# A Parallel Approach to Hierarchical Radiosity

**Christian-A. Bohn**[†]

[†]German National Research Center
for Computer Science
Dept. Visualization and Media Systems Design
Sankt Augustin, Germany

**Robert Garmann**[‡]

[‡]University of Dortmund
The Computer Graphics Group
Departement of Computer Science
Dortmund, Germany

## Abstract

A parallel algorithm solving the radiosity equation is presented. It is based on the hierarchical approach (HR) [5] and realized on a massively parallel supercomputer — the *ConnectionMachine 5*.

Our algorithm considers the HR approach as a process that manipulates an huge graph structure. Simulated annealing is used in the graph's rearranging procedure to achieve a good work-balance and nearly optimal communication costs.

The implementation shows a significant step to facilitate the application of a radiosity solver, produced on one hand by the few user-support that HR needs, on the other hand by the fast calculation times the parallel implementation offers. On 64 processors we obtained a speed-up of 8.4.

**Keywords:** Computer Graphics, Scientific Visualization, Hierarchical, Radiosity, Rendering, Simulation, Parallelization, Message Passing, Supercomputer, Architecture

# 1 Introduction

## 1.1 Global Illumination

An important challenge in the field of computing pictures from three dimensional polygon data is the simulation of the physics of the global light flow — the *global illumination*. The essence is to find mathematical models which imitate the physics of light accurately and deliver algorithms that are realizable on recent computer technology.

The theory of algorithms concerning the global illumination problem may be describted by the *rendering equation* [7]. As this equation is too complex to evaluate directly, certain elements need to be neglected. This should lead to more tractable formulations, providing models which nevertheless are able to deliver adequate simulation results, like realistic looking images.

One of these short cuts can be unified under the class of *radiosity* approaches, developed in the last ten years. They have basically in common that only Lambertian diffuse reflection is taken into account. The rendering equation, which defines the light flow as a three-point relation is shortened to a two-point type — 'the reflected ray from an arbitrary point doesn't depend directly on the direction of the rays received at this point'.

Solving the purely diffuse global illumination problem means solving

$$B(x) = E(x) + \rho \int_S B(x')G(x,x')dA' \quad (1)$$

— the *radiosity equation*, where the outgoing intensity $B$ at a point $x$ is defined from the integral over the incoming energy leaving from all surfaces $S$, multiplied by the diffuse reflection coefficient $\rho$ at this point. The integral is built from a term that expresses the portion of light sent from another point $x'$ to $x$ ($G$) multiplied by the outgoing intensity at $x'$.

Calculating the equation is either done by *Monte-Carlo* techniques or by discretizing the scene into $n$ finite elements [6]. Projecting the radiosity equation into a set of basis functions delivers a system of linear equations. The coefficients of this system ($G$) are based on the geometrical relationships of the scene and determine the transport of light between elements [3]. The unknowns are *nodal values* at each element, which define the resulting intensity for these elements.

The selection of basis functions is one way of classifying the different methods developed for solving the radiosity equation. For a rough classification, see [4]. For a more specific explanation see [3], or [2, 13].

[†]bohn@viswiz.gmd.de
[‡]garmann@ls7.informatik.uni-dortmund.de

### 1.2 Hierarchical Radiosity

Non-hierarchical radiosity approaches calculate the flow of light by simulating its transport between all the elements of the scene, qualified by the matrix $G$ from (1). For a number $n$ of elements there are $n^2$ interactions to be taken into account. The idea of hierarchical radiosity is borrowed from the *N-body* problem [5]. For example, consider two pairs of elements that are positioned far away. As the four components in $G$ are nearly the same, they may communicate through one common path. Their components can be combined to one interaction, as one block in the matrix $G$. The goal of hierarchical radiosity is to recognize different levels of communication. The granularity of comunication of one surface may vary over the different surfaces.

In the following the algorithm is discussed from a practical point of view. For a more detailed explanation in the sense of basis functions, see [4, 11].

The matrix itself is not of interest, *links* (interactions) between nodes of elements, which have effect on groups of elements (the hierarchy), have to be built. These links refer to blocks in the matrix $G$. As an alternative to building up the matrix of all elements, and to group its components into blocks, the following method is presented, which creates the links on the fly. The all over time- and space-complexity is of order $O(m)$ in common cases [5], where $m$ is the number of the final elements.

**Algorithm**

Consider elements with constant light intensity. The effect of a patch $j$ on another patch $i$ with sizes $A_j$, $A_i$ can be approximated by the point-to-disk *formfactor* $F_{ij}$ (equation 2) [12].

$$F_{ij} = H_{ij} \cdot \frac{\cos\theta_i \cos\theta_j A_j}{\pi r_{ij}^2 + A_i} \tag{2}$$

where the $\theta$ are the angles between the element normals and the connections of their centers, with distance $r_{ij}$ (see figure 1). $H_{ij}$ determines the percentage of area $j$, which is visible to area $i$. The overall effect of area $j$ on area $i$ is built by multiplying $F_{ij}$ by the outgoing energy from area $j$ ($B_j$).

**Subdivision** — In case of a large energy transport ($F_{ij}B_j$) through one link, it should be replaced by new links to compensate the error because of a more accurate modeling of the light flow. Even new elements may arise from this subdivision. The essence of the hierarchical approach is that these new patches don't necessarily have to interact with all other elements in the scene. Only the links determine where the light flow has to be taken into account.

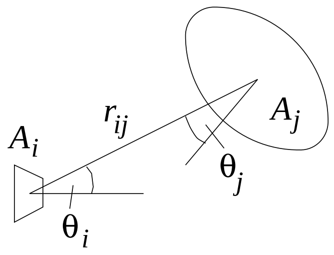


Figure 1: The Formfactor

At the beginning of the whole algorithm each patch may be connected to each other by intervening links, if they are not occluded. Subdivision enlarges this initial graph to a common graph with patches and links as nodes and connections as interactions between patches and links and between the elements of the subdivision hierarchy of each surface.

**Simulation** — Subdivision creates an element hierarchy et each surface, a tree-like structure. The simulation of the light through the links — the solving of the radiosity equation — requires following transition through each element tree (*push-pull*). It must be done after each propagation (*shooting*) of energy from the patches before repropagating it into the scene. Energy received at higher levels must be led downwards through the tree, and added to the passed elements. The same holds for energy coming in at lower levels, which must be accumulated up into higher levels.

These two steps (subdividing/simulation and push-pull) have to be repeated until the system converges, that is either the maximum of the transferred energy through the links ($F_{ij}B_j$), or the maximum energy propagated through the patch tree falls below certain error boundaries $BF_e$ or $B_e$ resp.

**Multigridding** — The above process is started again with a lower error boundary of the mentioned energies. The solution calculated in the preceding steps is used. The cycles are repeated until a global minimum of $F_{ij}B_j$ is reached. See figure 3.

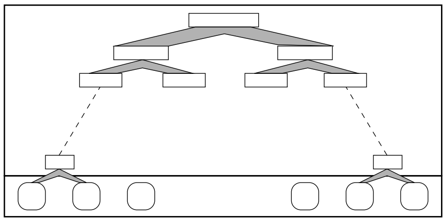


Figure 2: Hardware Architecture

# 2 Parallelization

## 2.1 Hardware

Parallelization of algorithms strongly depends on the underlying hardware architecture. The *ConnectionMachine 5* (*ThinkingMachines Corp.*) serves for this work. The architecture is of the MIMD-type (Multiple Instruction Multiple Data). Several independent processors (*SPARC 2*) with own their local memory (32 MBytes) work together. Communication is done by sending messages to specific or all processing units (*Message Passing*). Application of the message passing principle delivers a scheme, that is easy adaptable to other MIMD computers. The architecture is scalable up to 16384 processors. For this work a 64 processors *CM-5* was used.

The processors are organized into a tree structure of connection nodes. The bandwidth of the network is 5 MByte/sec at the leave nodes — the processors, and increases in the direction of the root node (*fattree*). Thus, the architecture may be seen as a network of fully connected processors with nearly equal communication power on each connection (see figure 2).

The CM-5 is connected to an *Onyx* workstation (*SiliconGraphics*) which serves as front-end computer for steering the CM-5 in the background, and for displaying the scene and the calculated results.

## 2.2 The Algorithm

Parallelization on a MIMD architecture is achieved by partitioning the given algorithm into independent tasks, which are assigned to single processing units. The purpose is, that all processors should have a job at each time segment, none of them should be idle anytime. Certain constraints have to be taken into account, the limited local memory and the communication costs between tasks on different processors.

We can think of the tasks as nodes of an undirected graph. Communication between the tasks is mimicked by the connections between the corresponding nodes. Nodes and connections are weighted by the amount of time and the costs of communication respectively to fulfill the whole job.

Instead of tasks one may assign data items to the graph's nodes. Then every single task is mapped to a specific data item. For the parallelization of the HR algorithm we assign the scene's patches and the links to single nodes of a graph. The connections include the subdivision hierarchies of each surface. Besides, every link is connected to its two corresponding patches.

We map the task of subdividing a link and the task of shooting energy along some link to the link's data item. The push-pull operation on a single patch is mapped to the patch's data item.

Now the HR algorithm can be thought of as a manipulation of a graph. The execution performance depends on the actual distribution of the graph's nodes over the processors. For an efficient computation, tasks must be shared equally on the processors, the communication rate should be kept as low as possible. So the weights of the nodes must be equalized, while the weights of connections must be minimized. Formalization of this problem is well known as the *graph-partitioning-problem* (GPP).
Let's define the following terminology:
A <u>surface</u> or <u>blocker</u> is a geometrical element which appears in the initial scene. A <u>patch</u> is a unit on which the simulation bases — energy is simulated by its effect on other patches. A <u>patchtree</u> is built from a surface and contains patches. From the <u>toppatch</u>, the root, grow the the subdivided patches (elements). The elements of the patchree (the subdivision hierarchy for one surface) are connected by <u>branches</u>. The flow of light from one patch to another is led through a <u>link</u>.

Figure 3 shows the whole algorithm. In the following a detailed look on the various stages is done.

### 2.2.1 Graph Initialization {1}

From $n$ surfaces, which define the geometry of the scene, $O(n^2)$ inital interactions (links) are computed and connected to the corresponding patches. The links are distributed equally on the single processors. Each processor contains a list of the surfaces. This garuantees fast access for visibility calculations between elements later on. A subset of the blockers (a *candidate list*) is stored for each link, and a formfactor estimation is done.

```
{0}  input BF_low
{1}  initialize_graph()
{2}  set BF_e = BF_low
{3}  repeat
{4}      repeat
{5}          if (graph not balanced)
{6}              graph_balance()
{7}          repeat
{8}              if (link BF > BF_e)
{9}                 subdivide_link()
{10}                add_new_links_to_graph()
{11}             else
{12}                do_link()
{13}         until (all links are done)
{14}         push_pull()
{15}         render()
{16}     until (converged)
{17}     decrease BF_e
{18} until (BF_e < BF_low)
```

Figure 3: The Algorithm

#### 2.2.2 Subdivision, Shooting {7..13}

If $BF$ of a link is too strong, it needs to be subdivided. Four new links and four new patches are created. The old link is deleted. The patch with the larger area is subdivided, if its area is above a certain boundary $A_e$. Subdivision may be demanded repeatedly until $BF$ reaches a low limit `BF_e`. To minimize communication between processors, the new elements are stored locally until the final subdivision hierarchy for a patch is reached. Then `do_link()` does the shooting of the energy through the final links.

Information for a local task may lie on other processors. These can be the patch geometry for subdividing the links and patches, the energy after propagation through links, or the element hierarchy, which has to be updated for a local patch. Data requests are sent to get these informations.

**Data requests** — Sending and receiving data from other processing units requires a large amount of time. The processors should not wait for a communication result. Consequently the whole algorithm is divided into subtasks, which can be completely executed without getting further information from other units. All data from other processors is fetched before their start by sending requests to other units. In this time the subtask is stored on a local stack until the data has been received, and other jobs can be started. A scheduler manages the local stack and calls subtasks depending on incoming messages and using the local subgraph.

**Scheduling** — Let's define the following subtasks:
task test judges on the refine-decision of a link, refine performs the subdivision of a link. the task interact exchanges energy between two patches. Communication is provided by the task search, which looks for the location of data and sends requests to other processors, if needed. fetch catches data arrived from other units and assigns them to its local requester. If the information is complete, it starts the task, which was waiting for it.

For communication count the request-, answer- and store-tasks. There are two approaches for determining the priority of executing them, the productive and the economical way. The first one strives for fast execution on the processors and tries to do the request-messages first. The second one manages the store- and answer-messages first, which almost update local data, such that stacks are tidied, and messages in the network are limited. The size of the local stacks are the criterion for switching between the two approaches.

#### 2.2.3 Push-Pull {14}

After the subdivision is done all $BF$ have values below $BF_e$ and shooting is done once. To repeat this task, the energies have to be propagated through the patch hierarchies.

To update the energy in between a patchtree, it has to be traversed in deepest-descent order. While one processor is working on a certain local part of the tree, it could happen that it has to wait for the complete traversal of another lower subtree which is localized on another processor. As there are most likely more subtrees than processors, deadlocks may arise, if a requested subtree again lies on the same processor. To avoid those deadlocks and to achieve higher performance again we use the principle of data requests and scheduling.

**Data Requests** — If data from another processor is needed to manage a task, a request is sent. Instead of waiting, the task is put to a local stack and other tasks are started, like starting traversals of different hierarchies. The work on each surface hierarchy can be seen as building Euler-circles, where single segments are done on different processors.

**Scheduling** — There are the following principle tasks which can be finished without further communication if the data is available:
push-messages ask for a traversal of a subtree on another processor. If patches of this subtree are locally existent, they are moved on the wait-for-push-stack. If a traversal is completed, the result is

stored on the wait-for-pull-stack and a pull-message is sent to the requesting processor The priority of pull-messages and and wait-for-pull-tasks is the highest, because "open" Euler-circles should be finished as fast as possible. This delimits the number of messages in network.

### 2.2.4 Relaxing the inner loop

While application of the algorithm from figure 3 it could be observed that the rigid scheme of the inner loop ({4..16}) interferes with fast convergence. The reason for repeated execution of the inner loop with decreasing values of `BF_e` is the avoidance of very unbalanced graphs and the acceleration of the convergence of the simulation. But, starting with many patches to refine, at the end there are only few links that don't fulfill the $BF$ criterion; this prevents fast convergence of the whole algorithm.

So, $BF_e$ is decreased on the fly if the following conditions hold: The number of links that are newly generated goes beyond a certain threshold and the maximum intensity change during step {14} is below a certain $B_{max}$ which in turn is greater than `B_e`. The loop condition in {16} changes to `(approx. converged)`, so $BF_e$ is decreased faster. An additional condition `(converged)` in step {18} will ensure overall strict convergence.

### 2.2.5 Graph Balance {5,6}

During subdivision the graph is modified, and its assignment on the processor network may not be optimal for the execution of the algorithm. As this problem is very complex to solve, some simplifications are introduced.

- Let's assume that a good graph at a certain time is also a sufficient solution for an efficient continuation of the algorithm.
- As mentioned, finding a balanced graph is of the same complexity as the GPP problem, a suboptimal solution must suffice.

**GPP — the Problem —** Given is a graph $G = (V, E)$ with connection weights $a_{vw} \in \mathrm{IR}_0^+$ and node weights $g_v \in \mathrm{IR}_0^+$. The problem is to find a disjoint partition of $V$ into $p$ subsets $V_1 \ldots V_p$, such that

$$C = \frac{1}{2} \sum_{1 \le i \le p} C_i \qquad (3)$$

is minimized and $W_1 = \cdots = W_p$ holds. $C_i$ is the sum of the connection weights between $V_i$ and all other $V_*$, and $W_i$ is the sum of the node weights in $V_i$. As we want an approximation of the optimal solution, the equality of the $W_i$ may be subsided to

$$W_1 \approx \ldots \approx W_p \qquad (4)$$

Recall that the $V_i$ and the connection weights correspond with the amount of work in the processing units and the costs of communication respectively.

**GPP — the Solution —** A combination of *simulated annealing* [9] and *clustering* is used in this work. The equation

$$Z = \max_{1 \le i \le p} (\alpha C_i + (1 - \alpha) W_i) \qquad (5)$$

which is a built by combination of (3) and (4) to one term, must be minimized by simulated anealing. $\alpha$ is the relative importance of a good balancing compared with cheap communication. Depending on the size of the graph to be calculated on, a clustering precedes the simulated annealing. It is accomplished in the following manner: Starting at arbitrary nodes the graph is traversed on connections with maximum weight. All passed nodes are added to one cluster until a certain boundary is reached. The resulting clusters are connected by the sum of their connection weights, node weights of a cluster are the sum of the inner node weights. Now, simulated annealing is first applied on the new clustered graph.

Simulated annealing tries to minimize the cost function (5). It is accomplished by generating new configurations of the problem (new graphs) and calculating its expense. Depending on these costs a configuration may be accepted or not. The criterion for acceptance is important. The *Metropolis*-algorithm [10] even allows for acceptance of worse solutions with a probability that is decreased over the whole process — the annealing.

The implementation in parallel is done by computing new configurations on each single processor. The information of the graph is placed on each processor in a coded (compressed) form. The units work indepently for a certain time, and decide the acceptance of new configurations by their own. After this sequence a certain processor selects one solution from all existing solutions. This is sent to all other processors and defines new common configurations on which the processes continue their search [1].

The graph balancing is only executed if there is need for it. To become aware of this, the cost function could be calculated, but this is too expensive; instead the idle time of the processors in

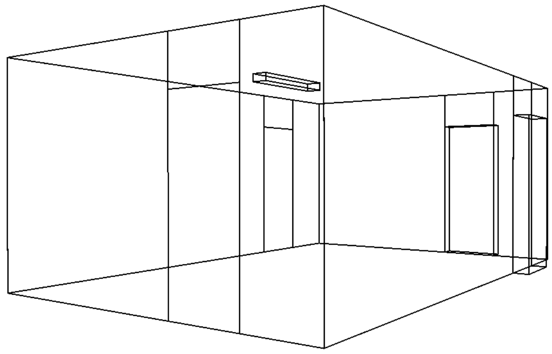

Figure 4: "room" (27 polygons)

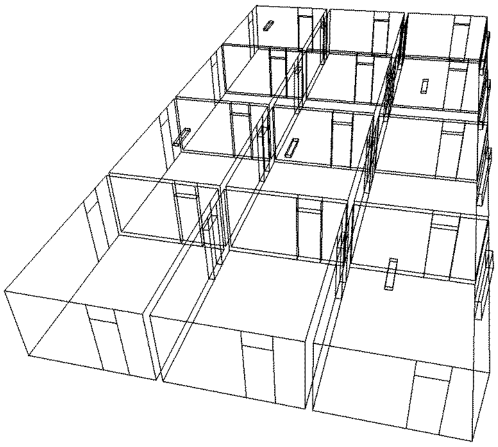

Figure 5: "floor" (289 polygons)

steps {14} and {7..13} is choosen for the criterion, if the graph is balanced or not.

#### 2.2.6 Rendering {15}

Rendering means sending the computation results to the front-end workstation. This makes sense after each new calculation of the scene. New results can be inspected at an early state and the user has an intermediate impression. If the overall computation converges, the rendering quality increases.

### 2.3 Application, Tests, Results

Tests were done with a fixed set of parameters. Two sample scenes were choosen. One of them is called *room* the other *floor*. *room* contains 27 surfaces, *floor* is built from combining 12 *rooms*. See figures 4 and 5.

For the test case the room gets 3300 elements and 23200 links. See table 1 for initial results. The refinement of the graph needs the most time. The push-pull phase can nearly be neglected. The optimization of the graph (simulated annealing) becomes expensive if the number of processors is increased.

| | time (seconds) | | | | |
|---|---|---|---|---|---|
| proc. | init graph | refine | push-pull | sim. ann. | sum |
| 1 | 6 | 354 | 14 | 0 | 374 |
| 2 | 4 | 404 | 13 | 0 | 421 |
| 4 | 2 | 362 | 11 | 12 | 377 |
| 8 | 2 | 209 | 9 | 17 | 238 |
| 16 | 2 | 130 | 5 | 14 | 151 |
| 32 | 3 | 85 | 4 | 22 | 114 |
| 64 | 6 | 74 | 4 | 60 | 144 |

Table 1: time for calculatimg "room" up to 3300 elements and 23200 links

| | time (seconds) | | | | |
|---|---|---|---|---|---|
| proc. | init graph | refine | push-pull | sim. ann. | sum |
| 1 | 1134 | 1278 | 61 | 0 | 2473 |
| 2 | 644 | 1567 | 36 | 0 | 2247 |
| 4 | 349 | 1145 | 22 | 20 | 1536 |
| 8 | 189 | 508 | 11 | 25 | 733 |
| 16 | 110 | 392 | 9 | 25 | 536 |
| 32 | 68 | 271 | 7 | 32 | 378 |
| 64 | 53 | 170 | 6 | 67 | 296 |

Table 2: time for calculatimg "floor" up to 6380 elements and 56400 links

We got better results for the *floor*-scene. The time needed for the different phases is shown in table 2. Especially the initialization of the graph works well for this bigger scene.

Figure 6 shows the speed-up of the graph refinement in *floor*. Figure 7 shows the speedup of the sum of all calculation phases. That the acceleration is not linear to the number of processors illustrates that communication is the most time critical action.

With the massive communication costs in mind, we sought the optimal distribution of a graph's nodes over the processors. We examined the graph of the "room"-scene with 3300 elements after the calculation was completed. A very cautious parametrized version of the simulated annealing algorithm was applied to the graph. It turned out that the resulting assignment of nodes of the graph to processors, which required much more calculation time than the "normal" parametrized algorithm, hardly exceeded the quality of the graph's distribution before this rearranging. So we think of our graph balance algorithm as an almost optimal algorithm.

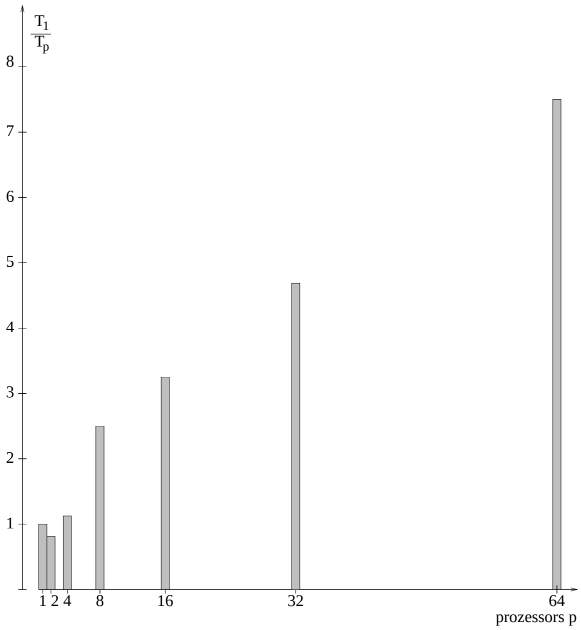


Figure 6: speed-up for the graph refinement in "floor"

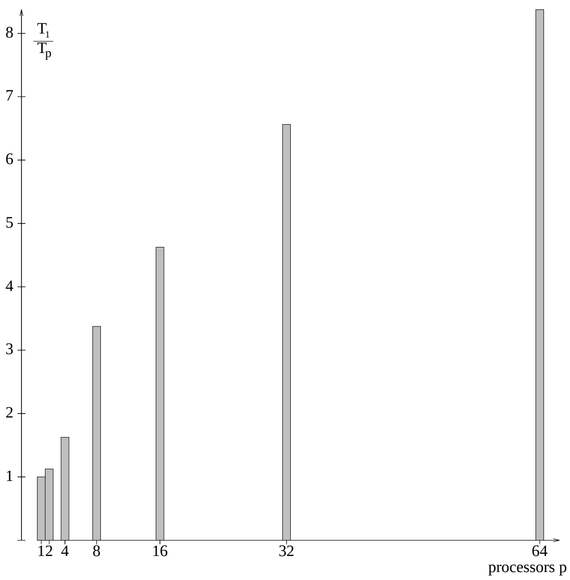


Figure 7: speed-up for "floor" (all phases)

# 3 Conclusion

Hierarchical Radiosity works on a huge tree-like data structure. This kind of algorithm seems to be one of the most difficult problems for execution on a MIMD computer, because of its non-local data distribution. A lot of communication is needed and slows the process down.

Making communication independent from the computational tasks by storing tasks that need to wait for data, while executing another job is one important criterion for getting an efficient implementation, as shown in this work. Another enhancement is proven by redistributing the task-graph on the processors after arriving at a certain job unequality on the single processing units. Simulated annealing does this to minimize the communication, while equalizing the amount of work on each processor.

The developed algorithm is scalable by simply adding processing units to the underlying hardware architecture. Although acceleration doesn't correspond in a linear way to the number of processors, a significant increase has been proven, which delivers a fast solution of the HR problem. Together with a high-end graphics workstation, a good practicability of HR is achieved.

Figures 8, 9, 10, 11 show a more complex scenario. The initial geometry contained about 4000 polygons. Calculation produced about 939 000 links and 30 000 elements and took about 104 minutes on 64 processors.

**Future work** — The massive communication costs in mind and the experience, that our balance algorithm can be thought of as nearly optimal under the restriction of equal workload, duplication of data seems to be reasonable. Any node of the graph on a particular processor $i$, that is heavy-connected to $i$'s and to a single other processor's $j$ nodes may be copied (not moved) to $j$ during the balance algorithm, since moving the node again leads to high communication costs. A deterministic heuristic optimization algorithm like [8] could be adapted for this purpose. Of course this strategy will reach memory limits quickly for very complex scenes, so care must be taken at the decision between duplication and displacement.

**Acknowledgement** — We would like to thank Heinrich Müller for his valuable comments, Georg Pietrek and Michael Pietsch for their work on the front-end renderer, and Andreas Bleicher who was strongly involved in modeling and rendering sample scenarios.

## References


[1] Aarts E H L, de Bont F M J, Habers E H A, van Laarhoven P J M: "Parallel Implementations of the Statistical Cooling Algorithm", *North Holland INTEGRATION, the VLSI journal*, 4, 1986.

[2] Cohen M F, Greenberg D P: "The Hemi-Cube: A Radiosity Solution for Complex Environment" *Computer Graphics* 19, 3, July 1985.

[3] Cohen F C, Wallace J R: "Radiosity and Realistic Image Synthesis" *Academic Press, Inc.*, Cambridge, Massachusetts, 1993.

[4] Gortler S, Schröder P, Cohen M F, Hanrahan P. "Wavelet Radiosity" *Computer Graphics (SIGGRAPH proceedings '93)*, July 1993.

[5] Hanrahan P, Salzman D, Aupperle L: "A Rapid Hierarchical Radiosity Algorithm", *Computer Graphics (SIGGRAPH proceedings '91)*, July 1991.

[6] Heckbert P S, Winget J M: "Finite Element Methods for Global Illumination", Tech. Report UCP/CSD 91/643, Computer Science Division (EECS), University of Berkeley, July 1991.

[7] Kajiya J T: "The Rendering Equation", *Computer Graphics (SIGGRAPH proceedings '86)*, August 1986.

[8] Kernighan B W, Lin S: "An efficient heuristic procedure for partitioning graphs", *The Bell System Technical Journal)*, February 1970.

[9] Kirkpatrick S, Gelatt Jr. C D, Vecchi M P: "Optimization by Simulated Annealing", *Science*, 220, May 1983.

[10] Metropolis W, Rosenbluth A, Rosenbluth M, Teller A, Teller E: "Equation of State Calculations by Fast Computing Machines", *J. Chem. Phys.*, 1953.

[11] Schröder P, Gortler S J, Cohen M F, Hanrahan P: "Wavelet Projections for Radiosity", *Proc. Fourth Eurographics Workshop on Rendering*, Eurographics, June 1993.

[12] Wallace J R, Elmquist K A, Haines E A: "A Ray Tracing Algorithm for Progressive Radiosity", *Computer Graphics (SIGGRAPH proceedings '89)*, July 1989.

[13] Zatz H R: "Galerkin Radiosity: A Higher-order Solution Method for Global Illumination", *Computer Graphics (SIGGRAPH proceedings '93)*, August 1993.

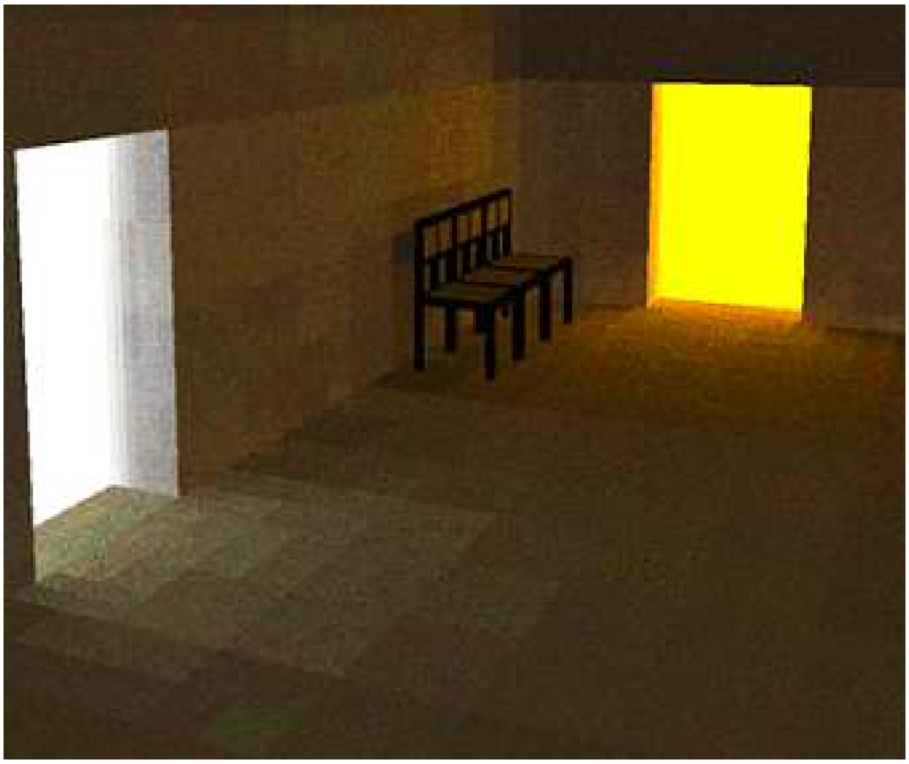

Figure 8:

Figure 9: The German Museum of Architecture in Frankfurt/Main

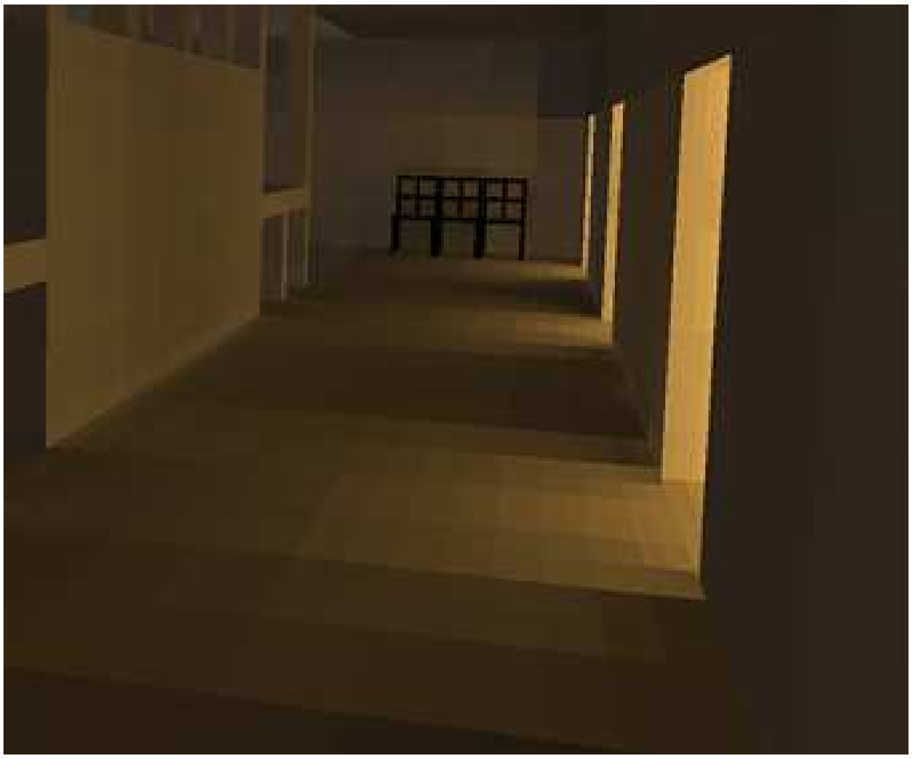

Figure 10:

Figure 11: Inside the German Museum of Architecture in Frankfurt/Main