1 Fast Radiosity Using A Data Parallel Architecture

Steven M. Drucker, Peter Schröder

ABSTRACT
We present a data parallel algorithm for radiosity. The algorithm was designed to take advantage of large numbers of processors. It has been implemented on the Connection Machine CM2 system and scales linearly in the number of available processors over a wide range. All parts of the algorithm — form-factor computation, visibility determination, adaptive subdivision, and linear algebra solution — execute in parallel with a completely distributed database. Load balancing is achieved through processor allocation and dynamic data structures which reconfigure appropriately to match the granularity of the required calculations.

1.1 Introduction

One goal of Computer Graphics has been the generation of realistic pictures quickly and accurately. To this end local, heuristic shading models have been replaced by more and more sophisticated techniques such as ray tracing [33] and radiosity [14]. With the advent of a number of sophisticated global, physically based shading models interest in speedup techniques has increased and a number of those can be found in the literature [8], [10], [12], [13], to name just a few. While some of these techniques attempt to, for example in the radiosity case, speed up convergence to the correct solution, or in the raytracing case, optimize sampling patterns [23] to minimize the cost of computing a picture, another major thrust has been to parallelize the basic algorithms [6], [11], [27], [24], [26], [2], [25], [5], [16]. In the present paper we discuss a new method for the computation of radiosity on a massively parallel architecture. A particular goal of our work was an algorithm which scales favorably over large ranges of input data sizes as well as numbers of available processors. We put special emphasis on the load balancing problem to allow for the use of acceleration techniques that scale for large numbers of processors and enable the ability to perform adaptive, dynamic meshing of the environment.

The algorithm has been implemented on a Connection Machine CM2 system [20]. The CM2 is a SIMD computer with between 4096 and 65536 bit slice processors running at 7MHz in which each group of 32 processors shares one floating point chip. The memory is distributed, thus specific attention must be devoted to keeping off-processor memory references to a minimum. This is a particular challenge in the context of global illumination models such as radiosity. In any parallel algorithm, load balancing among processors is extremely important. Since the algorithm was designed for a SIMD architecture in which all processors follow a single thread of execution load balancing presents itself as the challenge to have the largest number of processors participate at each step. We will show below how data parallel computation primitives such as processor allocation and parallel

prefix operators [3] can be used to meet these challenges.

In the following sections we will give a brief review of the radiosity formulation and previous acceleration methods. This is followed by a detailed description of our data parallel algorithm. Section 1.4 contains a discussion of adaptive Patch-Element subdivision, in which we examine the problem of dynamic subdivision on data parallel hardware which has not been examined in the existing literature. Section 1.5 will briefly discuss rendering the resulting radiosity scene. We finish with a discussion of the performance of our algorithm and suggestions for further work.

1.2 Radiosity Formulation

Our formulation for radiosity follows the recent literature, specifically [9] and [32]. Radiosity (Intensity per unit projected area per solid angle) is found for all the surfaces in the environment assuming only diffuse emittance and reflectance. Once the radiosities for the surfaces of a scene are known, they may be rendered directly using fast z-buffer hardware or by ray tracing to include specular effects. The formulation is based on an energy balance argument and describes the radiosity at a surface as the sum of its emittance and the diffusely reflected contributions of all other surfaces in the environment:

$$B_{dA}dA = E_{dA}dA + \rho_{dA}\int_{2\pi}\frac{B_L\cos\theta_L d\omega_L}{\pi}$$

Where dA is a differential surface, E is the emittance at that surface, ρ is the diffuse reflectance constant, and the integral is taken over the entire solid angle for a unit hemisphere around the surface. For purposes of solving this equation the integral is discretized into finite patches for which the radiosity is assumed to be constant over the patch, turning the above integral equation into a linear problem:

$$B_{A_i} = E_{A_i} + \rho_{A_i}\sum_j B_{A_j}F_{A_j-A_i}$$

where $FA_j - A_i$, called the Form-Factor from A_j to A_i, represents the fraction of energy leaving A_j and arriving at A_i. Since for all practical purposes we may assume that not all energy received by a patch is reemitted the spectral radius of this operator is strictly less than 1, allowing us to solve for the radiosities through a von-Neumann series (e.g. Gauss Seidel iteration). The latter may be interpreted as the computation of successive bounces of energy in the environment. Cohen, et al [7] suggested a faster approach. Instead of calculating the effects of all the patches on a single patch at a time—as a Gauss-Seidel approach effectively does—one can consider the effect of a single patch on the entire environment. Each patch in turn distributes its energy to the other patches scaled by the form factor. This process is called Progressive Radiosity. If the patches with the highest

amount of undistributed energy are chosen first, the solution will converge rapidly to the end result, and far less computation needs to be done before a visually accurate picture is rendered.

Finding the form factors is the most time consuming part of the radiosity process since it involves finding the visibility between all patches in the environment. In order to speed up the visibility determination, Cohen, et al [8] proposed the hemicube method. For each patch the environment is scan-converted onto a hemicube above the given patch, determining both visibility and projected area of all other patches in the environment. This method can be parallelized by distributing the calculation of form-factors to several processors, each of which computes the form-factors from a given patch to all other patches in the environment [24], [26]. Bottlenecks result when the main processor cannot communicate quickly enough with the processors computing the form factors, as the results of the form factor computations arrive. Furthermore each one of the processors needs to have access to the entire database, forcing the user to either duplicate the entire database at each processor—which becomes less and less acceptable as the number of objects in the scene increases—or accept another potential bottleneck, the concurrent access of a central database by all processors. Duplication of the database at each processing node is not a viable option for large numbers of processors, since the amount of wasted memory increases as a linear function of the number of processors. In other approaches networks of transputers have been used in a variety of configurations [5], [25], [16]. Although the latter methods scale somewhat better than sets of networked workstations used by [24], [26], they still result in bottlenecks for more than 20 or so processors. For example, Purgathofer and Zeiller [25], use a ring of 28 transputers and their efficiency dropped to 48% when all processors were used.

An alternate approach is to let a single graphics processor, such as an SGI GTX, use its already parallel geometry pipeline to calculate the form factors via the hemicube method, while the linear algebra part of the computation is parallelized onto several general purpose CPUs. This too suffers from bottlenecks since the geometry pipeline can only handle a limited number of patches at once [2]. Notice that these approaches are MIMD in nature and use message passing to coordinate the various components of the algorithm.

In the present algorithm we exploit parallelism throughout and structure the entire algorithm in a data parallel way. Since our goal was to use general purpose parallel processors, the hemicube approach, whose power is mainly derived from the available scan conversion hardware, was not as attractive as the use of ray casting to estimate visibility [32]. Furthermore, all rays arising from a given patch can be processed in parallel and various acceleration schemes from the raytracing literature can be applied. This approach also gives users a finer level of control over the necessary precision through the use of adaptive sampling. The iterative solution of the associated linear system runs in parallel as well, allowing us to use the progressive radiosity techniques of Cohen, et al. for further speed improvements. Adaptive subdivision of patches between iterations of the linear solver, is accommodated in our data parallel framework with the use of processor allocation.

1.3 Parallel Form Factor Calculation

We make use of parallel computation in several areas of the overall radiosity calculation; simultaneous form factor calculation, visibility determination during form factor calculation, adaptive patch-element subdivision, and weighted averaging of element radiosities to determine patch radiosities. In Section 1.4, we direct our attention to patch-element and adaptive subdivision techniques (substructuring) and how they can be parallelized while maintaining good processor utilization overall.

Wallace, et al [32] discuss several problems inherent in using a hemicube to calculate form factors between patches including the aliasing due to the sampling resolution of the hemicube. As an alternative, they use raytracing between patches as a way to estimate visibility. A single patch is sampled at various points from all the vertices in the environment to determine the visibility between that patch and all vertices. This can be expressed by the following equation:

$$F_{dA_1-A_2} = \frac{A_2}{n}\sum_{k=1}^{n} vis(k)\frac{\cos\phi_{ik}\cos\phi_{jk}}{\pi r^2 + A_2}$$

where patch 2 is sampled n times from vertex 1, and $vis(k)$ is 1 when the patch is visible and 0 when it is occluded. The angles represent the orientation of each patch with respect to a vector connecting the centers of each, and r represents the distance between the two patches. This calculation can be made from all vertices in the environment at the same time on a data parallel architecture. The algorithm proceeds in the standard progressive radiosity manner. The patch with the highest undistributed radiosity is found and its radiosity is distributed to all other patches in the environment, by calculating the form factor between that patch and the rest of the vertices in the environment. The radiosity for each patch is then determined from the patch vertices and another patch is selected. This process continues until the amount of undistributed radiosity present in any patch is below a specified threshold.

1.3.1 Object-Serial/Ray-Parallel Raytracing

Solving the visibility problem in parallel however is not necessarily straightforward. Initially, an object-serial/ray-parallel algorithm was used. Rays from all the vertices to a single patch are generated. All rays are then intersected at the same time with each of the objects in the database in turn. For small numbers of objects this is a viable approach, however, its disadvantages are as follows:

- All objects must be intersected with all rays, thus none of the spatial subdivision methods that are so successful at accelerating ray tracing are applied.
- Since every object must be intersected with every ray, the time it takes to calculate the intersections is independent of the number of rays (to within the number of

available processors). Under certain circumstances—incremental radiosity update, for example—it is desirable to only calculate intersections for a few rays.

- The algorithm is linear in the number of objects, so as the database grows larger, it takes that much longer to compute visibility.

Spatial partitioning techniques allow a better matching of rays and candidate objects (for intersection purposes). This is particularly helpful in a distributed memory paradigm such as ours, since it takes advantage of the spatial locality and regularity of the underlying problem.

1.3.2 Object-Parallel/Ray-Parallel Raytracing

Since the visibility problem is one of the most time consuming parts of the radiosity calculation, a more efficient method must be used. An alternative strategy for solving the visibility problem was therefore developed. This method addresses several of the drawbacks listed above. It is important to note that in the following algorithm, we are always discussing a processor set at the finest granularity of parallelism possible for the algorithm, without regard to how the actual hardware is maintaining that processor set. In the case of the Connection Machine System, a mapping from the processor set that we are using to the underlying physical processors is done either by the compiler or the operating system, thus the algorithm can be formulated independent of the number of physical processors that are actually in the system. This allows us to achieve the desired scalability properties in terms of any number of available processors.

The visibility algorithm uses a technique called *processor allocation*. Before proceeding with the description of the visibility algorithm we briefly discuss processor allocation. Consider for example, a ray which is to be intersected against a candidate list of objects. All rays, stored in their own processors, typically need to be intersected against different numbers of objects. In order to exploit ray/object parallelism, each ray allocates a number of object-processors. This is accomplished by allocating a new processor set with enough processors to hold the sum total of requested processors. See Figure 1.1. This new processor set is segmented so that each segment consists of as many processors as the associated requesting processor required. The allocating processors receive a pointer (processor address) to the segment allocated to them, which can be used to move data between the allocating and allocated processors. Though this requires general communication, if a large amount of computation is performed between allocation steps, the overhead can be amortized over the duration of the subsequent calculation. The segment_bit can be used in *segmented-scan* operations (a parallel prefix operation) to execute instructions on a per segment basis. For example, propagating (*segmented-copy-scan*) ray data to all objects which need to be intersected with the given ray. Another typical use is the *segmented-downward-min-scan*, which can be used to find the minimum intersection distance along a ray.

The processor allocation paradigm provides a general way to implement algorithms which

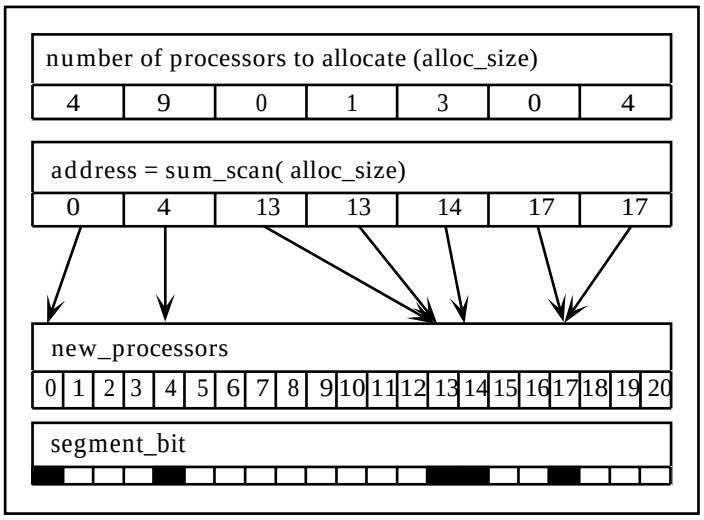

FIGURE 1.1. Processor Allocation: Each box corresponds to a processor in a 1 dimensional processor set. The parallel variable alloc_size holds the number of processors to allocate. The address of each allocated segment is given by the sum scan of alloc_size. The segment_bit delineates actual segments.

dynamically require new resources of uneven length. Another advantage of this approach is the implied load balancing. Since each processor allocates as many new processors as it needs, there are no idle processors in the new processor set (though there can be idle processors if the total number of processors requested is not some integer multiple of the number of physical processors).

With processor allocation in mind, a naive strategy to solve a multiple ray/multiple object problem would be to simply allocate a processor for every possible ray/object intersection and have all the processors compute the intersections simultaneously and then extract the closest intersections. Essentially this implies computing the full cross-product between the objects and rays; even on modest size databases, this would require far too many resources. This cross product can be reduced significantly by observing that most rays can only potentially intersect a small fraction of the entire object database. Thus the main task is to derive a small list of candidate objects which might be intersected by a given ray.

Our raytracing visibility estimation is similar to the DDA algorithm of Fujimoto, et al [12] and the shaft culling technique of Haines and Wallace [17]. All of world space is first discretized during a parallel preprocessing phase into constant sized voxels. Each voxel — in its own processor —maintains a list of objects that intersect it. To find the candidate objects along the rays, every ray enumerates the voxels through which it passes. This is accomplished by successively allocating new processor sets based on the number of divisions through which every ray passes in each of the x, y, and z directions (see Figure 1.3.2). Processors are now allocated so that every processor contains a ray/voxel pair. Each of these processors in turn finds the number of objects contained in a given voxel and allocates that many more processors. In this way we now have a processor for each candidate object along a given ray. Intersection calculations can now be performed simultaneously at every ray/object pair. The minimum intersection distance along a ray

is then found and returned for every ray in the initial set. Note that any voxel which contains no objects is immediately disregarded and no intersection calculations occur. When considering processor utilization the important measure is not the number of rays or the number of objects, but rather the product of rays and objects to be intersected. Hence, intersecting a small number of rays against a large numbers of objects is equally well accommodated as intersecting a large numbers of rays against a small set of objects.

There are still some disadvantages in the algorithm. Unnecessary work is performed since intersection calculations along the entire length of a ray are done in parallel precluding the early termination of such computations if a close object intersects a given ray. This is mostly due to the SIMD nature of the underlying hardware. On a more flexible MIMD hardware such as the CM5 this handicap could be removed easily. Timings on even small databases however show an improvement in visibility testing of more than 60 fold over

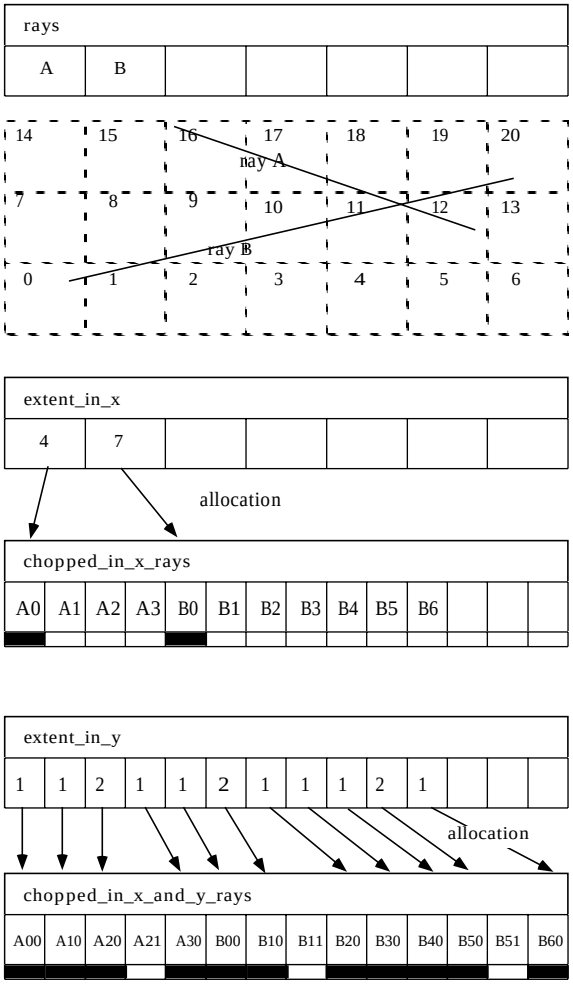

FIGURE 1.2. A 2D example 'chopping lines' to the voxel grid along the two dimensions in turn. In the real visibility algorithm, a further chop along the z axis occurs, and then, based on which voxels each chopped segment is in, the number of objects is retrieved and a final processor set is allocated which contains ray segments with candidate objects. Intersection is then calculated and the minimum intersection for each ray is found. If this intersection is between a sending patch and a receiving vertex, then occlusion takes place.

object-serial/ray-parallel visibility testing.

1.4 Substructuring

Cohen, et al [9] introduced the concept of patch element subdivision. This was based on the observation that it is often sufficient to use relatively large patches when emitting

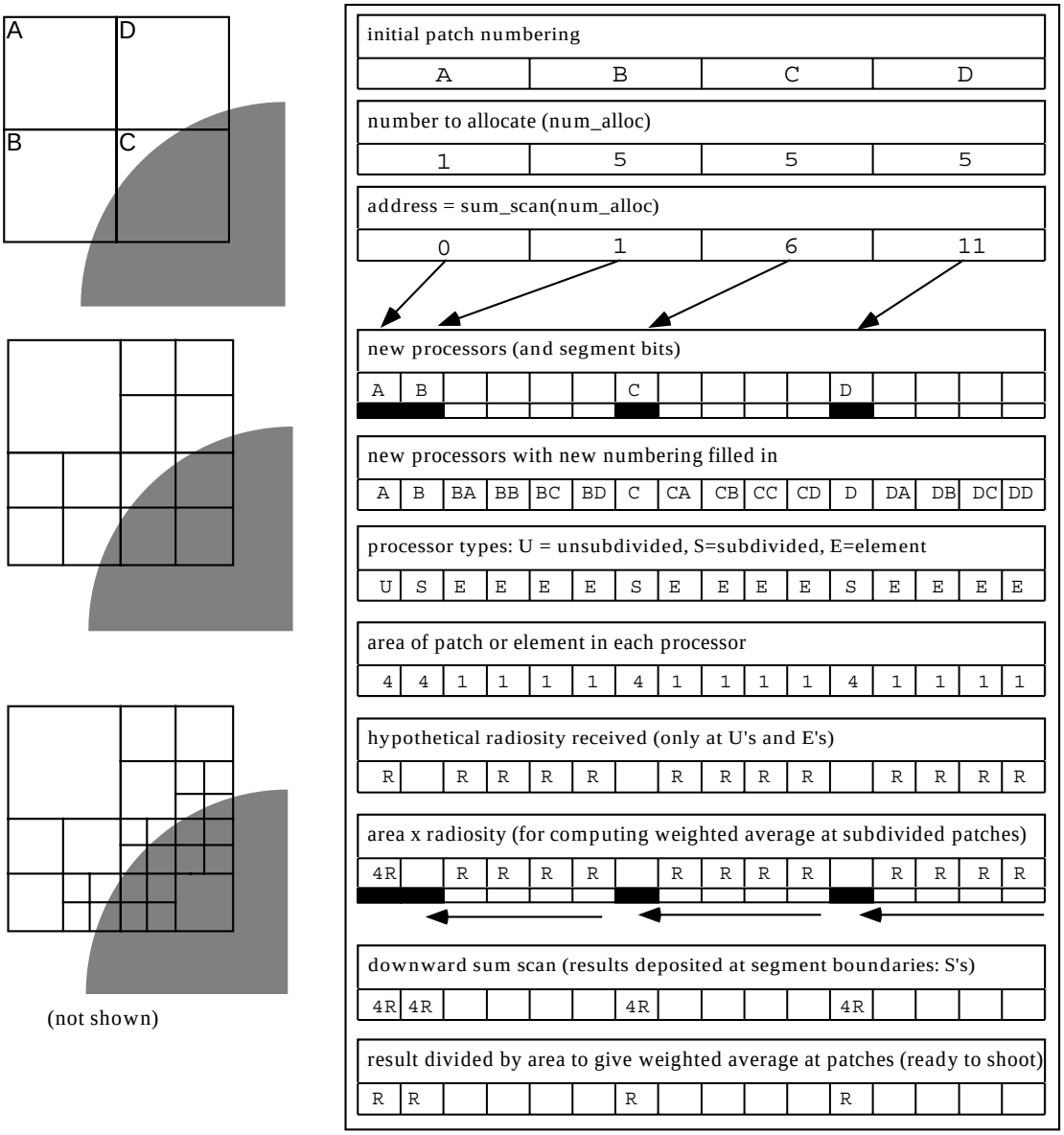

FIGURE 1.3. The process of adaptive patch element subdivision is detailed here. All processors compute whether they need to subdivide and allocate the proper number of processors. Data is sent from the allocating set to the newly allocated set. Radiosity is gathered only at unsubdivided processors and at elements. The diagram shows how a weighted average of element radiosities is gathered at the patches using a downward sum scan.

energy while small patches must be used to gather energy so that shadow boundaries can be accurately rendered.

Briefly, the algorithm proceeds as follows. Patches are divided into elements and form factors are computed from patches to element vertices. For an unsubdivided patch, radiosity is calculated at each vertex and averaged to find the total radiosity for the given patch. If a patch is subdivided into elements, radiosities are calculated at each element vertex, and an area weighted average over all the elements is calculated to yield the total radiosity for the subdivided patch.

In the parallel implementation of this operation, patches and elements are laid out in a linear processor array. This corresponds to a linear vector model of the underlying (quad) tree data structure (using the inorder numbering of the nodes and leaves of the tree). Notice that this tree is typically not balanced. The radiosity computations simply move up and down the tree. These operations can be expressed in terms of segmented scan operators (see [3]), which execute in time logarithmic in the number of data items.

This so called substructuring can either be specified with the initial database, or determined adaptively based on criteria such as radiosity gradients. Specification with the initial database requires a preprocessing step which considers size and possible shadow boundaries [15]. Notice that this preprocessing has to be very conservative—keeping the algorithm from fully realizing the savings potential—since the energy distribution is not yet known. Baum et al [1] discuss in detail pre-meshing requirements for radiosity databases.

More accurate subdivisions, which take into account the actual distribution of light in the scene, can only be found dynamically, as the radiosity computation proceeds. Adaptive subdivision was first proposed by Cohen et al in 1988 [7]. Later, Campbell et al used BSP trees to mesh polygons along shadow boundaries, though currently this method can produce a prohibitive number of polygons [4]. Hanrahan et al [18] have used dynamic subdivision along with concepts borrowed from n-body algorithms to produce significant speedups in addition to well subdivided scenes. [30] discuss a testbed for adaptive subdivision techniques to determine what criteria should be used for subdivision.

In our algorithm we use a criterion based on the difference in radiosity received at each of the vertices of a patch. During each iteration of the solution algorithm all elements evaluate in parallel the radiosity difference across their respective surface. Based on this information patches are adaptively subdivided. Since the tree data structure which encodes the relationship between sub elements and patches has been mapped onto a linear array of processors those patches/elements that require subdivision allocate new processors according to the number of child elements generated. When a leaf node in this tree is expanded, processor allocation effectively inserts the necessary number of processors into the linear vector model of the tree data structure at that node. This allocation operation can be expressed in terms of parallel prefix operators as well (see [3]). The refinement process can continue until a prespecified threshold is reached. Thresholds can be based either on the magnitude of the radiosity gradient, lower limits on patch area, or a lower limit on the projected area of a patch from a particular viewpoint [19].

An alternative to calculating the radiosity at the new vertices is to use an averaged value from the neighboring vertices. Significant artifacts can result from this approach if the radiosity gradient is not a monotonic function along the surface of a patch. In other words, if subdivision causes a newly created vertex to be in shadow when none of the patch vertices was in shadow before, the value of the radiosity at that vertex based on the averages of the radiosities at adjoining vertices would be greatly overestimated.

On the average, only a small percentage of a scene needs to be subdivided and only the form factors between the patches that have previously shot and the newly created vertices need to be recalculated. Hence our desire to use a visibility module which works equally well with few rays as with many rays. The object-parallel/ray-parallel algorithm achieves this, making the recalculation process very fast.

The flattened quad-tree representation can be augmented to implement the hierarchical radiosity described in [18]. Already the representations are quite similar, however Hanrahan et al keep track of which patches have contributed to which elements and do not need to reshoot to them.

1.5 Rendering

Once the radiosity has been calculated, or even partially calculated, the scene can be rendered by using Gouraud shading to interpolate the radiosity values at the patch vertices. This is accomplished most efficiently with modern graphics hardware. Our implementation communicates the computed radiosity values to an SGI graphic workstation via a high bandwidth channel for subsequent rendering.

Several methods have recently been introduced in the literature to include specular effects in the radiosity solution [31], [29]. Although specular effects are currently not incorporated into the radiosity pass, a second view dependent pass is included to render reflections and other specular effects. Ray tracing using the same ray tracing kernel discussed in Section 1.3.2 is used and the color values for the rays are determined for diffuse surfaces by interpolating the radiosity values from the vertices. Specular surfaces reflect the rays until the ray reaches a diffuse surface or some other termination criteria is reached.

1.6 Performance

It is always a difficult task to compare computation times for radiosity problems. Since databases tend to differ as do overall methods, there can only be approximate performance comparisons. The following table compares recent algorithms and the performance of this algorithm (labeled SIMD) on different sized machines. Note the linear scaling with the number of processors which our algorithm exhibits.

TABLE 1.1. Performance Comparison

Algorithm/Machine	No. Patches in Scene	Time / 1 Shoot (secs)	patches / sec. / shoot
Wallace (HP835)	903	34	26
Recker (6-10 HP835/325s)	1386	5.4	282
SIMD (8K CM2)	5761	3.4	1694
SIMD (16K CM2)	5761	1.7	3388
SIMD (32 K CM2)	5761	0.9	6401

1.7 Conclusion

This paper presents an algorithm on a general purpose parallel computer that can be used to calculate radiosity efficiently. More processors can be exploited without any bottlenecks resulting from the algorithm. Performance scales linearly with the number of processors. The algorithm spends most of its time computing form factors between patches and thus particular attention has been paid to the visibility problem. In addition, subdivision can be performed adaptively based on radiosity differences. Adaptive subdivision is very efficient due to the ability to exploit fast parallel operators for accumulating radiosity information at element vertices. The ray tracing kernel used in this algorithm is currently exploited both to test visibility and as a second pass to add specular information to the scene. It can also be used as a standard ray tracer [28].

Further work could incorporate the subdivision criteria of Hanrahan et al and/or use extended form factors [29]. With the current algorithm the ground work is laid to use the available compute power of massively parallel architectures for the huge computational needs of more complete solutions to the rendering equation [21]. Visibility computations still consume the largest part of the overall execution time. In this area in particular we expect major improvements since the current visibility module does not yet incorporate many of the customary optimizations known from the raytracing literature.

Acknowledgements:

Special thanks to Lew Tucker and Jim Salem who provided the atmosphere for this research to occur at Thinking Machines and to Matt Fitzgibbon who has been a constant source of help and suggestions. The rest of the Viz Team including Karl Sims, Gary Oberbrunner and Michael Johnson were inspiring friends and co-workers, and this work could never have been done without them.

1.8 References

[1] Daniel Baum and J. Winget S. Mann, K. Smith. Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions. *ACM Computer Graphics*, 25(4):51–60, 1991.

[2] Daniel Baum and James Winget. Real Time Radiosity through Parallel Processing and Hardware Acceleration. *Proceedings of the ACM 1990 Symposium on Interactive 3D Graphics*, 24(2):67–75, 1990.

[3] Guy Blelloch. *Vector Models for Data Parallel Computing*. Artificial Intelligence Series. MIT Press, Cambridge, Massachusetts, 1990.

[4] A. Campbell and D. Fussel. Adaptive Mesh Generation for Global Diffuse Illumination. *ACM Computer Graphics*, 24(4):155–163, 1990.

[5] Alan Chalmers and D. Paddon. Parallel Processing of Progressive Refinement Radiosity Methods. *Proceedings of Second Eurographics Workshop on Rendering*, May 1991.

[6] John Cleary, Brian Wyvill, Graham Birtwistle, and Reddy Vatii. Multiprocessor Ray-tracing. *Computer Graphics Forum*, 5:3–12, 1986.

[7] Michael Cohen, Eric Chen, John Wallace, and Don Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. *ACM Computer Graphics*, 22(4):75–84, 1988.

[8] Michael Cohen and Don Greenberg. The Hemi-cube: A Radiosity Solution for Complex Environments. *ACM Computer Graphics*, 19(3):31–40, 1985.

[9] Michael Cohen, Don Greenberg, D. Immel, and P. Brock. An Efficient Radiosity Approach for Realistic Image Synthesis. *IEEE Computer Graphics and Applications*, January 1986.

[10] S. Coquillart. An Improvement of the Ray-tracing Algorithm. *Proceedings Eurographics*, pages 77–88, 1985.

[11] Hubert C. Delaney. Ray Tracing on a Connection Machine. *Proceedings of the 1988 ACM/INRIA. International Conference on Supercomputing*, pages 659–667, 1988.

[12] A. Fujiomoto, T. Tanaka, and K. Iwata. ARTS: Accelerated Ray Tracing System. *IEEE Computer Graphics and Applications*, April 1986.

[13] Andrew Glassner. Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics and Applications*, October 1984.

[14] Cynthia Goral, Ken Torrance, Don Greenberg, and B. Battaile. Modeling the Interaction of Light between Diffuse Surfaces. *ACM Computer Graphics*, 18(3):213–222, 1984.

[15] Don Greenberg, Michael Cohen, and Ken Torrance. Radiosity: A Method for Computing Global Illumination. *The Visual Computer*, 2:291–297, 1986.

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

[17] Eric Haines and John Wallace. Shaft Culling for Efficient Ray-traced Radiosity. *Proceedings of Second Eurographics Workshop on Rendering*, May 1991.

[18] Pat Hanrahan, David Salzman, and Larry Aupperle. A Rapid Hierarchical Radiosity Algorithm. *ACM Computer Graphics*, 25(4):197–206, 1991.

[19] Paul S. Heckbert. Adaptive Radiosity Textures for Bidirectional Ray Tracing. *ACM Computer Graphics*, 24(4):145–154, 1990.

[20] Danny W. Hillis. *The Connection Machine*. MIT Press, Cambridge, Massachusetts, 1985.

[21] James T. Kajiya. The Rendering Equation. *ACM Computer Graphics*, 20(4):143–150, 1986.

[22] Tim Kay and James Kajiya. Ray Tracing Complex Scenes. *ACM Computer Graphics*, 20(4):269–278, 1986.

[23] Don P. Mitchell. Spectrally Optimal Sampling for Distribution Ray Tracing. *ACM Computer Graphics*, 25(4):157–164, 1991.

[24] Claude Puech, Francois Sillion, and C. Vedel. Improving Interaction with Radiosity-based Lighting Simulation Programs. *Proceedings of the ACM 1990 Symposium on Interactive 3D Graphics*, 24(2):51–58, 1990.

[25] W. Purgathofer and M. Zeiller. Fast Radiosity by Parallelization. *Proceedings Eurographics Workshop on Photosimulation, Realism, and Physics*, pages 173–185, 1990.

[26] R. Recker, D. George, and D. Greenberg. Acceleration Techniques for Progressive Refinement Radiosity. *Proceedings of the ACM 1990 Symposium on Interactive 3D Graphics*, 24(2):59–64, 1990.

[27] I. Scherson and E. Caspary. Multiprocessing for Ray Tracing: a Hierarchical Self-Balancing Approach. *The Visual Computer*, 4, 1988.

[28] Peter Schröder and Steven Drucker. A Data Parallel Algorithm for Raytracing of Heterogenous Databases. *Proceedings of Graphics Interface '92*, pages 167–175, May 1992.

[29] Francois Sillion and Claude Puech. A General Two-Pass Method Integration Specular and Diffuse Reflection. *ACM Computer Graphics*, 23(3):335–344, 1989.

[30] C. Vedel and C. Puech. A Testbed for Adaptive Subdivision in Progressive Radiosity. *Proceedings of Second Eurographics Workshop on Rendering*, May 1991.

[31] John Wallace, Michael Cohen, and Don Greenberg. A Two Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods. *ACM Computer Graphics*, 21(4):311–320, 1987.

[32] John Wallace, K.A. Elmquist, and Eric Haines. A Raytracing Algorithm for Progressive Radiosity. *ACM Computer Graphics*, 23(3):315–324, 1989.

[33] Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–3349, 1980.