Efficient Parallel Global Illumination using Density Estimation

David Zareski Bretton Wade Philip Hubbard Peter Shirley

Program of Computer Graphics *
Cornell University

Abstract

This paper presents a multi-computer, parallel version of the recently-proposed "Density Estimation" (DE) global illumination method, designed for computing solutions of environments with high geometric complexity (as many as hundreds of thousands of initial surfaces). In addition to the diffuse inter-reflections commonly handled by conventional radiosity methods, this new method can also handle energy transport involving arbitrary non-diffuse surfaces. Output can either be Gouraud-shaded elements for interactive walkthroughs, or ray-traced images for higher quality still frames.

The key difference of the DE algorithm from conventional radiosity, in terms of its ability to parallelize efficiently, is its microscopic view of energy transport, which avoids the $O(n^2)$ pairwise surface interactions of most previous macroscopic radiosity algorithms (*i.e.*, those without clustering).

Parallel DE is implemented as two separate parallel programs which perform different phases of the DE method. The first program performs the particle-tracing phase, and the second performs the density-estimation and meshing phases. Each parallel program consists of a single master task and multiple worker tasks executing on separate workstations connected over a local area network. Communication is performed using the PVM software package and a shared file system.

The goal of this effort is to provide a near-linear speedup for solutions to existing environment models using tens of processors. The parallel efficiency of the first program has been measured to be above 90% for as many as 16 workers, and the parallel efficiency of the second program has been measured to be above 70% for as many as 12 workers.

CR Categories and Subject Descriptors: D.1.3 [Programming Techniques]: Parallel Programming; I.3.0 [Computer Graphics]: General; I.3.6 [Computer Graphics]: Methodology and Techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Additional Key Words and Phrases: Parallel computing, network computing, radiosity, realistic image synthesis, global illumination, density estimation.

*580 Frank H. T. Rhodes Hall, Ithaca, NY 14853. World Wide Web home page: http://www.graphics.cornell.edu

1 Introduction

Since the introduction of the radiosity method into computer graphics, it has become one of the key global illumination algorithms for use in photorealistic rendering. Its ability to accurately compute interreflections in a view-independent manner makes it an excellent technique to use in many applications, particularly architectural walkthroughs.

Even though several performance improvements to the original algorithm have been made, including hierarchical radiosity [14, 6, 27] and Monte Carlo radiosity [18, 1, 22, 20], which can in practice reduce the execution time from $O(n^2)$ to $O(n \log n)$ in many cases, performance remains a key research issue. The time needed to compute a sufficiently accurate view-independent global illumination solution is still often measured in hours, or even days, particularly for large, complex environments. This long execution time continues to be a major reason why these algorithms have failed to gain wider acceptance and usage in commercial applications.

Therefore, the problem of rapidly computing an accurate view-independent global illumination solution is one that could benefit from applying all of the present-day computing resources that can be brought to bear in order to achieve the best possible performance. Such problems are typically beyond the capabilities of even the most powerful uniprocessor, either because of the sheer number of floating point operations (flops) required, the main memory size needed, or a combination of both. Only by integrating many individual processors into a single parallel system can we harness the number of gigaflops of processing power and gigabytes of memory needed to attempt such a rapid solution.

The key issues in parallel programming are decomposing the original problem into parallel tasks and distributing these tasks and the associated data structures across the system's collection of processors and memory. In order to do this efficiently, a good understanding of both the original problem and the communication and memory characteristics of the parallel system are required. Tuning based on empirical experience is also essential to maximize performance.

It is often appropriate to re-examine the original problem to be solved, and to devise a new algorithm to solve it, rather than to naively parallelize an existing serial algorithm. If the new algorithm is designed to possess inherent parallelism, it is likely that it can be parallelized more easily and efficiently, and will outperform the existing algorithm by a wider margin as a larger number of parallel processors are used.

We believe that our research provides a practical case study of this process. We re-examined the global illumination problem, determined the limits to parallelism in existing radiosity algorithms, devised a new view-independent global illumination algorithm with much richer inherent parallelism, and then implemented both serial and parallel versions of the new algorithm.

This new global illumination method, called "Density Estimation," is named after the statistical technique we used to compute surface irradiance [24], and is described in detail in our companion paper [23]. This current paper briefly summarizes our study of global illumination and development of the DE method, and then focuses on the aspects of that method that are key to the parallel implementation. Next, performance results from that implementation are presented. Finally, we make some concluding remarks and briefly discuss some ideas for future work.

2 Prior Work

Prior research has been done on parallel progressive radiosity by several groups [9, 21, 4, 11, 8, 10, 31, 7, 5], and Chalmers and Paddon have written a survey paper [19] which analyzes both parallel full-matrix (gathering) radiosity and parallel progressive (shooting) radiosity methods. The most recent of these papers, by Bouatouch and Priol [5], showed an efficiency greater than 82% for as many as 32 processors on scenes of 17,500 and 50,500 patches.

Unfortunately, much of this prior work deals primarily with older varieties of the serial radiosity algorithm, which can lead to some inaccuracies in the resulting solutions, such as the well-known artifacts of the hemi-cube method [3]. An even larger concern is that progressive refinement radiosity [32] is an O(n^2) algorithm when run to final convergence[1].

This places a severe limitation on the size of the environment that can be solved, as even a perfectly linear increase in performance due to parallel processing cannot offset a quadratic increase in execution time as the number of surfaces in the environment is increased.

The most prominent sub-quadratic radiosity algorithm is hierarchical radiosity [14], which treats subdivided surface patches as a hierarchy rather than a set of independent patches for the purpose of energy transport. Subsequent "clustering" enhancements have been devised [6, 27] which can hierarchically group separate surfaces as well. Hierarchical radiosity was inspired by hierarchical solutions to the "N-body" gravitational force problem in physics, where the time complexity was reduced from O(n^2) first to O(n log n) and later to O(n) by approximating the forces between two well-separated clusters of particles by a single force, and using an octree to store the resulting 3-D hierarchy.

Little prior work has been published to date on parallel hierarchical radiosity. Publications from the Computer Systems Laboratory at Stanford University [25, 26] state that most ideas which succeed for parallelizing the gravitational N-body methods do not succeed nearly as well for hierarchical radiosity. The only technique they found to be effective was on-the-fly task-stealing to balance the load among processors during each iteration, which requires shared memory and global data access to be efficient. Their attempt to implement this algorithm on a distributed-memory message passing parallel system became "an exercise in frustration" and was abandoned as not being worthwhile [25].

Another attempt to parallelize hierarchical radiosity on a distributed-memory system is described by Carter in his PhD thesis [6]. Despite a sophisticated parallel decomposition on the nCUBE 2 parallel supercomputer, including a detailed performance analysis, Carter concludes that "the absolute performance realized is not impressive" (page 103) and "the apparently poor showing ... serves to emphasize the communications-intensive nature of the algorithm" (page 98).

[1]If a smaller number of iterations is used, the time complexity improves, but it becomes difficult to judge the quality of the resulting solution.

We have also made some initial attempts to parallelize hierarchical radiosity (before the advent of clustering), using a two-pass method with discontinuity meshing [17]. The "local pass," which refines the initial "global pass" solution, parallelized quite well on a small scale, with a parallel efficiency of 75% or higher for up to eight processors [34]. This was because the potentially $O(n^2)$ interactions between patches during the local pass was significantly reduced using visibility-determination preprocessing techniques [30]. We found no clear way to parallelize the global pass, however.

Since the global pass takes about one quarter of the total serial execution time in typical usage, the overall parallel speedup of this two-pass algorithm was limited to a factor of four. This limitation led us to rethink the global illumination problem itself, and to search for a new algorithm with richer inherent parallelism.

3 Re-examination of Radiosity and Global Illumination

After studying existing radiosity algorithms, we came to believe that "the" main limit to parallelism was their view of energy transport, as exemplified by the global pass of the two-pass hierarchical-radiosity algorithms [17, 27]. By viewing energy transport from the macroscopic level, the process potentially involves pairwise interactions between all surfaces in the environment.

Although hierarchical techniques with clustering [6, 27] can usually reduce the number of interactions from O(n^2) to O(n log n) by clustering collections of surfaces of limited visibility to another surface, and by eliminating interactions between mutually-occluded pairs of surfaces, the determination of visibility is usually not a local problem[2]. Therefore these techniques may still require global data access to nearly all surfaces in the environment. This access pattern makes the job of parallelizing radiosity methods efficiently quite difficult, since frequent global data access implies a lot of intertask communication.

As an alternative, we chose to take a microscopic view of energy transport. By modeling light as discrete particles, light energy transport can be simulated by following the paths of individual light particles emitted from a luminaire[3]. These particles travel in a straight line until they encounter another surface in the environment, where they are either reflected, refracted, or absorbed.

With this view, energy transport is completely parallelizable. Any number of parallel tasks can be used, since each light particle can be tracked independently. If the size of the input environment is not too large, and the memory overhead is kept low, it is feasible to fit the entire input environment into each task's local memory. In this case no intertask communication is needed at all.

[2]In cases where the environment has a very high "global complexity" (large number of surfaces), but a limited "local complexity" (only small subsets of surfaces are mutually visible), partitioning can be used to decompose the environment into subsets which can be solved separately [29]. But if any subset has a high local complexity, then partitioning may not reduce the subproblems to a solvable size.

[3]This idea, of course, is not new, and has been the basis of Monte Carlo algorithms since the 1950s, including the classic computer graphics ray tracing algorithm [33] and and Monte Carlo radiosity's photon tracing [22, 20]. The parallelization of Monte Carlo algorithms has a long history as well, from Ulam's work in the 1950s to more recent research [28].

Therefore, our new algorithm for energy transport more closely resembles ray tracing than the classical radiosity algorithms, which is encouraging since ray tracing has been parallelized successfully [13, 2, 16]. One key difference is that our algorithm traces rays from the luminaires (light sources) to the surfaces, while a ray tracer sends rays from the eye to the surfaces and then to the luminaires. In this respect, our algorithm more closely resembles Heckbert's bidirectional ray tracer [15], and can produce either a 3-D view-independent solution or a 2-D view-dependent image, whereas a ray tracer can only produce the latter.

Unlike Monte Carlo radiosity, which also uses particle tracing but then typically lumps the incoming energy received by each surface back into a single scalar value, our algorithm records this energy as a list of particle "hit points".

The number of hit points needed to determine the exact illumination of every surface is unacceptably large. In fact it is probably several orders of magnitude beyond the capabilities of today's computing resources.

Fortunately, however, the illumination of each surface can be approximated, or "estimated," from a reasonably small set of hit points using a statistical technique known as "density estimation" [24].

The density estimation of the illumination of each surface can be computed independently, and therefore parallelizes quite well. Furthermore, since the density estimation at any given point on the surface depends only on the hit points in its local neighborhood, the processing of a single, large surface can also be parallelized.

4 Overview of the Density-Estimation Algorithm

The DE algorithm first emits photon-like particles from the luminaires and then records their paths through the environment. When all particles traces are completed, each surface has a distribution of particle hits that are denser in bright regions than in dimly-lit regions. A density-estimation [24] technique is then used to construct a smooth illumination function with a shape that is consistent with the hit point densities. This function is then converted into a piecewise linear mesh for more efficient display.

The algorithm is composed of three basic phases:

1. **Particle-tracing phase:** Power carrying particles are emitted from each luminaire using an appropriate radiant intensity distribution, and are then tracked through the environment until they are probabilistically absorbed. A list of all particle "hit points" is saved.

2. **Density-estimation phase:** The stored hit points are used to construct an approximate irradiance function $H(u, v)$ for each receiving surface.

3. **Meshing phase:** The approximate irradiance function $H(u, v)$ is further approximated to a more compact form $\bar{H}(u, v)$ that can be used for efficient hardware rendering or ray tracing display. If the desired output is a set of Gouraud-shaded polygonal elements for interactive display and walk-through on a conventional graphics workstation, then $\bar{H}(u, v)$ will be piecewise linear.

   We have found that a convenient way to generate this piecewise linear approximation is to use two steps: generate a high-resolution "initial mesh" and then remove unneeded mesh elements by applying a variation of a geometric mesh-decimation algorithm.

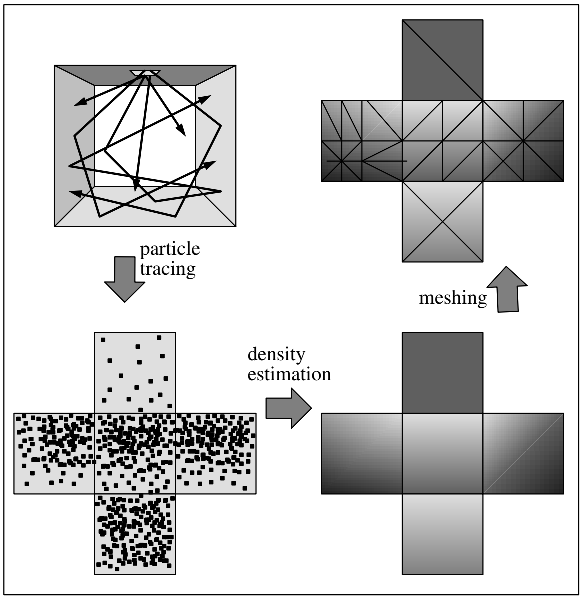

Figure 1: Overview of density estimation algorithm. The surfaces in the room at the upper left are depicted "unfolded" in the three subsequent images.

The algorithm is outlined in Figure 1. Further details of this algorithm can be found in our companion paper [23].

The adjustable variable in this algorithm is the number of particles that are traced. More particles will result in better resolution of illumination detail. One of the strengths of the algorithm is that all particles can be traced independently, and all surfaces can be processed independently once the tracing is completed.

5 Parallel Implementation

We implemented the Density Estimation algorithm as two separate parallel programs which communicate using files, as shown in figure 2. Each of these programs is described in more detail in separate sections below following some general comments that apply to both programs.

The first program performs the particle-tracing phase and the second performs the density-estimation and meshing phases. Temporary files are used within the second program to communicate between the initial-meshing and mesh-decimation routines.

We chose the classic master/worker model as the parallel decomposition for both parallel programs. In each program, the master task is started first, and then spawns the user-specified number of worker tasks.

Each worker task should be run on a separate processor to avoid contention for a processor's computing cycles and memory. The master task can either be run on a separate processor, or on the same processor as one of the workers, as the master's use of computing cycles and memory is generally minimal during the parallel sections of the algorithm.

For implementing the intertask communication, we chose the PVM 3 (Parallel Virtual Machine version 3) message passing library, which is developed and distributed by Oak Ridge National Laboratory [12]. PVM's main advantages are that it is readily available and it supports a wide range of machines, including many brands of workstations, main-

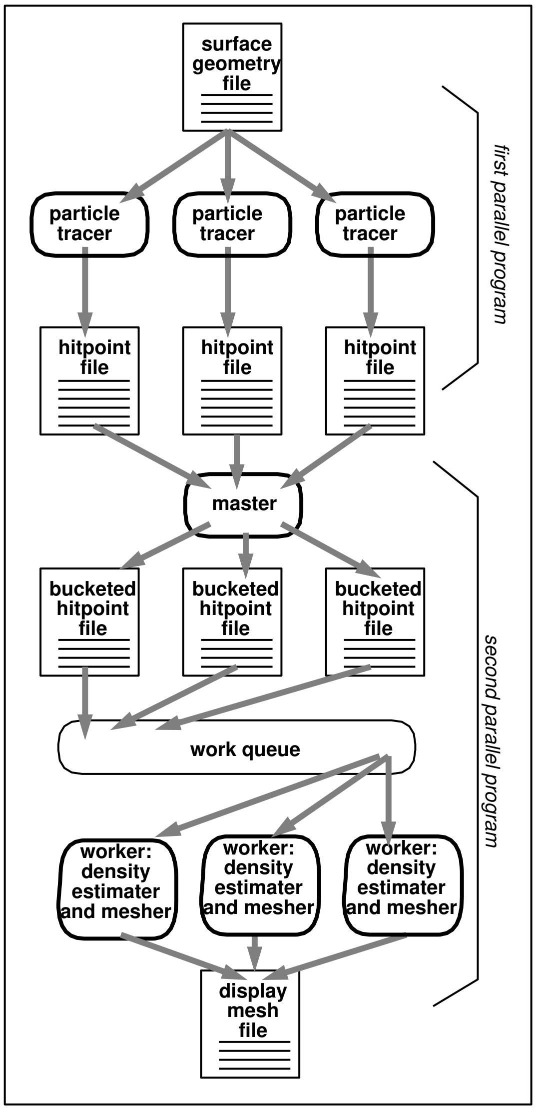

Figure 2: Overview of the parallel implementation.

frames, and parallel supercomputers. PVM's main disadvantage, at least for our cluster of workstations, is that it is layered on top of UNIX and network software protocols, so it carries a fairly high overhead, particularly in terms of the start-up cost of sending and receiving each message.

Therefore, we made the following choices:

- keep the amount of communication between the master and the workers to a minimum;
- buffer data into larger pieces for message passing;
- write and read all persistent data directly to/from the shared network file system to lessen the PVM message traffic on the network.

5.1 Parallel Particle Tracing

The parallel particle tracer is written in C++, and is a straightforward extension of the serial particle tracer. Both the parallel and serial versions share the same source code, and the appropriate pieces of the code for the desired version are conditionally selected at compile time.

In the parallel version, the user starts the master task, specifying the total number of particles to be traced (n_p) and the number of worker tasks to be used (n_w). The master spawns the specified number of workers and assigns some number of particles to each worker. If all workers are running on separate processors of equal speed, then each worker will be assigned n_p/n_w particles to trace. Otherwise different fractions of the particles are assigned to each worker using ratios of the relative processor speeds to balance the load.

There is considerable variation in the execution time required to trace a single particle, often as much as 100 to 1, depending on how often it is reflected or refracted. The "law of large numbers" implies, however, that the variation in total execution times among workers will be small compared to the total execution times themselves. Therefore each worker should finish tracing its set of particles at approximately the same time, resulting in efficient load balancing.

Each worker reads the input environment from the shared file system and traces the assigned number of particles. When a particle hits a diffuse surface, a hit point is recorded and buffered. When a buffer becomes full, the worker writes its contents to a separate hit point file.

Given the Monte Carlo nature of the particle tracing, all of the hit point sets from the separate workers can be treated as if they were a single large set of hit points from a serial particle tracing. The only requirement is that each worker use a different random number seed for its execution. This can be accomplished by specifying a random number seed set of consecutive integers to be assigned one to each worker at the beginning of the parallel run.

At a minimum, no further output from the parallel particle tracing is needed beyond the files of hit points. To aid in estimating the cost of performing the parallel density estimation and meshing of each surface however, it is advantageous to output a file of hit-point counts for each surface. These counts can be used by the next parallel program to help divide the environment into pieces of roughly the same processing cost. This avoids an extra pass through all the hits points that would otherwise be needed to recompute the counts later.

Each worker maintains these counts as it executes and optionally outputs them upon completion. In our current implementation, these counts are passed back from each worker to the master, who sums them together to produce the total counts which are then written to a file. Performing this summation serially on the master is acceptable for a limited number of workers, since it can be done very quickly relative to the time spent on the rest of the particle tracing phase. If a very large number of workers are used, this process could instead be performed by the workers in a binary tree fashion.

As with the serial particle tracer, each parallel run produces a file of hit points for a single frequency channel. Therefore three separate runs (red, green, and blue) are needed to produce the three files of hits points required for a color (RGB) solution.

The parallel particle tracer has been run successfully using as many as 24 workers. Some representative performance timings of the parallel particle tracer are given in Section 6.

5.2 Parallel Density Estimation and Meshing

First we present an overview of the parallel density estimation and meshing algorithm, followed by some separate discussion of some of the finer points and techniques used to improve the parallel performance.

5.2.1 Overview

The parallel density estimation and meshing program is written as two separate pieces of code: a master task (written in C) and a worker task (written in C++). The user starts the master task and supplies it with the needed parameters, including the number of worker tasks to spawn, and the number of "work pieces" to create.

The input surfaces are partitioned and distributed either statically or dynamically, depending on the number of work pieces specified. Piece distribution is described more fully in Section 5.2.3.

Given this information, the master spawns the desired number of worker tasks and identifies which piece each surface is to be assigned to. It then creates one piece for each worker by packing the appropriate surface data for each initial piece, and distributing these pieces to the workers.

The workers each receive their piece, unpack the surface data, and perform the density estimation and initial meshing for each of the surfaces in that piece. Then, if mesh decimation is specified, all of the initial meshes are decimated to produce the final meshes. Otherwise the initial, undecimated meshes become the final meshes[4].

When the meshing of a piece is completed, the worker notifies the master, and then waits for permission before writing the final meshes to a permanent output file. The mesh output can be written to either a single shared output file, or to separate output files which can be concatenated later into a single file.

As the workers complete the processing of their pieces, the master grants them access to the shared output file (if used). When dynamic distribution is specified, the master distributes the remaining pieces one at a time until there are none left. At that point the master tells each finishing worker to terminate. Once all workers have terminated the master concatenates the separate output files (if used), informs the user that the solution is complete, and exits.

5.2.2 Approximate Cost Function

In order to divide the work equally among workers, an approximate cost function is needed to predict the amount of time needed to perform the density estimation and meshing for a given input surface. The desired characteristics of this function are that it be both accurate and inexpensive to compute.

Two such functions were tried:

1. cost for each surface is the same (constant cost);

2. cost for each surface is $k_1 + k_2 n$
   (fixed cost plus linear cost per hit point)[5].

The first function (constant cost) is trivial to compute, and is easily implemented by distributing an equal number of surfaces to each worker. The drawback is that it can be highly inaccurate, as it treats every surface identically, whether it is a tiny surface in a dimly-lit corner or the entire floor of a large room.

[4]Mesh decimation can greatly reduce the time needed to display the solution, which is desirable for an interactive walkthrough application. If only a single high-quality image of the solution is needed, mesh decimation can be skipped.

[5]We obtained good results with $k_1 = 1000$ and $k_2 = 1$.

The second function (fixed cost plus linear cost per hit point) is slightly more work to compute, but it is straightforward to implement by using the files of hit-point counts optionally available from the parallel particle tracer. Its accuracy was found to be much better than that of the constant cost function, and therefore was the function used in measuring all of the results listed in Section 6.

5.2.3 Piece Identification and Distribution

Two methods of identifying and distributing work pieces with approximately equal cost were tried, each based on adding surfaces to a piece until the desired cost per piece is reached or exceeded. These methods are:

1. Sequential assignment:

   Assign surfaces i, $i+1$, $i+2$, etc. sequentially, summing the total cost, until the desired cost is reached.

2. Histogram-based assignment:

   Build a set of surface buckets based on a $\log_2$ histogram of hit-point counts, and then choose surfaces starting from the "top" nonempty bucket downward until the desired cost is reached.

Both methods were found to be effective for roughly dividing the total work evenly among the worker tasks. The histogram-based method took more effort to implement, and required larger data structures, but its ability to identify the surfaces with the most hit points (i.e. those in the top bucket) and process them first definitely made this extra effort worthwhile.

The distribution scheme used was simply to divide the input scene into the number of pieces specified by the user, distribute one piece initially to each worker, and then distribute any remaining pieces individually upon demand. Static distribution is achieved by setting the number of work pieces equal to the number of worker tasks. Increasing the number of work pieces results in a dynamic distribution.

This allowed us to experiment with computing solution times for any desired number of pieces. For the experiments we ran with up to 16 worker tasks (each running on a separate processor), we found empirically that 50 or 100 pieces were a reasonable number to obtain a good load balance without adding excessive communications overhead.

This dynamic distribution of many small pieces, as opposed to a static distribution of one large piece per worker, was worthwhile. It helped to smooth out inaccuracies in the approximate cost function and provide a more balanced load and quicker overall solution time than could usually be obtained from an *a priori* static distribution.

5.2.4 Hit Point Partitioning and Distribution

The best method for partitioning and distributing the hit points is likely to be system-dependent, with the main considerations being the real memory size of the processors, the effective bandwidth of the interconnection network, and the configuration of the file system. For our system, an FDDI network of workstations with a shared file system and a single disk server, we obtained the best performance by partitioning the original set of hit point files into another set of temporary hit point piece files in the master task running on the disk server[6].

[6]Two possible ways to avoid doubling the disk space needed are to drop the hit point id field for single-surface pieces, and to erase the original input files during the piece-partitioning process.

The workers read the temporary hit point piece files directly via the shared file system, which in this case was found to be more efficient than communicating this data with PVM messages. The count of hit points is sent from the master to the worker as part of a piece distribution, and tells the workers how many hit points to expect. This allows for a parallel overlap, with the workers reading the partially written hit points files for their initially-assigned pieces even while the master is still writing them.

Once a set of hit points is distributed to a worker, there is again a choice of how to process the points most efficiently. When the number of hit points is small enough, then it may be best to read them all into real memory and process them simultaneously. When this is not practical, then the density estimation program should be able to process them incrementally (in sequential subsets). The latter choice places some restrictions on the type of density-estimation technique that can be used however[7].

6 Results

Environments of varying size and complexity were used to test and debug the parallel DE programs. Of those environments, we chose three for systematic study and analysis.

Sample color images of these environments are shown in Plates 1, 2, and 3, with each plate showing a pair of images of the same environment side by side. The ones on the left are rendered from undecimated meshes, which are preferable for higher-quality still frames. The ones on the right are rendered from decimated meshes, which are preferable for use in an interactive walkthrough application. As can be seen is these plates, our mesh decimation algorithm does introduce some additional error, but also substantially decreases the mesh size of the solution.

Plate 1 shows the "Cornell box," a traditional radiosity test environment of 17 polygons, with fairly predictable illumination and several existing "reference" images produced by various radiosity algorithms that can be used for comparison. It is included here as a proof-of-concept. The performance results for parallel particle tracing of the Cornell box is shown in Figure 3, and the performance results for the parallel density estimation and meshing of the Cornell box is shown in Figure 4.

Plate 2 shows a long and narrow, furnished room based on an architectural design by Charles Rennie Mackintosh, containing approximately 10,000 surfaces. This is a richer environment than the Cornell box, and is representative of a typical-sized environment that is solvable with most contemporary radiosity programs. It is sufficiently complex to produce a non-trivial execution time, so that a significant speedup would be highly desirable for production use. The parallel performance results for the Mackintosh room are shown in Figures 5 and 6.

Plate 3 is a large furniture warehouse, which contains approximately 150,000 surfaces. This environment is three to four times larger than the largest scenes used in prior radiosity research [29, 27, 5], and is more typical of the size that view-independent global illumination algorithms will be facing in the near future. To simplify the modeling, this environment was built by replicating a base model along two

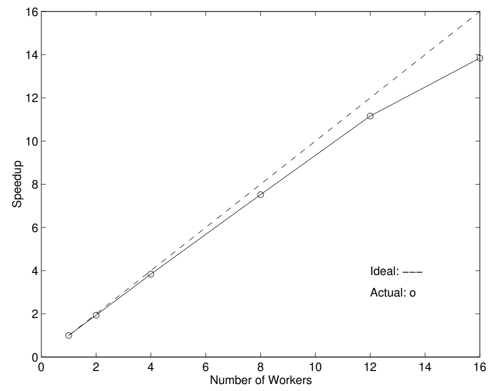

Figure 3: Parallel particle tracing speedup for the Cornell box. (The 45-degree line indicates ideal linear speedup and the line below it shows actual results.) The execution time for this phase ranged from a maximum of 5h46m to a minimum of 25m.

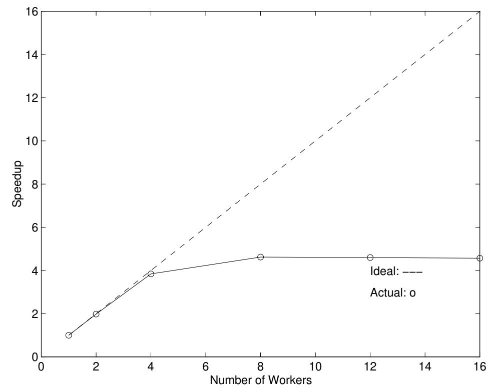

Figure 4: Parallel density estimation and meshing speedup for the Cornell box. Execution time ranged from a maximum of 12h52m to a minimum of 2h47m. The speedup is limited in the current implementation by the time needed to process the surface with the largest fraction of the hit points (22%), as the the processing of an individual surface has not yet been parallelized.

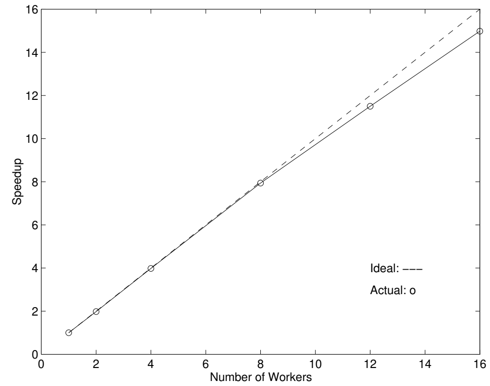

Figure 5: Parallel particle tracing speedup for the Mackintosh room. Execution time ranged from a maximum of 46h11m to a minimum of 3h05m.

[7] An incremental approach would generally be more compatible with a single-pass kernel method, rather than a multi-pass adaptive kernel method.

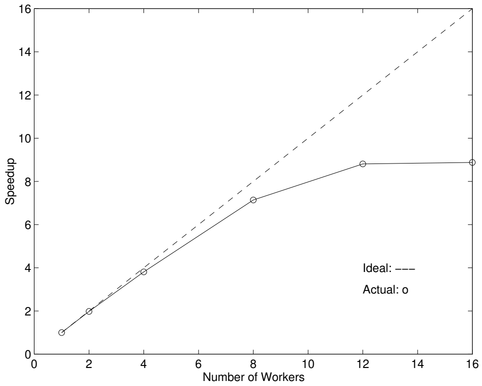

Figure 6: Parallel density estimation and meshing speedup for the Mackintosh room. Execution time ranged from a maximum of 20h07m to a minimum of 2h16m. Again the speedup is limited by the surface with the largest fraction of the hit points (11%).

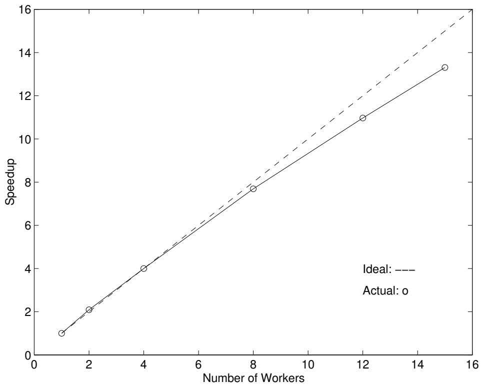

Figure 7: Parallel particle tracing speedup for the furniture warehouse. Execution time ranged from a maximum of 18h56m to a minimum of 1h29m.

dimensions.[8] The parallel performance results for the furniture warehouse are shown in Figures 7 and 8.

All solutions reported in this section were run on a cluster of 100MHz Hewlett-Packard 9000 PA-Risc 7100 workstations. The particle-tracing used 28 million particles per color channel[9].

For density-estimation and meshing, the room and warehouse environments were split into 100 pieces, and the Cornell box was split into 17 single-polygon pieces. The density-estimation constants C_1 and C_2 were set to 40 and 8 respectively[10]. With these parameter settings, the hit point file processing time, including partitioning, was no more than 1% of the total serial execution time, and therefore

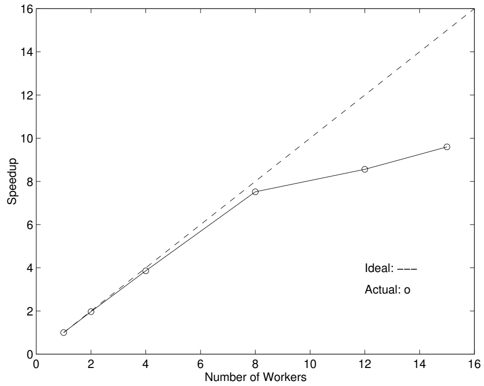

Figure 8: Parallel density estimation and meshing speedup for the furniture warehouse. Execution time ranged from a maximum of 9h59m to a minimum of 2h03m.

did not noticeably affect the parallel performance.

Additional results and further analysis of this parallel implementation are also available [34].

7 Conclusion and Future Work

We have presented a parallel version of the recently introduced Density-Estimation global illumination algorithm [23] that has a straightforward implementation and provides reasonably good parallel performance, at least on a small scale.

The most obvious bottleneck to larger scale parallelism is that a single surface may accumulate more than $1/n_w$ of the hit points, resulting in a lower bound on performance at the time required to process that surface. To eliminate this bottleneck, such surfaces need to be subdivided into parts which can be distributed to multiple workers and processed in parallel. Each part should be "expanded" to include nearby hit points on the other side of each subdivision edge, thus preventing the density estimation from introducing edge discontinuities. Decimation of these parts should initially preserve all mesh vertices lying on the subdivision edges; these vertices may then be decimated in a second pass that keeps the meshes on each side of a subdivision edge consistent.

Unlike previous radiosity methods which must fit the input environment surface description, the display meshes for each surface, and its intermediate data structures in virtual memory all at once, our parallel particle-tracing program stores only the surface descriptions in virtual memory (roughly 100 bytes per surface). This relatively low memory overhead makes it possible to compute view-independent global illumination solutions of input environments an order of magnitude larger than ever before, which effectively "bridges the gap" in environment size that has previously existed between conventional view-independent and view-dependent solutions.

Even so, it may still be desirable to enhance the parallel particle-tracing program to distribute the input surfaces of the environment throughout the parallel system. This would make it possible to compute solutions of even more complex environments (with millions of surfaces) which presently cannot be computed using a serial version of the particle tracer on a single workstation.

[8] A desirable side effect of this modeling technique is that no single surface accumulates a significantly large fraction of the hit points, which helps in load balancing the parallel density and meshing. An undesirable side effect is that this can introduce visible artifacts due to shading discontinuities or Z-buffer effects at some of the seams.

[9] A pilot study which used various number of particles (from 1 million to 40 million) found comparable speedup results, with execution time increasing quite linearly with the number of particles.

[10] See section 4.2 of our companion paper [23] for a description of these density-estimation constants.

Acknowledgments

Thanks to Gene Greger for modeling the Mackintosh room and warehouse scenes, to Frank Jary for assisting with the performance timings and graphs, and to Hurf Sheldon for system and network support and advice above and beyond the call of duty. This work was supported by the NSF/DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219), NSF CCR-9401961, and generous donations of equipment from the Hewlett-Packard Corporation.

References

[1] John M. Airey and Ming Ouh-Young. Two adaptive techniques let progressive radiosity outperform the traditional radiosity algorithm. Technical Report TR89-20, University of North Carolina at Chapel Hill, August 1989.

[2] Didier Badouel, Thierry Priol, and Kadi Bouatouch. Ray tracing on distributed memory parallel computers: Strategies for distributing computations and data. *Parallel Algorithms and Architectures for 3D Image Generation*, 1990. ACM Siggraph '90 Course 28 Notes.

[3] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. *Computer Graphics*, 23(3):325–334, July 1989. ACM Siggraph '89 Conference Proceedings.

[4] Daniel R. Baum and James M. Winget. Real time radiosity through parallel processing and hardware acceleration. *Computer Graphics*, 24(1):67–75, 1990. ACM Workshop on Interactive Graphics Proceedings.

[5] Kadi Bouatouch and Thierry Priol. Data management sceme for parallel radiosity. *Computer Aided Design*, 26(12):876–882, December 1994.

[6] Michael B. Carter. *Parallel Hierarchical Radiosity Rendering*. PhD thesis, Iowa State University, Department of Electrical Engineering and Computer Engineering, Ames, Iowa, November 1993.

[7] Çapin, Aykanat, and Özgüç. Progressive refinement radiosity on ring-connected multicomputers. In *Proceedings of the 1993 Symposium on Parallel Rendering in San Jose, California*, pages 71–76, 1993.

[8] Alan Chalmers and Derek Paddon. Parallel processing of progressive refinement radiosity methods. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[9] Shenchang Eric Chen. A progressive radiosity method and its implementation in a distributed processing environment. Master's thesis, Program of Computer Graphics, Cornell University, January 1989.

[10] Steven M. Drucker and Peter Schröder. Fast radiosity using a data parallel architecture. In *Proceedings of the Third Eurographics Workshop on Rendering*, pages 247–258, 1992.

[11] Martin Feda and Werner Purgathofer. Progressive refinement radiosity on a transputer network. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[12] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. PVM 3 user's guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, May 1994.

[13] Stuart A. Green and Derek J. Paddon. A highly flexible multiprocessor solution for ray tracing. *Visual Computer*, 6:62–73, 1990.

[14] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, July 1991. ACM Siggraph '91 Conference Proceedings.

[15] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics*, 24(3):145–154, August 1990. ACM Siggraph '90 Conference Proceedings.

[16] Wilfred Lefer. An efficient parallel ray tracing scheme for distributed memory parallel computers. In *Proceedings of the 1993 Symposium on Parallel Rendering in San Jose, California*, pages 77–88, 1993.

[17] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. *Computer Graphics*, pages 199–208, August 1993. ACM Siggraph '93 Conference Proceedings.

[18] Thomas J. V. Malley. A shading method for computer generated images. Master's thesis, University of Utah, June 1988.

[19] Derek Paddon and Alan Chalmers. Parallel processing of the radiosity method. *Computer Aided Design*, 26(12):917–927, December 1994.

[20] S. N. Pattanaik and S. P. Mudur. Computation of global illumination by Monte Carlo simulation of the particle model of light. *Third Eurographics Workshop on Rendering*, pages 71–83, May 1992.

[21] William Recker, David George, and Donald P. Greenberg. Acceleration techniques for progressive refinement radiosity. *Computer Graphics*, 24(1):59–66, 1990. ACM Workshop on Interactive Graphics Proceedings.

[22] Peter Shirley. A ray tracing method for illumination calculation in diffuse-specular scenes. In *Proceedings of Graphics Interface '90*, pages 205–212, May 1990.

[23] Peter Shirley, Bretton Wade, David Zareski, Philip Hubbard, Bruce Walter, and Donald P. Greenberg. Global illumination via density estimation. In *Proceedings of the Sixth Eurographics Workshop on Rendering*, pages 187–199, June 1995.

[24] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1985.

[25] Jaswinder P. Singh, Annop Gupta, and Marc Levoy. Parallel visualization algorithms: Performance and architectural implications. *IEEE Computer*, 27(7):45–55, July 1994.

[26] Jaswinder P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. L. Hennessy. Load balancing and data locality in hierarchical N-body methods. CSL-TR-92-505., Computer Systems Laboratory, Stanford University, 1992.

[27] Brian E. Smits, James R. Arvo, and Donald P. Greenberg. A clustering algorithm for radiosity in complex environments. *Computer Graphics*, 28(3):435–442, July 1994. ACM Siggraph '94 Conference Proceedings.

[28] J. C. Solem. A multiprocessor concept specialized to Monte Carlo. *Springer-Verlag Lecture Notes in Physics*, 240, 1985.

[29] Seth Teller, Celeste Fowler, Thomas Funkhouser, and Pat Hanrahan. Partitioning and ordering large radiosity calculations. *Computer Graphics*, 28(3):443–450, July 1994. ACM Siggraph '94 Conference Proceedings.

[30] Seth Teller and Pat Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 239–246, 1993.

[31] Amitabh Varshney and Jan F. Prins. An environment projection approach to radiosity for mesh-connected computers. In *Proceedings of the Third Eurographics Workshop on Rendering*, pages 271–281, 1992.

[32] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A ray tracing algorithm for progressive radiosity. *Computer Graphics*, 23(3):335–344, July 1989. ACM Siggraph '89 Conference Proceedings.

[33] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[34] David M. Zareski. Parallel decomposition of view-independent global illumination algorithms. Master's thesis, Program of Computer Graphics, Cornell University, expected 1995.