Alcom-FT Technical Report Series
ALCOMFT-TR-02-94

The HiQoS Rendering System*

Tomas Plachetka, Olaf Schmidt, Frank Albracht

University of Paderborn, Department of Computer Science, Fürstenallee 11,
D-33095 Paderborn, Germany

Summary. The simulation of the global illumination in 3D scenes is a computationally expensive task. One of the goals of the project *HiQoS* (High Performance Multimedia Services with Quality of Service Guarantees) was to develop and test a prototype of an e-commerce system which simulates realistic lighting of large scenes on high performance parallel computers. The system, although tailored to the needs of this specific application, is very generic and exhibits metacomputing features: 1.the access to high performance computers is fully transparent to the user; 2.the modular architecture of the system allows to dynamically add or remove computing resources in geographically different computing centers. The prototype of the proposed system was evaluated in the industrial contexts of architectural visualization and film production. This paper summarizes scientific and technical problems which arose during the project as well as their solutions and engineering decisions.

1 Introduction

Photo-realistic visualization of 3D models is important in many industrial areas, for instance in architecture, entertainment industry and film production. The requirements to the level of realism as well as the complexity of the models grow very fast and so do the requirements to the performance of the systems used for the visualization. The computing power needed for the synthesis of photo-realistic images (*rendering* in this paper) in a reasonable time falls into the category of high performance computing.

Companies needing high-quality visualizations seldom have supercomputers on their own. Such machines are either too expensive or regarded as irrelevant by many IT managers. A rental and a temporary physical installation of additional computers in order to meet production deadlines are accompanied by technical problems and additional costs.

A prototype of an advanced e-commerce rendering system addressing the above problems has been developed during the project *HiQoS* (**Hi**gh Performance Multimedia Services with **Q**uality **o**f **S**ervice Guarantees) [1], [2]. This system, *HiQoS Rendering System*, allows a user to submit rendering jobs via

* Supported by the German Ministry of Education and Research (BMBF) and the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

a simple web interface. The further processing of jobs is fully automatic. The complexity and the distributed nature of the system are hidden from the user. The project HiQoS was financially supported by the German Ministry of Education and Research (BMBF). The HiQoS Rendering project partners were: Axcent Media AG, GPO mbH, IEZ AG, University of Paderborn and Upstart! GmbH.

We know about several projects related to a remote global lighting simulation. *Virtual Frog* [3] is one of the pioneering works which aims to support teaching of biological principles: "...our goal is to provide accessibility over the Web in order to reduce the complexity of installing, running, and managing the software." The interactivity played a major role in this project – the model (of a frog) resides on a server and the server computes a desired visualization of the model (a schematic view, a view of a scanned slice, a volume traced view, etc.) *Online Rendering of Mars* [4] is technically very similar: the user chooses a perspective and a lighting and within a few minutes gets back a rendered picture of Mars. Closer to the HiQoS project idea is a rendering server using the (sequential) *Radiance* program to compute ray traced pictures of a user's model [5]. There also are companies offering their computers for rendering purposes [6] – however, the data exchange, job specification and scheduling of the rendering jobs are handled by human operators. The HiQoS rendering project went further, offering an *automatic and parallel* rendering service on demand, whereby parallel computers in several computing centers can be combined into a single computing system.

The global illumination problem is defined in section 2. Parallelizations of two global illumination methods, ray tracing and radiosity, is discussed in the same section. Special attention is devoted to an efficient representation of the diffuse global illumination resulting from the radiosity method. Two approaches to the simplification of radiosity solutions are presented: mesh decimation and radiosity maps. The architecture of the HiQoS Rendering System is described in section 3. Section 4 describes an evaluation of the HiQoS Rendering System in two industrial scenarios, in architectural visualization and in film production. In section 5 we draw our conclusions and sketch our future research directions.

2 Simulation of Global Illumination

Photorealistic visualization of 3D models is one of the quests of computer graphics. Almost all light phenomena are well explained by the quantum electrodynamics. However, for practical purposes it is not desirable to simulate the propagation of light or to model large-scale 3D models on a subatomic level. Synthesis of photorealistic pictures is a chain of simplifications. Kajiya's rendering equation [7] provides a framework for the simulation of global lighting on the level of geometrical optics. The rendering equation is an integral equation describing an energy balance in all surface points of a

3D model. With exception of extremely simple cases this equation cannot be solved analytically. The ray tracing and radiosity methods make additional assumptions about the light-object interactions in order to simplify the rendering equation.

The assumption of ray tracing is that all indirect light reflections are perfectly specular. Ray tracing does not solve the rendering equation explicitly – it traces photons from the camera into the 3D scene (in a backward direction), measuring the contributions of light sources to the camera pixels along the traced paths. Ray tracing is inherently view dependent. Its product is the picture seen by a given camera. The illumination is not stored in the 3D model.

Radiosity [8] assumes that all light reflections are perfectly diffuse and that the 3D model consists of a finite number of small elements (*patches*). Under these assumptions the rendering equation can be formulated as a system of linear equations. The system is usually very large and the computation of all its coefficients is time intensive. The radiosity algorithms solve the system iteratively, storing the illumination in the 3D model. The radiosity method is view independent. A converged radiosity solution is an illuminated 3D model which can later be viewed from different perspectives without having to rerun the lighting simulation.

Data-parallel radiosity and ray tracing algorithms have been developed and implemented within the HiQoS project. The following sections briefly describe the parallelization ideas. (An asynchronous distributed memory model with message passing is assumed.)

2.1 Parallel Ray Tracing

Our ray tracing parallelization is based on a screen subdivision strategy which exploits the independence of computations on any two pixels of the camera. Processor farming is a natural way of the parallelization. However, there are two potential problems with a straightforward implementation: bad load balancing (computation times for two pixels are different) and no data-scalability (replication of the 3D model in each processor imposes a limit to the maximum model size).

The problem of unequal processor load can be solved under an assumption that the ratio of the computation times on any two equally large areas of the screen can be bounded by a constant [9]. The idea is to assign large screen areas to processors first, then smaller, ending up with single pixels (or other sufficiently small pieces). Almost linear speedups can be measured up to 64 processors.

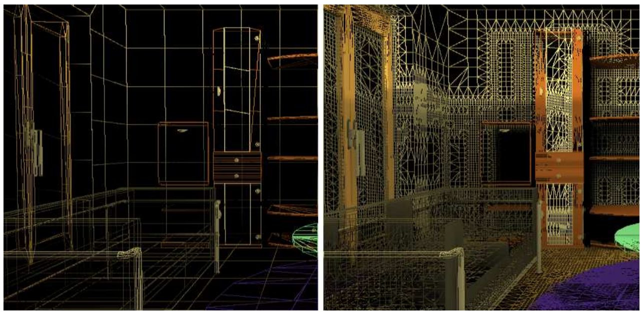
A distributed object database is used to avoid the necessity of storing the whole 3D model in every processor. Large models (requiring several Gigabytes of memory) can so be rendered on currently available parallel machines. The only limitation to the model size is the *total* memory of the processors used for the computation. Each processor holds a resident subset of all objects of the 3D model. The remaining objects (or a part of them) are stored in a cache memory. If a non-resident object is needed during the computation, the processor stops the computation, sends a request to the processor holding the object and makes space for the requested object in its cache by removing some other objects from the cache if needed. Upon having received the requested object, the processor resumes the computation. An LRU strategy is used for the cache management (always the *L*ast *R*ecently *U*sed object is removed from the cache). A similar implementation is described in [10].

2.2 Parallel Radiosity

A progressive refinement method [11] is used for the simulation of light propagation. Each patch of the model is a potential light source. The unshot energy and the total reflected energy are stored by each patch. The original (sequential) progressive refinement method iteratively selects a patch with the most unshot energy (a *shooter*) and shoots the whole unshot energy in the half-sphere surrounding the shooting patch. The total reflected and unshot energies of all other patches (*receivers*) are updated during the shooting according to their visibility from the shooter (the receiving patches can be refined in this step to store the gathered energy accurately enough). This process is iterated until the total unshot energy drops down under a threshold (or some additional termination criterium is fulfilled). The computation of a visibility between two patches (a so-called *form factor*) is a non-trivial problem. We are currently using a ray casting method for the form factor computation [12]. This method randomly generates samples on the shooter and the receiver and checks how many pairs of the shooter-receiver samples are mutually visible. The number of visible sample pairs is used as an estimation of the total mutual visibility between the two patches.

The parallel radiosity algorithm begins with a preprocessing step (a meshing step) in which the model is discretized into patches (in our implementation the mesh consists only of triangle and quadrangle patches). The discretized model is partitioned into 3D subscenes which are distributed onto processors. The shooting iterations run asynchronously in all processors. Aside of the locally stored patches, each processor maintains an incoming message queue which contains information about shooters selected by other processors. At the beginning of each iteration a processor selects a shooter with the most unshot energy from the patches in its message queue and the locally stored patches. Then it performs either an *external shooting* (if a shooter from the message queue has been selected) or an *internal shooting* (if a local shooter has been selected). A shooting influences only the locally stored patches. To spread the information about shooting iterations performed locally, the processor broadcasts the selected shooter in the case of an internal shooting [13], [14].

Additional improvements to the parallelization described above include dynamic load balancing (a processor's load can be measured by the num-



(a) (b)

Fig. 1. Refinement of a polygonal mesh: (a) original mesh; (b) refined mesh

ber of yet unshot external shooters waiting in the message queue) and two representations of the 3D model used to speed up form factor computations [15].

2.3 Simplification of Radiosity Solutions

The diffuse illumination computed by the parallel radiosity program is stored directly in the vertices of the polygon mesh (vertex radiosities). Interpolation is used to compute the outgoing radiosities in other surface points. The original mesh gets progressively refined during the radiosity computation (new vertices are created) in order to store the illumination accurately enough. This refinement leads to huge data volumes resulting from radiosity simulations (Fig. 1). Memory requirements of radiosity solutions cause problems by transferring the data to the user over the internet, by a subsequent postprocessing of the scene, by interactive walkthroughs, by rendering final pictures, etc. The following sections describe two methods of simplification of radiosity solutions. Both methods were implemented (both sequential and data-parallel versions) and the later was integrated in the HiQoS Rendering System.

Mesh Decimation. The idea of the mesh decimation method is an iterative deletion of edges from the model using the *vertex-unify* operation (joining of two adjacent vertices into one vertex). The problem of overwhelming memory complexity also arises by acquisition of 3D models using 3D scanners and the first existing mesh decimation methods have been developed in this application area [16], [17]. The presence of additional radiosity information stored in the vertices of an illuminated model influences only the choice of a metric used in the algorithm.

Mesh decimation algorithm
INPUT: a polygon mesh M, a desired compression ratio c
WHILE (compression ratio c is not reached)
(1) In M, select edge $e = [P_1, P_2]$ for removal
(2) Remove edge e by joining points P_1, $P2$ into point P
(3) Find optimal placement for point P
OUTPUT: a simplified polygon mesh M

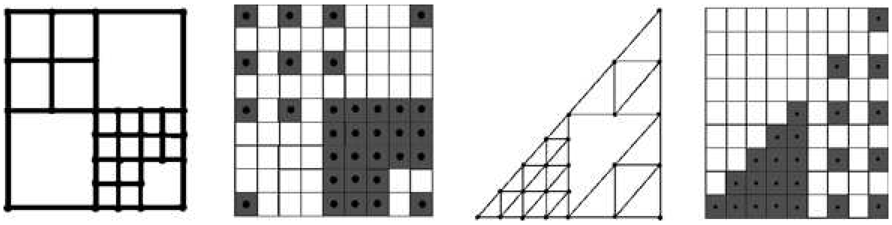
It is not obvious in which order the edges should be scheduled for the removal (line 1) and how to find the optimal placement for the vertices resulting from the *vertex-unify* operation (line 3). An objective *metric* is used to answer both questions in a single optimization step. The metric evaluates the current quality of the simplified mesh. The selection of the edge to be removed and the placement of the resulting vertex are a solution to an optimization problem which minimizes the quality reduction over all possible edge selections and all possible placements of the resulting vertex. The choice of the metric determines the complexity of the optimization problem (costs of evaluation of the metric alone must also be considered) and influences the final visual quality of the simplified mesh. We used a quadric metric [18], [19] which takes the vertex radiosities into account [20].

One problem of the mesh decimation method is that the visual quality of the simplified mesh is not guaranteed. Mesh decimation can be seen as a lossy compression of a model. While compression ratios of up to 90% can be achieved retaining an acceptable visual quality for some scenes, the visual error by the same compression ratios is high for other scenes. The compression ratio can be chosen interactively (using the operator's visual perception as a quality measure) but an interaction would make the method unsuitable for a use in an automated remote rendering system.

Radiosity Maps. The method of radiosity maps provides a non-lossy compression of illuminated meshes using a more efficient representation of the illumination. The vertex radiosities are stored in radiosity maps (texture maps) instead of in mesh vertices. One radiosity map is assigned to each object (an object is a set of patches forming a logical element, e.g. chair, table, staircase, etc.) Additionally, uv-mapping coordinates are stored in the vertices of the resulting mesh. The entire substructuring information is removed from the mesh. Typical compression ratios are 70%–80%. Texture mapping (see Fig. 3 (a)) is supported by the hardware of recent graphics cards which significantly increases framerates by interactive walkthroughs.

Computation of radiosity maps
INPUT: a mesh M with vertex radiosities
FOR (each object obj_k in mesh M)
  FOR (each patch p_i in object obj_k)
    (1) Find optimal resolution of radiosity map for patch p_i
    (2) Create radiosity map $pmap_i$ for patch p_i
    (3) Fill radiosity map $pmap_i$ for patch p_i (Fig. 2)
  (4) Pack patch maps $pmap_i$ into object map $omap_k$ (Fig. 3 (b))
  (5) Remove substructuring from all patches p_i in object obj_k
  (6) Assign uv-mapping coordinates to all patches p_i in object obj_k
OUTPUT: a simplified mesh M with radiosity maps assigned to objects
  (substructuring information removed)

The optimal resolution of a patch map (line 1) is a minimal resolution allowing a non-lossy representation of the original illumination information. This resolution depends on the level and on the form of the substructuring of a given patch (examples are shown in Fig. 2). An exhaustive search in the substructuring tree is used for computing the optimal resolution.



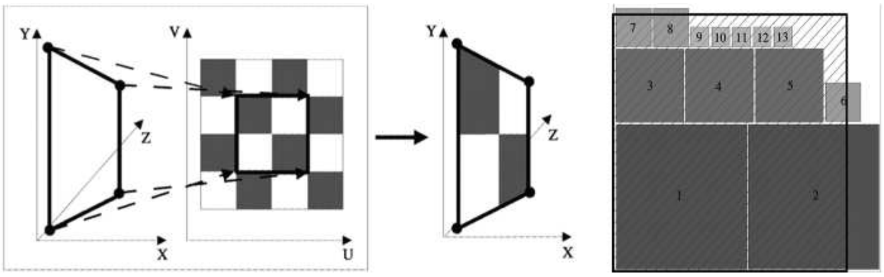
(a) (b)

Fig. 2. Optimal resolutions of radiosity maps: (a) a quadrangle patch and the corresponding map; (b) a triangle patch and the corresponding map.
Original vertex radiosities are stored in the marked texels. The colours of empty texels are interpolated from the marked ones

An optimal packing of patch maps into an object map (line 4) is an NP-hard problem. A heuristic is used in this step. The heuristic tries to keep the object map as small as possible and to use the rectangular area of the object map efficiently (Fig. 3 (b)).

3 Architecture of the HiQoS Rendering System

The HiQoS Rendering System is a distributed system consisting of four subsystems: *Client*, *Service Broker*, *Scheduling Subsystem* and *Rendering Server*. All of these subsystems may be replicated. An exception is the *Service Broker*, the system's central entry point. The subsystems and their components



(a) (b)

Fig. 3. (a) Mapping of a radiosity map onto a 3D object (*uv*-texture mapping). The *uv*-mapping coordinates are stored in the object's vertices; (b) Packing of patch radiosity maps into an object radiosity map. The dashed area corresponds to the total area of the patch maps (a lower bound for the optimal object map size)

communicate via TCP/IP sockets. No shared file system is needed. A possible configuration of the system is shown in Fig. 4. The main objectives of this architecture are:

- minimal hardware and software requirements on the user's side (only an Internet connection and a web browser are needed)
- a transparent access to the high performance computing systems (the user does not know where the computation takes place or what computing systems are used for computing his job)
- a good resource utilization (especially by rendering walkthroughs consisting of many frames).

3.1 Client

The *Client* subsystem is responsible for a user's authorization by the *Service Broker*, submission of a rendering job and providing input data needed by the job. The input data generated automatically by the user's modelling software describe a 3D scene: its surface geometry, surface materials, light sources and cameras. Extended *VRML 2* and *3DS* formats are currently supported. It is practical to separate the camera information from the rest of the scene description (a proprietary camera format is used which allows a definition of single cameras as well as camera paths). The rest of the data are controlling parameters specific to the selected illumination method. These are provided by a human operator who fills out an HTML form when submitting a rendering job.

The 3D scene data can be large. They bypass the service broker and flow from the *Client* directly to the *Global Scheduler*. Two transmission scenarios were considered:

- *A passive client scenario*, in which the user places the exported scene data in a public area in the Internet (e.g. the user's web area) and tells the rendering system where they are (URLs). This is the currently implemented scenario. Main disadvantages of this scenario are an additional load in the system related to the downloading the user's data and a violation of the privacy of the data. An advantage is a simple implementation.
- *An active client scenario*, in which the user (e.g. a software tool integrated in the user's 3D modeller) uploads the data to the rendering system. Advantages are a possibility of integration of accessing the remote rendering system directly from the user's modeller, a possibility of maintaining an object cache on the rendering system's side for reducing the transmission costs, ensuring the privacy of the transmitted scenes, etc. A disadvantage is that additional software components and communication protocols between them must be implemented.

3.2 Service Broker

The *Service Broker* is the system's entry point. This software component communicates with users (and administrators of the system), accepts new jobs, generates a unique id for each accepted job and delivers computed results to users. The *Service Broker* is implemented in an Apache web server (PHP scripts are used for implementing the communication with the *Global Scheduler* and for accessing databases kept on the server).

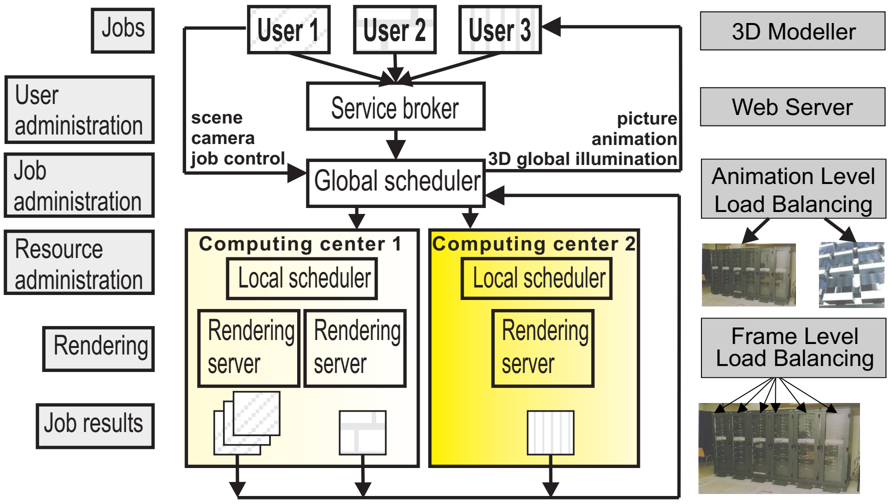


Fig. 4. An example configuration of the HiQoS Rendering System

3.3 Scheduling Subsystem

The *Scheduling subsystem* consists of a *Global Scheduler* and several *Local Schedulers*). The former takes care about downloading the job data, converts the data, distributes jobs (or their parts) to the network of *Local Schedulers*, collects the results (or partial results) from *Local Schedulers* and passes the results and a status information to the *Service Broker*. There is usually only one *Global Scheduler* running in the entire system.

Different parallel computing systems in different computing centers can be used in the system. There is usually one instance of the *Local Scheduler* running in one computing center. The main task of the *Local Scheduler* is to hide the differences between the parallel systems, providing a unique interface for allocation and deallocation of processors, for starting a parallel application on allocated processors, etc.

Such two-level scheduling scheme is modular and also helps to efficiently utilize the computing resources by rendering of ray traced walkthrough animations. In this case the *Global Scheduler* acts as a farmer distributing single frames to several *Local Schedulers*. After having computed a frame, the *Local Scheduler* sends the frame (a picture) to the *Global Scheduler* and gets in turn a new frame to compute. Between the computations of two subsequent frames the 3D scene persists in memories of running parallel processors on the *Local Scheduler*'s side. Thus only a short description of the new camera must be retransmitted between the schedulers with each single frame.

3.4 Rendering Server

Data-parallel ray tracing and data-parallel radiosity algorithms are currently supported by the HiQoS Rendering System. These programs are precompiled for the target parallel systems in computing centers. An actual implementation of parallel global illumination algorithms is hidden in the *Rendering Server*. The *Rendering server* provides an interface to a *Local Scheduler* allowing starting, continuation and termination of a chosen parallel program on an allocated partition of a parallel system. The interface is independent of the illumination method, even though the methods considerably differ from each other (also in input and output parameters). A radiosity job is handled by a *Local Scheduler* in the same way as a ray tracing job with one camera.

4 Evaluation of the HiQoS Rendering System

The remote rendering system was evaluated in two industrial scenarios: architectural visualization and film production. Although the final goal of both scenarios is a synthesis of photorealistic pictures, the way of achieving this goal differs. The user of the first scenario was the company *GPO mbH* which creates complex CAD models using the software *Speedikon* (by *IEZ AG*) and

(a) (b)

Fig. 5. (a) Model of the Dom in Wetzlar. Global diffuse illumination was computed by radiosity, textures and camera effects were added in an independent rendering step; (b) A ray traced model of a furnished house

Arcon: Architektur Visualisierung (by *mb-Software AG*). The requirement on the rendering system was a synthesis of high quality visualization pictures and camera animations of the models. The results were used for supporting the building contractors during the planning phase and for a public presentation of the models. The parallel ray tracing offered by the HiQoS Rendering System allowed to significantly reduce the rendering times in comparison to a sequential computation on a PC. The measured total overhead of the system (the effective rendering time against the time spent in downloading the models and in scheduling) was below 10% already by small rendering jobs consisting of about 10 frames (and smaller by more complex jobs).

The user of the film production scenario was the company *Upstart! GmbH* which creates special visual effects for movies (e.g. “Operation Noah”) and advertisements. *3D Studio Max* (by *Discreet*) is used for the 3D modeling. A frequent problem is a realistic illumination of building interiors which do not really exist. A sequence of pictures is not desirable as a product of the global illumination simulation because this would drastically reduce the flexibility by the final composition. Rather than pictures, an explicit 3D representation of the illuminated model is required as an intermediate result of the simulation. The (sequential) radiosity of *Lightscape* (by *Discreet*) is used in this step of the original production chain. The sequential radiosity computation of a complex model can take many hours, sometimes days. Another, even more serious problem are the run-time memory requirements of the radiosity method which sometimes exceed the possibilities of a PC. The data-parallel radiosity integrated in the HiQoS Rendering System leads to shorter computation times and overcomes the memory problems by using more processors with more total memory. The radiosity maps described in section 2.3 are used for the compression of the resulting data.

5 Conclusions

We described a parallel rendering system which allows computation of the global illumination of complex 3D models sent by users via the Internet. Currently data-parallel ray tracing and radiosity illumination methods are supported by the system. The system was integrated as a prototype of an e-commerce service and successfully evaluated in two industrial scenarios. Possible extensions and research areas include implementations of further global illumination methods, a seamless integration of the remote rendering service with existing 3D modellers, a support for dynamical scenes and a better resource management.

References

1. P. Altenbernd, F. Cortes, M. Holch, J. Jensch, O. Michel, C. Moar, T. Prill, R. Lüling, K. Morisse, I. Neumann, T. Plachetka, M. Reith, O. Schmidt, A. Schmitt, A. Wabro (1999) BMBF-Projekt HiQoS. Statustagung des BMBF, HPSC '99, Höchstleistungsrechnen in der Bundesrepublik Deutschland, Bonn, R. Krahl (ed.), 29–32
2. Projekt HiQoS: High Performance Multimediadienste mit Quality of Service Garantien. http://www.c-lab.de/hiqos
3. D. W. Robertson, W. E. Johnston., W. Nip (1994) Virtual Frog Dissection: Interactive 3D Graphics via the Web. Proc. of The Second International WWW Conf. Mosaic and the Web, Chicago, Illinois
4. Mars Online Rendering. http://astronomy.swin.edu.au/mars
5. Artifice, Inc., RenderCity! – Free Online Ray Tracing with Radiance. http://www.artifice.com/rendering/render.html
6. MTP Grafx, Render Farm. http://www.mtpgrafx.com/rfarm.htm
7. J. Kajiya (1986) The Rendering Equation. Computer Graphics, 20, **4**, 143–150
8. F. X. Sillion (1994) Radiosity and Global Illumination. Morgan Kaufmann Publishers, Inc.
9. T. Plachetka (1998) POV||Ray: Persistence of Vision Parallel Raytracer. Proc. of Spring Conf. on Computer Graphics, Budmerice, Slovakia, 123–129
10. S. Green (1991) Parallel Processing for Computer Graphics. Pitman MIT Press
11. M. F. Cohen, S. E. Chen, J. R. Wallace, D. P. Greenberg (1988) A Progressive Refinement Approach to Fast Radiosity Image Generation. Computer Graphics, 22, 75–84
12. J. R. Wallace, K. A. Elmquist, E. A. Haines (1989) A Ray Tracing Algorithm for Progressive Radiosity. Computer Graphics, 23, **3**, 315–324
13. O. Schmidt, J. Rathert, L. Reeker, T. Plachetka (1998) Parallel Simulation of Global Illumination. Proc. of the Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA '98), Las Vegas, Nevada, CSREA Press, H. R. Arabnia (ed.), **3**, 1289-1296
14. L. Reeker, O. Schmidt (1999) New Dynamic Load Balancing Strategy for Efficient Data-Parallel Radiosity Calculations. Proc. of the Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA '99), Las Vegas, Nevada, CSREA Press, H. R. Arabnia (ed.), **1**, 532–538

15. O. Schmidt (2000) Parallele Simulation der globalen Beleuchtung in komplexen Architekturmodellen. PhD Dissertation, Fachbereich Mathematik/Informatik, Universität Paderborn
16. J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright (1996) Simplification Envelopes. Computer Graphics (Proc. of SIGGRAPH '96), 119–128
17. H. Hoppe (1996) Progressive Meshes. Computer Graphics (Proc. of SIGGRAPH '96). 99–108
18. M. Garland, P. Heckbert (1998) Simplifying Surfaces with Color and Texture using Quadric Error Metrics. Proc. of IEEE Visualization '98, 263–269
19. H. Hoppe (1999) New quadric metric for simplifying meshes with appearance attributes. Proc. of IEEE Visualization '99, 59–66
20. M. Rasch, O. Schmidt (2000) Parallel Mesh Simplification. Proc. of the Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA '00), Las Vegas, Nevada, CSREA Press, H. R. Arabnia (ed.), **3**, 1361–1367