Scalability and Granularity Issues of the Hierarchical Radiosity Method

AXEL PODEHL THOMAS RAUBER GUDULA RÜNGER *

Computer Science Dep., Universität des Saarlandes, 66041 Saarbrücken, Germany

Abstract. The radiosity method is a global illumination method from computer graphics to visualize light in scenes of diffuse objects within an enclosure. The hierarchical radiosity method reduces the problem size considerably but results in a highly irregular algorithm which makes a parallel implementation more difficult. We investigate a task-oriented shared memory implementation and present optimizations with different effects concerning locality and granularity properties. As execution platform we use the SB-PRAM, a shared-memory machine with uniform memory access time, which allows us to concentrate on load balancing and scalability issues.

1 Introduction

The radiosity method is a simulation method from computer graphics to generate photo-realistic images of computer-generated three-dimensional environments with objects which diffusely reflect or emit light [4]. The method is based on the energy radiation between surfaces of objects and accounts for direct illumination and multiple reflections between the surfaces. First, all intensity values of an environment are determined in a view-independent stage, and then an image is computed using conventional visible-surface and interpolative shading algorithms.

The radiosity method uses a decomposition of the surfaces of objects in the scene into small elements, for each of which a radiosity value has to be computed. The radiosity values are the radiant energy per unit time and per unit area which are determined by solving a transport equation of energy, a system of linear equations describing the mutual interactions between radiosity energies of different surfaces with the help of geometric configuration factors. Building up and solving the transport equation causes high computational costs. For a reduction of the computational costs, a variety of methods have been proposed, including an adaptive refinement technique [7], hierarchical methods [6], or progressive methods [2]. Adaptive and hierarchical methods reduce the number configuration factors to be computed by combining several mutual dependencies. The progressive method reduces the costs of solving the linear equation system. A further reduction of computation time can be achieved by parallel implementations [9].

The efficient computational technique of the hierarchical radiosity method is achieved by computing the mutual illumination of surfaces more precisely only for short distances and less precisely for far surfaces. The mutual influence decays with the square of the distance and, thus, a uniform accuracy is achieved. This

* author supported by DFG

results in a smaller number of radiosity values to be computed and a reduction of interactions which can be computed efficiently on the hierarchical data structure supporting the hierarchical method.

In this article, we investigate the implementation of the hierarchical radiosity method on shared memory machines. The starting point of the investigation is the SPLASH-2 benchmark implementation [11] described in [9] which is tuned towards an execution on a cache-based virtual shared-memory machine with a physically distributed memory (Stanford DASH). In this implementation load balance is realized by using distributed task queues with task stealing. This competes with the locality of accesses to the task queues, because each attempt to steal a task includes an access to a remote task queue.

Usually, the competition between load balance (and granularity) and data locality hinders a concise study for scalability. This limitation vanishes when using an execution platform like the SB-PRAM providing a large number of processors and a global shared memory with unit access time [1]. Thus, the implementation can concentrate on the efficient exploitation of the task granularity and can neglect effects of locality. The original implementation is optimized on the algorithmic level, on the design level for tasks (towards a finer granularity), and on the task administration level. The optimized version is derived from the SPLASH-2 implementation by several optimization steps improving the exploitation of the degree of parallelism. Both implementations have good speedup values on the SB-PRAM for a small number of processors, exceeding the speedup values on the DASH. The optimized version exhibits good speedup values also for large numbers of processors (up to 2048).

The remainder of the paper is organized as follows. Section 2 summarizes the classical and the hierarchical radiosity methods. Section 3 describes the shared memory implementation. Section 4 discusses the experiments and Section 5 concludes.

2 The hierarchical radiosity method

The classical radiosity method starts with the subdivision of the input polygons (representing the surfaces of the objects in the scene) into a number of small patches with area A_j, $j = 1, \ldots, n$. For each of the patches, a *radiosity value* B_j (of dimension [watt/m^2]) is computed which describes the specific radiant energy per unit time and per unit area of A_j. Because of the mutual illumination of diffuse objects, a radiosity value B_j is composed of two parts: the emission energy per unit area E_j and the reflections of light that is incident on patch j from all other visible patches not occluded by patches in between. The light incident from patch i on patch j is a portion $H_i = B_i F_{ij}$ of the radiosity B_i; the dimensionless *configuration factor* or *form factor*) F_{ij} describes the fraction of lightening from patch i incident to patch j. Each form factor is a double integral depending only on the geometric constellation of the two elements i and j. Using the symmetry relation $F_{ij}A_i = F_{ji}A_j$, and diffuse reflectivity factors ρ_j, the unknown radiosity values B_j can be specified by a linear system of equations (see [3]):

$$B_j = E_j + \rho_j \sum_{i=1}^{n} F_{ji} B_i, \quad j = 1, \ldots, n. \tag{1}$$

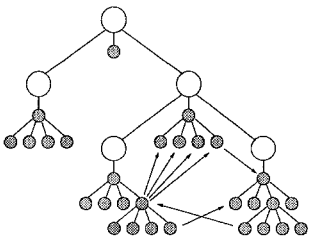

Fig. 1. Mixed BSP-quad-tree data structure: the large circles represent nodes of the BSP tree, the small shaded circles represent nodes of the quadtrees. The arrows show interactions between nodes of different quadtrees.

System (1) is solved by an iterative solution method for linear equation systems like the Jacobi method. But the main computational effort of the radiosity method consists in computing the $n(n-1)/2$ configuration factors.

The *hierarchical* radiosity method [6] reduces the computational effort by adaptively subdividing polygons into a hierarchy of smaller pieces of the surfaces. For each input polygon the subdivision into the hierarchy is organized in a quadtree such that the four children of a node represent a partition of the patch attached to the parents node. The input polygons themselves are organized in a BSP tree (binary space partitioning tree) where each node of the BSP tree represents the root of the corresponding quad-tree (see Figure 1). The union of the patches attached to leaves of all quad-trees of this data structure now represent the patches A_i, $i = 1, \ldots, n$, of the scene.

The algorithm starts by building up the BSP-tree in an initialization step. During the computation of the form factors, the quad-trees are built up adaptively in order to guarantee the computations to be of sufficient precision.

For each patch, the decision about the set of interacting elements with which the energy exchange is computed depends on an a-priori estimation of the influence of the radiosity. The method computes the energy transport (i.e., the configuration factor) between two patches only if it is not too large; otherwise the patches are subdivided. The subdivision of patches is performed if the energy transport between two patches is not small enough, i.e., $V_{ij}F_{ij}B_j > \mathrm{BF}_\epsilon$. When the radiant surface A_j is larger (i.e., $A_j > A_i$) then this area A_j is divided and the interaction list are adapted appropriately. The patch A_i is deleted from the interaction list of A_j and is inserted in the interaction lists of the children of A_j. If $A_j < A_i$ then A_i is divided and the children of A_i are inserted into the interaction list of A_j. The division of patches is stopped such that the patches are not smaller than A_ϵ. This procedure is performed for all pairs of input polygons and because all interactions use the same quadtree, each patch in the tree has its individual set of interaction patches for which the configuration factors have to be computed. Thus, the computation of radiosity for a leaf element may use configuration factors with internal elements of the quadtree unifying all the interaction with nodes in the corresponding subtree (representing a further division of the node element).

The a-priori-estimations of the energy transport take into account the configuration factors, the radiosity values and the visibility values. The use of radiosity values is possible because the hierarchical method alternates iteration steps of the Jacobi method to solve the energy system (1) with a re-computation of the quad-tree and the interaction sets based on estimations of $V_{ji}F_{ji}B_i$ with radiosity values B_i from the last step. The values $V_{ji}F_{ji}B_i$ are estimated and must not be larger than a bound BF_ϵ. The iteration stops when the difference of the total radiosity of two successive iterations is small enough.

In this paper, we consider a version of the radiosity method described in

[6] and adopted in the implementation of the SPLASH-2 benchmark suite [11] with the following methods: The form factors are approximated numerically by a ray tracing method proposed in [10] using form factors from four points of A_i (which are points from pieces which form a partition of A_i) to four small disks A_{D_l}, $l = 1, \ldots, 4$, covering A_j. The visibility test proposed in [6] uses a ray tracing method with a fixed number of rays between two patches. Here we use rays from 16 sub-elements to 16 sub-elements The percentage of rays not blocked by intervening surfaces is used as visibility factor $V_{ij} \in [0, 1]$.

One iteration step of the Jacobian method consists in a top-down and a bottom-up traversal for each quadtree: In the top-down traversal, a radiosity value for each node is computed by computing the influence of all interaction partner $j \in I(patch)$ and adding the radiosities of the parents node. The leaves of the quadtrees also take into account the emission E_j. In the bottom-up traversal, each node computes area-weighted radiosity of the children in order to make new radiosity values available on every level of the tree for the estimations of energy exchange and the next iteration step.

Figure 2 shows the hierarchical radiosity methods already expressing explicitly the potential parallelism. Portions of the program that can be executed independently from each other in parallel are separated by a || sign. Loops with independent iterations are described by forall.

```
(1) do recursively: insert-polygon(p_1)
    with procedure insert-polygon(p_i) =
    {insert p_i into BSP tree;
      do in parallel {insert-polygon(p_{i+1}) || forall p_j, j < i, compute F_{p_j p_i}, F_{p_i p_j}}}
(2) Compute visibility factors V_{p_j p_i};
    do { forall polygons p ∈ {p_1, ..., p_k}
    do recursively: compute-interaction(root(p));
    with procedure compute-interaction(i) =
        { forall interactions j ∈ I(i) do {
          Compute visibility factor V_ij;
          if V_ij F_ij B_j > BF_ε and A_i, A_j > A_ε then {
            Divide (A_i, A_j);
            compute configuration factors of new interactions; }
          Compute B(i) = ρ_i Σ_{j∈I(i)} V_ij F_ij B(j) + B(parent(i));
          if i is leaf then {
            B(i) = B(i) + E(i);
            while i is last child of parent(i) do {
              i = parent(i);
              B(i) = 1/4 Σ_{j=child(i)} B(j); }
            else { forall children k of i do compute-interaction(k); } } } }
    while (ERROR > ε)
(3) forall elements in all quadtrees do { bilinear interpolation; }
```

Fig. 2. Hierarchical radiosity method with maximum degree of parallelism.

3 Parallel implementation

For the parallel implementation of the hierarchical radiosity method we used a task-oriented shared memory model and the SB-PRAM as execution platform. The SB-PRAM is a realization of a modified fluent machine [1]. A number p of physical processors has access to p memory modules each consisting of m memory cells. The processors are connected to the memory modules via a butterfly interconnection network. Thus, the memory is accessed as a virtual linear shared memory distributed among the modules. Besides the usual load and store operations to access memory cells, the SB-PRAM also offers *multiprefix* instructions which enable several processors to perform simple operations on a memory cell in parallel. The multiprefix instruction with addition MPADD starts with one local value for each processor and produces the sequence of prefix sums; initially one input value resides on one processors and at the end each processor holds one sum. A multiprefix operation is performed in two time units, independently of the number of participating processors. It is even possible that different groups of processors perform separate multiprefix operations in parallel. Multiprefix operations can be used for an efficient realization of access coordination to the global memory (*locking*) or parallel data structures like parallel task queues [5]. They can also be used for the implementation of a parallel loop which is controlled by a shared counter. The access to the counter by a multiprefix operation allows a dynamic execution of the loop iterations without sequentializations.

SPLASH implementation: The SPLASH2 benchmark suite comprises several parallel example programs realizing irregular applications, which are mainly intended for the Standorf DASH multiprocessor [8]. The multiprocessor is a cache coherent shared address space processor with physically distributed memory. The software simulator of the DASH (described in [9], [9]) simulates the non-uniform memory access with a constant access time for each level, i.e., cache, local memory and non-local memory or cache.

The parallel implementation of the hierarchical radiosity method realizes parallelism that occurs across input polygons, across the patches that a polygon is divided into, and across the interactions computed for a patch. This parallelism is reflected in the choice of tasks: The B-tasks and F-tasks realize parallelism in the first phase. The parallel computation of interactions and parallel BF-refinement are realized by R-tasks and V-tasks. The last phase uses A-tasks.

TASK NAME	COMPUTATIONS PERFORMED BY THE TASK
B-task(p)	insert input polygon p into BSP-tree and create B- and F-tasks
F-task(ij)	compute form factors F_{ij} and F_{ji} for input polygons i and j
R-task(p)	compute phase (2) for patch/element p except visibility factor
V-task(ij)	compute visibility factor V_{ij} between patches i and j
A-task(p)	create A-tasks for children of patch p and perform bilinear interpolation if p is an element

In the original SPLASH-2 implementation, each processor has its own task queue and inserts new tasks created by local tasks into this queue to maintain locality. But because of the dynamically changing hierarchical data structure, load balance cannot be achieved by a static assignment of tasks. To avoid a bad load balance, a processor is allowed to access task queues of the other processors, if its own queue is empty (*task stealing*) [11]. Concurrent accesses to the same data are avoided by the locking mechanism, e.g. when interactions between patches/elements are computed.

```
(1) do recursively: B-task(p_1)
    with task B-task(p_i) =
      { insert polygon p_i into BSP-tree;
        do in parallel { B-task(p_{i+1}); || forall p_j, j < i do F-task(ij)}}
(2) do { forall polygons p ∈ {p_1, ... p_k}
    do recursively: R-task(root(p));
      with task R-task(i) = {
        forall interactions j ∈ I(i) do {
          V-task(ij);
          hierarchical subdivision and computation of radiosity values;}
        forall children k of i do {
          compute configuration factors of new interactions;
          R-task(k);}}
    while (ERROR > ε )
(3) forall polygons p ∈ {p_1, ... p_k} do recursively: A-task(root(p));
    with task A-task(q) = { forall children r of q do A-task(r) }
```

Fig. 3. Task organization of the Splash implementation.

The SPLASH implementation is illustrated in a pseudo-code task-program in Figure 3. The pseudo language reflects the fact that tasks perform both computations and initiations of other tasks (similar to procedures in sequential programming calling other procedures). The B-tasks have to be executed sequentially. The initiation of data dependent F-tasks is expressed by a recursive call-structure using the keyword do recursively. The corresponding task is defined by a with task statement having a recursive structure due to the hierarchical tree structure used in the algorithm. A possible schedule of B-tasks and F-tasks on 4 processors is depicted in Figure 5 on the left. The independence of computations on different quad-trees in phase (2) is expressed by a forall construct. The interactions within each quadtree are performed recursively according to the tree structure determining data dependencies between R-tasks. The visibility V-tasks for children are initiated by the parents R-task. The last phase creates a hierarchy of A-tasks of which only the leaf-tasks perform the bilinear interpolation.

SB-PRAM implementation: The SB-PRAM implementation uses the SPLASH implementation as starting point. Optimizations of the parallel implementation include a parallel construction of the BSP tree, the use of a parallel task queue, and the use of parallel loops where locality can be ignored. Table ?? summarize the modifications.

The BSP tree is constructed by a parallel search over the polygon tree in contrast to a sequential construction in order to reduce the sequential parts of the computation. In the second phase, the tasks to compute interactions (one task for each input element) do not offer enough parallelism for a large number of processors. In the first iteration step the number of tasks cannot be increased. But because these interactions all take place on the same level, this phase is separated from the rest of the iteration and the mutual configuration factors are solved with a parallel loop. Moreover, the symmetry of the configuration factors is exploited. In all following iterations, the computation of configuration

PHASE	MODIFICATION	SPECIFIC MODIFICATION	ADVANTAGES
1	algorithmic	redefined B'-tasks for building BSP-tree	larger potential parallelism; worthwhile for large numbers of processors;
1, 2	algorithmic and task design	parallel loop for combined initial interactions V_{pq}, F_{pq}	creation of regular `forall`-loop realized by *parallel loop*; exploitation of form-factor symmetries is possible
2	task design	modified V'-tasks for computing V_{pq}, F_{pq}	refinement of granularity
3	implementation	*parallel loop* for bilinear interpolation	task-administration for `forall`-loop is avoided
1-3	software support	task allocation	parallel queue avoids failures in task stealing

Table 1. Optimizations for an efficient SB-PRAM implementation

factors and the visibility values are completely moved to lower levels in the quadtrees thus creating a high degree of parallelism. The tasks for the final bilinear interpolation for smoothing the solution are also executed in a parallel loop over the leaf elements. This strategy replaces the version where all internal patch nodes were involved in creating tasks for their child nodes for locality reasons. The modified tasks are:

TASK NAME	COMPUTATIONS PERFORMED BY THE TASK
B'-task(p)	insert polygon p into BSP-tree and build sublists of elements
F'-comp.(ij)	compute form factors F_{ij}, F_{ji} and visibility factors V_{ij}, V_{ji} for input polygons i and j
R'-task(p)	compute phase (2) for element/patch except form and visibility factors
V'-task(ij)	compute visibility factor V_{ij} and form factor F_{ij} between patches/elements i and j
A'-comp.(p)	compute bilinear interpolation for element p

The corresponding task program is given in Figure 4 showing four instead of three phases which are still separated by synchronization points. The implementation supports the use of parallel loops and increases the granularity. A possible schedule on 4 processors is given in Figure 5 on the right.

4 Experiments

Figure 6 (left) shows the speedup values of the original SPLASH implementation on the DASH (as reported in [9]) and on the SB-PRAM simulators for the SPLASH test scene (with 346 input polygons). Due to the task stealing mechanism the DASH reaches the best efficiency with a coarser granularity where the V-tasks are chosen to compute four visibility values instead of one; this causes locality advantages on the BSP-tree data structure (DASH(default) in Figure 6 left). In contrast, the SB-PRAM achieves a better speedup when a finer granularity with one visibility computation per V-task is chosen (SB-PRAM(finest) in Figure 6 left). The original SPLASH implementation performs better on the SB-PRAM (SB-PRAM(default) in Figure 6 left) than on the DASH because of unit memory access times and the redundance of locality.

```
(1') do recursively: B'-task(p_1, ..p_k);
       with task B'-task(p_1, ..p_k) = {
         insert p_1 in BSP-tree;
         built lists of polygons (q_1, ..q_l) visible and (r_1, ..r_m) invisible from p_1;
         do in parallel {B'-task(q_1, ..q_l) || B'-task(r_1, ..r_m) } }
(1") forall i, j = 1, ..., k with i < j do F'-computation(ij);
(2') do { forall polygons p ∈ {p_1, ... p_k}
     do recursively: R'-task(root(p));
       with task R'-task(i) = {
         forall interactions j ∈ I(i) do {
           V'-task(i,j);
           hierarchical subdivision and computation of radiosity values;}
         forall children k of i do R'-task(k);}
     while (ERROR > ε )
(3') forall elements p in all quadtrees do A'-computations(p)
```

Fig. 4. Task organization for the SB-PRAM implementation

The large number of processors and the additional software support (see Section 3) makes the SB-PRAM to an ideal platform to study the scalability properties inherent in an algorithms. But the massive parallelism and the uniform access time require a different implementation strategy for the study of the scalability of the hierarchical radiosity method. The optimizations described in Section 3 take this modified concept into account by replacing the expensive task concept by *parallel loops* when loops exhibit a regular, independent parallel structure (F'-computations, A'-computations), decreasing the granularity (new V'-tasks and R'-tasks with smaller runtime), increasing the degree of potential parallelism by destroying data locality (B'tasks), and exploiting a new task-administration concept by using unit access time task-queues.

Figure 6 (right) shows the speedup values for the original SPLASH implementation and the optimized implementation on the SB-PRAM simulator for up to 2048 processors. The SPLASH implementation does not scale well for more than 256 processors. For 1024 processors the optimized version still has an efficiency of 0.742. Table 7 reports the speedup values S_{1024} and the absolute runtimes T_{1024} for both implementations of the SPLAH test scene. The values for the SPLASH implementation with phases (1), (2), (3) are on the left; the values for the optimized version with phases (1'), (1"), (2'), (3') are on the right. The phases overlap according to the algorithmic structure. The timings in columns T_1 left and right show that the optimized version performs even better in the sequential case (5.8 %).

The efficient parallelization of the B-tasks (BSP-tree) is more important for massively parallel implementations than for a relatively small number of pro-

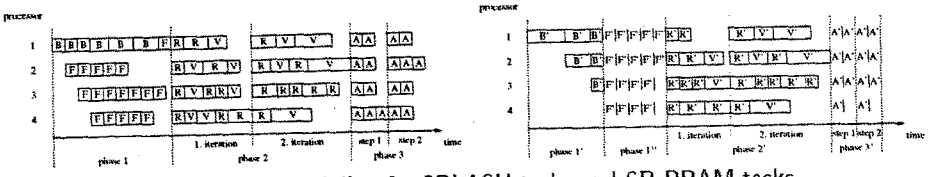

Fig. 5. Task scheduling for SPLASH tasks and SB-PRAM tasks

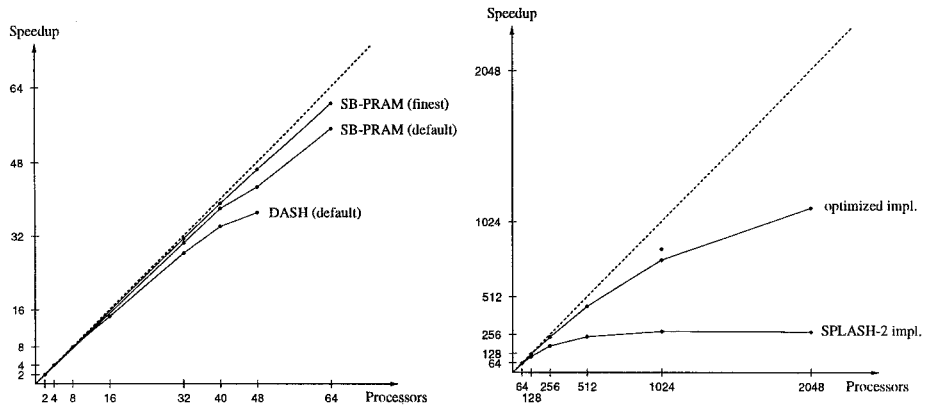

Fig. 6. Speedup values on the SB-PRAM and on the DASH. Left diagram: original SPLASH-2 implementation on the DASH and the SB-PRAM with default granularity of the tasks and the finest granularity that is possible. Right diagram: default granularity finest granularity on the SB-PRAM for larger number of processors.

cessors where the fraction $\frac{1}{14000}$ of the computation for building the BSP-tree is neglectable. Moreover, the prephase for sorting the input polygons is done before building the BSP-tree. Experiments have shown that the global execution time of the entire algorithms depends significantly on the order of the input polygons; the time varies by a factor of up to 10. This is a general phenomenon which should be separated from issues of parallelization.

A different phenomenon concern the convergence of the iteration steps solving the system of radiosity values (1). The iteration may converge faster in the parallel implementation than in the sequential one thus saving a full iteration step. The reason is the use of updated values within one iteration step if the number of patches is greater than the number of processors. The faster convergence corresponds to the faster convergence in the Gauss-seidel method for solving linear systems. This effect is exploited in the (non-hierarchical) progressive radiosity method [2]. Besides the improvements of the efficiency our investigations show that the optimized version leads to a much simpler source code. The main reasons are the lack of locality, the use of *parallel loops* which corresponds to the loops in the pseudocode algorithms, and the simplified task administration.

SPLASH2 implementation					**optimized implementation**				
	Phase	T_1	T_{1024}	S_{1024}	S_{1024}	T_{1024}	T_1	Phase	
dimension		sec	sec			sec	sec		
BSP and form factor	1	6898.7	104.6	66.0	1.44	5.409	7.805	1'	BSP
					876.3	7.413	6496.2	1"	root inter.
iteration	2	90726	244.1	371.7	791.5	108.0	85487	2'	iteration.
bil.interp.	3	33.782	4.225	8.0	814.9	0.036	29.336	3'	bil.interp.
total		97658	353.2	276.5	760.5	121.0	92021		total

Fig. 7. Timings of the SPLASH2 and the optimized implementation on the SB-PRAM. The input scene is the SPLASH2 test scene.

5 Conclusions

We have presented a task oriented shared memory implementation for the hierarchical radiosity method. The main interest was to investigate the scalability issues of the method. The SB-PRAM with uniform access time offers a good platform to study efficient implementations and scalability properties for irregular problems because the locality properties of the applications do not influence the resulting performance and the investigations can concentrate on the maximum degree of parallelism. The experiments have shown that an implementation designed for up to 64 processors is not suitable to achieve good speedups on a large number of processors. A redesign of the algorithms provides a large number of independent tasks. The means are regular forall-loops and an decrease of the granularity in the phases realizing the interactions between different surfaces. The resulting parallel algorithms shows good speedup for up to 2048 processors. Thus, the hierarchical radiosity methods can be implemented efficiently although it has highly irregular computation and access patterns. Moreover, the investigations have shown that parallel data structures provided by the underlying machine can support massively parallel, efficient implementations of highly irregular algorithms.

References

1. F. Abolhassan, J. Keller, and W.J. Paul. On the Cost–Effectiveness of PRAMs. In *Proceeding of the 3rd IEEE Symposium on Parallel and Distributed Processing*, pages 2–9, 1991.
2. Micheal Cohen, Shenchang Chen, John Wallace, and Donald Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics*, 22(4):75–84, 1988. Proceedings of the SIGGRAPH '88.
3. James Foley, Adries van Dam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice.* Addison–Wesley, Reading, USA, 1990.
4. Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennet Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18((3):212–222, 1984. Proceedings of the SIGGRAPH '84.
5. Th. Grün, Th. Rauber, and J. Röhrig. The programming environment of the SB-PRAM. In *Proc. 7th IASTED/ISMM Int.l Conf. on Parallel and Distributed Computing and Systems, Washington DC*, pages 504–509, October 1995.
6. Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 1991.
7. Paul S. Heckbert. *Simulating Global Illumination using Adaptive Meshing.* PhD thesis, university of California, Berkeley, 1991.
8. Daniel Lenoski, James Laudon, Truman Joe David Nakahira, Luis Stevens, Anoop Gupta, and John Hennessy. The dash prototype: Logic overhead and performance. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):41–61, 1993.
9. J.P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-hut, fast multipole, and radiosity. *Journal of Parallel and Distributed Computing*, 27:118–141, 1995.
10. John Wallace, Kells Elmquist, and Eric Haines. A ray tracing algorithm for progressive radiosity. *Computer Graphics*, 23(3):315–324, 1989. Proceedings of the SIGGRAPH '89.
11. S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *ISCA'95*, pages 24–36, 1995.