z-buffer a larger number of patches than others and thereby increase the time to complete the scan-conversion phase. This can be particularly true in simple implementations on a SIMD machine where synchronization may come at every local communication operation.

To assess the extent of load-imbalance with our data set, we measured the maximum item-buffer depth (that is, the maximum number of patches in any item-buffer) across all processors at two points in the scan-conversion phase:
(a) After the patch description is sent to the upper left corner of the bounding box of its projection on the single-plane but before the downward spread begins.
(b) After the downward spread is done but before the horizontal spread begins.

With 3959 patches and 4096 item-buffers in the single-plane, a perfectly distributed projection would yield approximately one patch per item-buffer at point (a) in the scan-conversion phase. Figure 1.4 shows the distribution of actual maximum item-buffer depths in the two stages above over all iterations and for different resolutions of the single-plane.

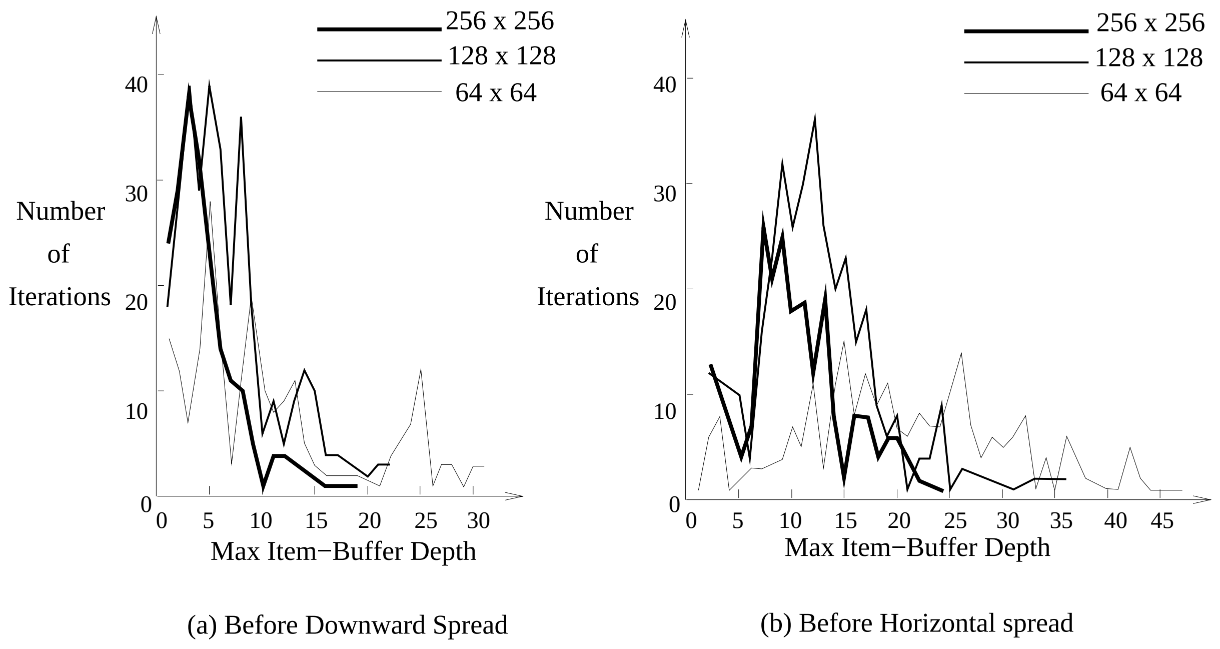

FIGURE 1.4. Load-balance during Scan-Conversion

TABLE 1.2. Average Item-Buffer Depths

Single Plane Resolution	Stage (a) Ave Max Depth	Stage (b) Ave Max Depth
64×64	10.6	20.4
128×128	6.5	11.3
256×256	4.4	10.6

The average over all iterations of maximum item-buffer depths for the two stages are shown in Table 1.2. From these values it is clear that there is substantial room for performance improvement through load balancing.

1.6.2 Form-factor Reuse

We have outlined a method to store form-factors in our approach in Section 1.5. Whether storing the form-factors for later reuse is worthwhile or not is an interesting question for all radiosity implementations. This is especially so for parallel implementations where per-processor memory limitations can restrict the sizes of problems that can be efficiently attempted. We investigated the number of form-factors being reused in our sample dataset. The results appear in Table 1.3. As can be seen from the table, most form-factors are not reused till a very late stage in the iteration process. By then the convergence is almost complete and advantages if any to be gained from storing the form-factors are minimal for a convergence as high as 95%.

TABLE 1.3. Form-Factor Reuse

Iteration	No of form-factors reused	Convergence %
13	2	53.03
32	3	76.57
59	3	88.76
101	3	94.40
177	6	97.20
392	39	98.61
1105	216	99.30
2462	810	99.65

1.7 Discussion

In this paper we have presented an environment-projection radiosity approach for mesh-connected SIMD and MIMD computers and an initial implementation. The approach builds on fine-grain parallelism available in the radiosity calculation, and hence is well-suited to highly-parallel architectures like the MasPar MP-1 as well as larger-grain parallel machines like the Intel Touchstone. We are encouraged by the results of the implementation but see substantial room for improvement. Some possible approaches follow.

Although we have not implemented the available parallelism at the top level of the classification scheme in section 1.3 such an addition would fit into our approach rather well. In particular, several projection-planes, corresponding to several shooting patches, can be mapped onto the processor mesh simultaneously. Patches can be projected onto each of the single-planes and z-buffering for several single-planes carried out simultaneously. In fact, we believe this would improve load-balance in the z-buffering phase because the

Another way to improve load-balance is to spread the item-buffers of a high-resolution single-plane more uniformly across the mesh by using a "cut-and-stack" mapping rather than a hierarchical mapping of the single-plane onto the mesh. Under the cut-and-stack mapping neighboring pixels of the single-plane are always in neighboring processors (and hence more spread-out). Toroidal topology of the processor mesh facilitates the implementation of this mapping.

Perhaps the best way to achieve load-balance is to dynamically adapt the mapping of the single-plane onto the processor mesh based on a sampling of the transformed patches or on the basis of the initial routing of top-left corners of the transformed patches. The size of the pixels on the single-plane could be altered to equalize the number of patches that project on them. If we constrain the mapping to remain orthogonal and monotonic using the techniques in [5], the efficient z-buffering technique can still be used. Thus, the single-plane would be divided finer in the regions where a large number of patches have their edges projected. This should yield better sampling than the modified single-plane method [19], and better load balancing of the computation.

In the area of application of this work, the real-time navigation and modification of architectural datasets, there is an opportunity to reduce the patches that must be considered in a radiosity calculation using visibility-cell techniques [1, 12, 20]. A large dataset, say a building, is subdivided into a number of visible cells (that would correspond to rooms) that have minimal interactions across the boundaries. Radiosity computations could then proceed in parallel across each cell, periodically transferring energies across the inter-cell portals (doors and windows for instance). This would not only parallelize the radiosity computations (at the top level of parallelization according to Section 1.3), but would also help in reducing the size of the system of equations to solve within any cell.

Acknowledgements:

We would like to acknowledge the constructive ideas, suggestions, and encouragement from Professor Frederick P. Brooks, Jr. at various stages of this work. We would also like to acknowledge the support and cooperation of the University of North Carolina at Chapel Hill Walkthrough Team. In particular, we would like to thank John Alspaugh for his model *Sitterson 365 office*.

This work has been supported in part by the following grants: DARPA Contract # DAEA18-90-C, NSF Grant #CCR-8609588, ONR Grants #N00014-86-K-0680 and #N00014-90-K-0004. Acquisition of MasPar MP-1 was supported by grants from ONR, NIH Contract #CA47982 and the State of North Carolina.

[1] J. M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculation.* PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, USA, 1990.

[2] J. M. Airey and M. Ouh-Young. Two Adaptive Techniques Let Progressive Radiosity Outperform the Traditional Radiosity Algorithm. Technical Report TR89-020, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, USA, 1989.

[3] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. In *Computer Graphics–Special Issue on 1990 Symposium on Interactive 3D Graphics*, volume 24, no. 2, pages 41–50. ACM SIGGRAPH, 1990.

[4] D. R. Baum and J. M. Winget. Real Time Radiosity Through Parallel Processing and Hardware Acceleration. In *Computer Graphics–Special Issue on 1990 Symposium on Interactive 3D Graphics*, volume 24, no. 2, pages 67–76. ACM SIGGRAPH, 1990.

[5] E. Biagioni and J. Prins. Scan-Directed Load-Balancing for Mesh-Connected Highly-Parallel Computers. In *Unstructured Scientific Computation on Scalable Multiprocessors.* MIT-Press, 1991.

[6] A. Chalmers and D. J. Paddon. Parallel Processing of Progressive Refinement Radiosity Methods. In *Second Eurographics Workshop on Rendering*, Barcelona, Spain, May 1991.

[7] S. E. Chen. A Progressive Radiosity Method and its Implementation in a Distributed Processing Environment. Master's thesis, Program of Computer Graphics, Cornell University, Ithaca, USA, Jan 1989.

[8] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. In *Computer Graphics: Proceedings of SIGGRAPH'88*, volume 22, no. 4, pages 75–84. ACM SIGGRAPH, 1988.

[9] M. F. Cohen and D. P. Greenberg. The Hemi-Cube: A Radiosity Solution for Complex Environments. In *Computer Graphics: Proceedings of SIGGRAPH'85*, volume 19, no. 3, pages 31–40. ACM SIGGRAPH, 1985.

[10] H.C. Delany. Ray Tracing On A Connection Machine. In *1988 International Conference on Supercomputing*, pages 659–667, St. Malo, France, July 1988.

[11] M. Feda and W. Purgathofer. Progressive Refinement Radiosity on a Transputer Network. In *Second Eurographics Workshop on Rendering*, Barcelona, Spain, May 1991.

[12] T.A. Funkhouser, C.H. Séquin, and S. J. Teller. Management of Large Amounts of Data in Interactive Building Walkthroughs. In *Computer Graphics–Special Issue on 1992 Symposium on Interactive 3D Graphics*, pages 11–20. ACM SIGGRAPH, 1992.

Laboratory, Connection Machine Facility, Washington, D.C., USA, Aug 1991.

[14] H. R. Good. Personal Communication, 1991.

[15] C. M. Goral, K. E. Torrance, and D. P. Greenberg. Modeling the Interaction of Light Between Diffuse Surfaces. In *Computer Graphics: Proceedings of SIGGRAPH'84*, volume 18, no. 3, pages 213–222. ACM SIGGRAPH, 1984.

[16] P. Guitton, J. Roman, and C. Schlick. Two Parallel Approaches for a Progressive Radiosity. In *Second Eurographics Workshop on Rendering*, Barcelona, Spain, May 1991.

[17] P. Hanrahan, D. Salzman, and L. Aupperle. A Rapid Hierarchical Radiosity Algorithm. In *Computer Graphics: Proceedings of SIGGRAPH'91*, volume 25, no. 4, pages 197–206. ACM SIGGRAPH, 1991.

[18] MasPar Computer Corporation, Sunnyvale, California, USA. *MasPar MP-1 Standard Programming Manuals.*

[19] R. J. Recker, D. W. George, and D. P. Greenberg. Acceleration Techniques for Progressive Refinement Radiosity. In *Computer Graphics–Special Issue on 1990 Symposium on Interactive 3D Graphics*, volume 24, no. 2, pages 59–66. ACM SIGGRAPH, 1990.

[20] S. J. Teller and C. H. Séquin. Visibility Preprocessing for Interactive Walkthroughs. In *Computer Graphics: Proceedings of SIGGRAPH'91*, volume 25, no. 4, pages 61–69. ACM SIGGRAPH, 1991.

[21] A. Varshney. Parallel Radiosity Techniques for Mesh-Connected SIMD Computers. Technical Report TR91-028, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, USA, July 1991.