Experimental System for Visualization of the Light Load

Martin Čadík
Czech Technical University
Faculty of Electrical Engineering
Karlovo náměstí 13
121 35 Prague, Czech Republic
cadikm@cs.felk.cvut.cz

Pavel Slavík
Czech Technical University
Faculty of Electrical Engineering
Karlovo náměstí 13
121 35 Prague, Czech Republic
slavik@cslab.felk.cvut.cz

Jan Přikryl
Czech Technical University
Faculty of Transport Sciences
Na Florenci 25
11000 Prague, Czech Republic
prikryl@fd.cvut.cz

ABSTRACT

This paper presents our work on an experimental system for visualization of the light load. The light load is thought as the total amount of light radiation received by all areas of the solved environment. The emphasis of the presented system is on outdoor architectural scenes, but indoor scenes are handled as well. The aim of the designed system is to visualize the light load either in one moment or integrated during a longer time period. We have selected the hierarchical Monte Carlo radiosity method to solve the specified problem. This method was enriched by parallel light sources and specular reflections to face the specific aspects of the problem. New "lighting" iteration was added to computation phase to consider natural light sources such as Sun and sky. The system was implemented in Java programming language using the Java3D API. Thanks to this implementation environment our system is flexible, easy to modify and extend and it is suitable for experimental and educational purposes.

Keywords

Computer graphics, global illumination, radiosity, ray tracing, Monte Carlo techniques.

1. INTRODUCTION

The aim of our work was to develop an experimental system for physically accurate global light propagation simulation and visualization of the light load mainly of outdoor architectural scenes. Light load is thought as the amount of light radiation received by the areas of the solved environment. The input of the system is a 3D model of a real scene and definitions of light sources (their position, intensity etc.). The emphasis is given on obtaining view-independent solution that visualizes the light load, in the form of a 3D scene suitable for presentation purposes. The essential parameters of the system are flexibility, simplicity, easy modification and extension and possibility to use it also for educational purposes.

Journal of WSCG, Vol.11, No.1., ISSN 1213-6972
WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.

Light conditions are necessary to take into account not only for light load of the facades of the buildings but also for safety reasons and creating a pleasant environment for habitants or employees residing in these buildings. As an example of an application (see Figures 1 and 9) we can mention an existing street where an old building made of wood and bricks is replaced with a new one made mainly of very reflective materials like glass, chrome etc. This new building affects the others buildings in the street by light reflections and thus light conditions in the street alone. We would like to know how many areas in the street are affected by the light reflected by the new building during the whole day or in some particular moment.

2. PREVIOUS WORK

Existing software for physically accurate solution of global illumination problems is typically hard to work with. It requires a lot of knowledge of the problem, it is usually very complex, expensive and it is intended mainly for indoor scenes. Majority of this software make only 2D pictures or animations and does not produce the output in the 3D scene format where we can investigate the situation more in detail, from different viewpoints.

We survey the most popular software packages for solution of global illumination problems:

Radiance Synthetic Imaging System [Lar98a] is a ray tracing software package that enables accurate and physically valid lighting and daylight simulations. Radiance is based on the stochastic backward ray-tracing algorithm and it has rather complicated command line based control. The *Desktop Radiance* is a graphical user interface for Radiance system which runs under the Windows operating system from within AutoCAD system using pull-down menus.

Integra Inspirer [Int02a] is a complex commercial package for architectural lighting design. It uses bi-directional ray tracing algorithm to analyse light propagation in a scene and to compute illuminance distribution. Ray tracing in Inspirer is accelerated by uniform space subdivision.

AutoDesk Lightscape [Aut02a] is another commercial package using both radiosity and ray tracing computation. The radiosity solution is obtained by the progressive refinement approach. After the radiosity calculation has been done the ray tracing can be performed for a given view to add specular reflections and transparency effects.

RenderPark [Bek00a] is a test-bed system for physically based photo-realistic image synthesis. It is a free software package providing a solid implementation of a wide variety of state-of-the-art ray-tracing and radiosity algorithms (stochastic ray tracing, bi-directional path tracing, Galerkin radiosity, photon maps, stochastic Jacobi radiosity, and various random walk radiosity algorithms). Using a common source code base allows to compare global illumination algorithms on a fair basis, and to evaluate their benefits and shortcomings.

For experimental and educational purposes we need a relatively simple, transparent and flexible system that is easy to maintain, modify and extend. The systems mentioned above are either too expensive or very complex and hard to work with or both. Even the open-source candidates (Radiance, RenderPark) are not suitable to experiment with, expand or modify for relatively inexperienced people. They are also primarily dedicated to produce realistic pictures of illuminated indoor scenes and majority of them is not able to produce 3D output or to visualize light load during some longer time period. This led us to development of our own system that is based on Java3D paradigm and so it lets the user to concentrate solely on solving the light distribution in a scene.

3. METHODS FOR SOLVING THE LIGHT DISTRIBUTION IN A SCENE

Physically accurate global light propagation simulation requires solving the *rendering equation* [Kaj86a]. A solution may be obtained by the means of two principal approaches. The first one could be classified as the group of ray tracing based methods, the second one is the group of finite element methods solving the rendering equation on a finite element mesh for purely diffuse scenes.

Ray tracing based methods

Basic ray tracing [Whi80a] is a point-sampling technique. It cares about rays of the light that enter the observer's eye which are traced in backward direction. Ray tracing based methods excel when computing scenes with point light sources, direct illumination, specular reflections and refraction through transparent materials. They typically give the result in the view-dependent 2D form and are relatively memory efficient. Nowadays ray tracing variants based on Monte Carlo sampling seem to be more perspective and more popular than the radiosity methods. However, ray tracing based methods have difficulties computing scenes with predominating area light sources or scenes with strong diffuse reflection, which is crucial for us.

Finite element methods

Typical representative of this group is the radiosity method. Classical radiosity methods produce a view-independent 3D solution — rather then computing pixel values on the screen, these methods calculate intensities for patches in a 3D environment. The original surfaces are divided into a mesh of smaller elements. The final radiosity values are stored for each element of the mesh. The main drawback of classical radiosity methods is their diffuse orientation – all objects in the scene are treated as Lambertian reflectors [Coh93a]. The memory consumption of these methods is higher owing to refined 3D mesh.

Initial approach to solving a system of radiosity equations stored a *full matrix* describing the system of linear equations [Coh93a]. It was very slow and complex pre-processing had to be performed. The next approaches like *progressive refinement* [Coh88a] are faster and they eliminate difficult and time-consuming pre-processing. The *hierarchical radiosity* [Han91a] is besides the *wavelet radiosity* [Gor93a] probably the most popular deterministic radiosity approach. Both methods create a hierarchy of elements and the fine initial tessellation of the input scene is not necessary.

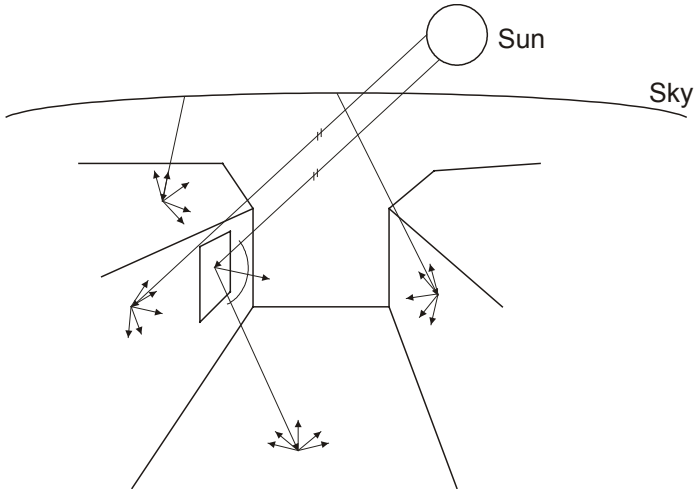

Figure 1. Light conditions in an outdoor architectural scene. The Sun and the sky are essential light sources. Diffuse reflections are prevalent, but specular reflections are not negligible.

Recently developed methods

Contemporary approaches (based either on stochastic radiosity [Gra01a] or on photon mapping [Jen02a]) to global light simulation join positive qualities of the methods of both mentioned groups, because random walk methods and finite element methods are in many ways complementary approaches. Stochastic principles and computation with iterations are also very often involved in the methods of both groups.

4. SOLUTION

The basic observation is that the light load is view independent. This implies that we can get the solution in the form of a static 3D scene.

Selection of the method

Our goal is to obtain a visualization in the form of a 3D scene and therefore we select solution from finite element methods group. Architectural scenes contain indeed many flat surfaces that can be approximated by diffuse patches. Such scenes could be solved in an exact way with the radiosity method. However, it is necessary to represent also specular reflections and non-diffuse light sources which are not possible to neglect. The selected method should be scalable enough and should be extendable for necessary phenomena for our purposes. We believe that the *hierarchical radiosity* based on *stochastic Jacobi iterative method* [Bek99a] meets best these requirements.

Hierarchical Monte Carlo radiosity

Hierarchical Monte Carlo radiosity is the conjunction of two popular approaches in the radiosity calculation – *hierarchical radiosity* and *stochastic computation*. The goal is to combine advantageous properties of the both methods. Hierarchical radiosity [Han91a] contributes with automatic adaptive subdivision of surfaces and with strong reduction of form factors that we are forced to evaluate and store in the memory. On the other hand - unfavourable is the sensitivity of this method to the error in the computation of the form factors that can lead to incorrect or even to forgotten energetic transfers. *Monte Carlo radiosity methods* excel with low memory requirements, because it is not necessary to explicitly evaluate and store the form factors. Compared to the hierarchical radiosity the stochastic methods can achieve higher stability of the computation and lower sensitivity to the error of missing or badly computed occlusion. The usual drawback of stochastic methods is the high frequency noise. As the stochastic part of the method we selected the *stochastic Jacobi iterative method* [Bek99a]. According to the author, this method is a good candidate to be used in practice, because the computational cost is related to the number of samples that need to be shot, rather than to the number of patches in the scene.

The hierarchical Monte Carlo radiosity method offers some advantageous features for computation of outdoor scenes: it can be relatively easily enriched of parallel light sources and specular reflections and the quality of the output scene could be easily controlled. However, properties of the whole system are affected by the choice of the computation method. Thus the input scene should be tessellated (divided into patches of finite area) – which is partially done by the hierarchical subdivision. All patches of the scene are Lambertian (diffuse) reflectors and media are not participated (outer environment is vacuum).

Extensions to the hierarchical Monte Carlo method

Specular Reflections

Smooth surfaces reflect the incoming light in a single direction – at an angle equal to the incident angle at which they arrive to the surface. A typical example of perfectly specular surface is a mirror (see Fig. 2). Specular surfaces like glass, chrome, metals etc. are very usual in architectural scenes and therefore it is necessary to extend diffuse-only method to solve specular reflections correctly.

Every object in our scene has its own specular reflection coefficient ρ_S. For each light ray crossing an object with $\rho_S \neq 0$ the recursive *ray tracing* is performed. The ray tracing finishes when the ray reaches a surface $\rho_S=0$ or when the depth of recursion is equal to the predefined fixed value. This pre-processing is performed only once in our system — during a "lighting" iteration (described below).

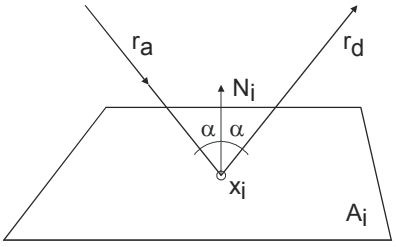

Figure 2. Specular reflection

Parallel Light Sources

Our extension that includes parallel light sources is necessary to allow the user to model non-diffuse light sources and also very distant light sources. This extension is straightforward and unpretentious: the computation for the patch A_P of a parallel light source is similar to the computation of a patch A_D of a diffuse light source, the difference is in sampling the ray leaving the patch. In the case of patch A_P the origin of the ray is uniformly sampled as in the case A_D, but its direction is equal to the normal of the patch A_P.

Lighting Iteration

There are two aspects of the sky and Sun - first they can form a background for outdoor scenes. In interior environments we can also look out through windows and other outlets. This aspect of exterior is necessary to make realistic-looking pictures, but it does not affect the light load. Therefore it is not considered in our system. The second case is the use of the sky and Sun to illuminate the scene. Sun and sky characteristics differ a lot from other synthetic light sources so they are treated separately in our system. For computation of effect of Sun and sky we inserted new lighting, initial iteration into the iterative process.

During the lighting iteration all parallel light sources in the input scene are considered. Within the computation there are different thresholds and other computational parameters involved and the oracle function (see [Coh93a] for explanation) also differs from the one used in other iterations. Thanks to this it is possible to set special behaviour of the algorithm to the Sun or sky patches. For example the distance of affected patches is not considered in the lighting oracle function and the division parameter $F_{L\varepsilon}$ used during the lighting iteration has the different value than parameter F_ε. Hierarchical subdivision of patches is performed only for objects of the scene and not for light sources during the lighting iteration. It is not necessary to subdivide patches of global light sources.

After all the energy of the global light sources has been shot out the gathered energy on the patches is treated as the self-emission Φ of the patch. During the next iteration the Sun or the sky are no-longer considered. The effect of these light sources has been incorporated into the self-emission of patches in the scene and diffuse reflections of the global light sources are neglected. A similar approach in context of progressive radiosity and pure hierarchical refinement is described by Müller and colleagues [Mul95a] and Daubert et al. [Dau97a], respectively.

Approximation of the Sun's path

Integral part of our solution is investigation of the influence of the sun *trajectory* on the light load in the given scene configuration. The Sun's relative position to the place on Earth changes during the day. The form of sun trajectory is also dependent on the season of the year and latitude and longitude on Earth.

In the case of Sun's path approximation the result is obtained in the form of J/m^2. Thanks to this we can visualize the light load during some time interval for example during the whole day.

Visualization

One of the most common and important tasks for a lighting designer is to determine the luminance levels of the space that is being designed or measured. Thus the primary goal is the visualization of the light load. Output scene rendered in colours based on wavelength of light reflecting off the surfaces does not permit to obtain a good idea of the light load. Even if this form of output is possible in our system, much better insight can be obtained by visualization in so called false colours. False colour means that instead of displaying surfaces of the scene in real colours we display them in colour that corresponds to the level of

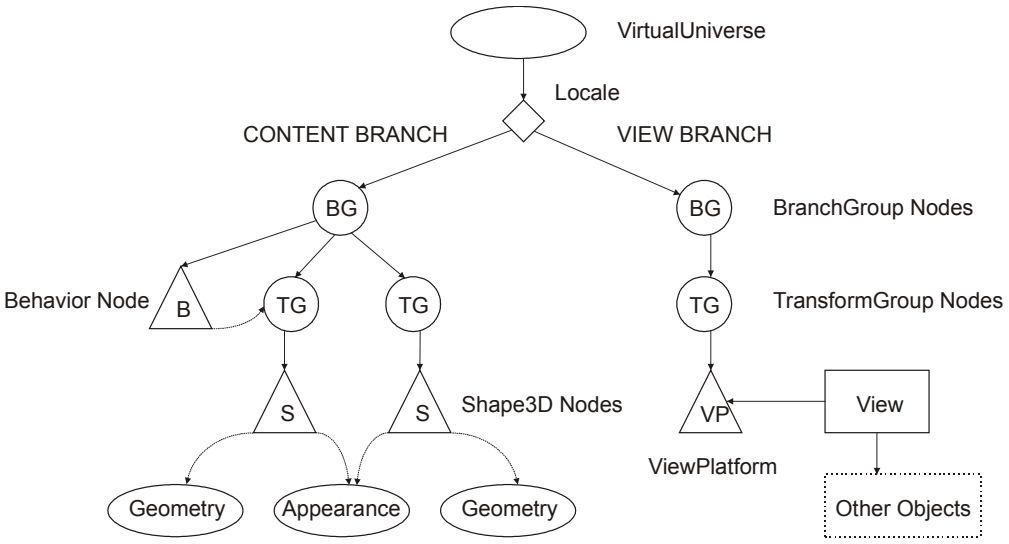

Figure 3. Scene Graph structure

irradiance (W/m^2) or light load (J/m^2) in the case of longer time period visualization. Visualization is performed in the 3D space so we do not obtain only 2D picture, but the lighting in the 3D scene. This scene can be saved in the VRML [ISO97a] format and can be used to investigate the results, perform walk-throughs where single parts of the scene can be investigated.

5. IMPLEMENTATION

Our experimental system was implemented in the Java programming language using the standardised Java3D application programming interface (API). This makes the system portable to various hardware platforms and easy to modify.

Java3D

Java3D is the interface for writing applications or applets that display or interact with three-dimensional graphics. Besides 3D graphics constructs it offers mathematical classes for vector calculations, constructs for manipulation with sounds etc.

Scene Graph

All Java3D programs are based on the Scene Graph data structure. Scene Graph is the tree data structure that describes entire scene (see Fig. 3). It consists of two main branches – a viewing branch and a content branch. Both branches are composed by instances of various Java3D objects. The content branch cares about geometry, appearance, behaviour and so on, while the viewing branch is intended to set up and change the viewing parameters. As the content branch can get very complex it can be automatically compiled into an optimised internal format. The Scene Graph is traversed by the Java3D renderer that chooses the traversal order – it is not restricted to left-to-right or top-to-bottom – to display it.

Data structure of our System

While loading an input 3D scene the scene graph is constructed. It consist predominantly of Shape3D objects representing shapes formed by triangle patches. As we need to store the hierarchy of elements that grows in consequence of subdivision during computation, we place it as a user data directly to Shape3D objects, see Fig. 5. Collections shapes and leafs enable fast direct access to Shape3D objects and leaf elements respectively.

Acceleration of the Java3D Ray Casting

As with any other computation method that is based on ray casting, the speed of a ray-primitive intersection test has an important influence on the overall speed of an implementation. Unfortunately, Java3D

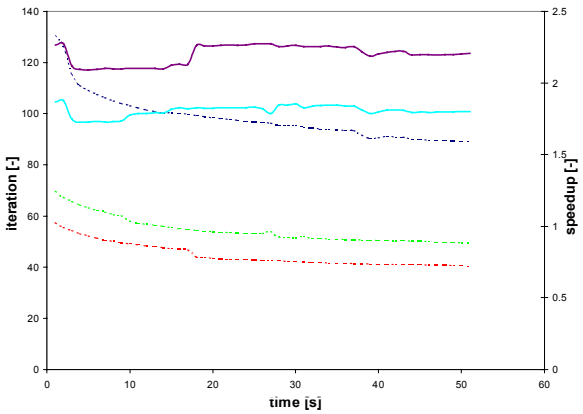

Figure 4. Speedup of the Java3D ray casting. Dashed curves show time demands (from top) of the original**,** org.j3d and our method. Solid curves show speedup of our method and org.j3d method.

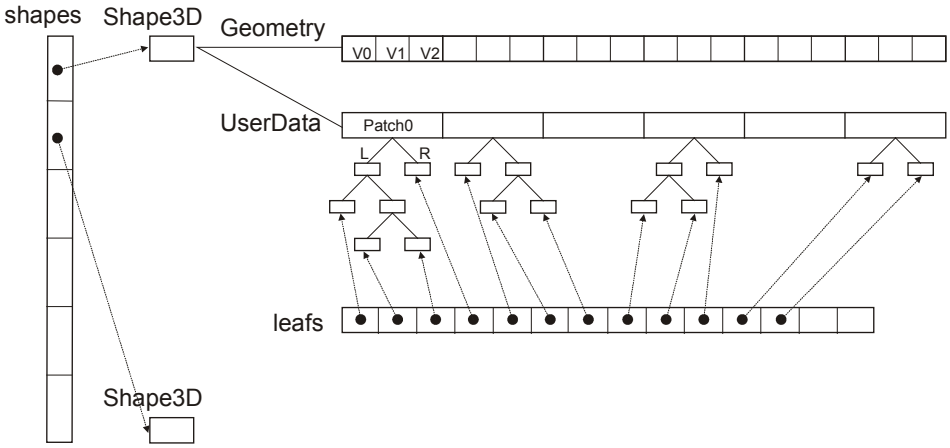

Figure 5. Data structure of the system

API was designed with hardware-assisted rendering in mind and native methods for ray-object intersections are quite inefficient. Original ray casting performed via object picking is actually very slow. In the absence of a decent object space subdivision scheme (regular grid, kd-tree [Hav00a]), the main potential of improving the performance of ray-primitive intersection tests lays in writing custom intersection routines for every primitive. As our system is limited to using triangle patches, this boils down to writing an optimised ray-triangle intersection routine, such as for example the one proposed by Badouel [Gla90a]. As we are using Java where frequent memory allocation is very expensive, we also have to keep the time-critical sections free of object instantiation and allocation.

In the search for the fastest possible intersection mechanism, we have first replaced the original ray-picking tests by open-source routines for ray-triangle intersection available from org.j3d group [J3d02a], which brought us a decent speedup. However, ray intersection routines in org.j3d package need to recompute many object parameters from scratch, which still brings some performance penalty.

The final improvement was computing the ray intersection directly on our data structures used for top-level patches using a variant of Badouel's algorithm [Gla90a], where all important constants used within the intersection test have been precomputed. In comparison with original Java3D implementation, we have achieved an average speedup of 2.2. See Fig. 4 for relative comparison of the three tested intersection methods.

6. RESULTS

We have verified proper operation of our experimental system on a set of test scenes including well-known Global illumination test scenes [Smi00a]. Figure 7 shows results of two of these scenes.

In order to compare whether the convergence of the implemented method corresponds to the expected convergence rate of stochastic Jacobi relaxation we carried out several measurements on various scenes. These scenes included simple indoor scenes similar to the tests scenes mentioned above and also outdoor scenes such as the one on the Figure 9.

The resulting graph on Figure 6 shows the magnitude of the error (in Watts) as a function of the iteration number. As one may see the convergence and especially variance strongly depends on the number of the samples shot to the scene. We can see that — according to our expectation — the error is decreasing with $O(1/N^k)$.

Figure 8 demonstrates the capability of the system to account for specular reflections in outdoor scenes. The right image on Figure 8 shows how our system handles the computation of light load for a sun-lit scene where the sun moves over the sky for approximately one hour. One can see that the illumination during this rather short time period did not completely wash out the borders between reflections on the street.

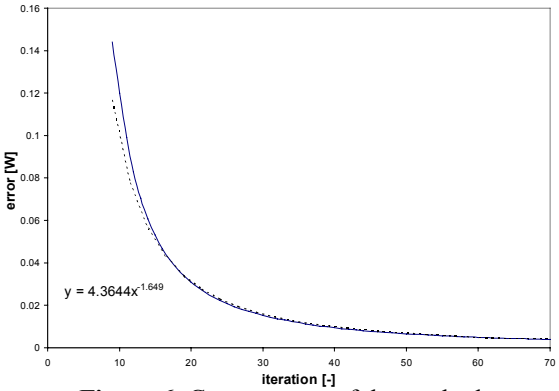

Figure 6. Convergence of the method. Approximation curve (dashed line) shows convergence rate as a least-squares fit to the error data.

7. CONCLUSIONS AND FUTURE WORK

In this paper we presented an experimental system for visualization of the light load. To face the specific aspects of this task we made several extensions to the hierarchical stochastic radiosity method. The hierarchical Monte Carlo radiosity method was enriched by treating parallel light sources and by specular reflections. New "lighting" iteration was added to computation phase to consider natural light sources such as the Sun and the sky. The system was implemented in Java programming language using the Java3D API.

The selection of the computation method gives some specific characteristics to our application: the result obtained is a view-independent representation of the whole 3D scene and thus it is not necessary to compute many 2D views to the scene. The result is saved in the VRML format and is suitable to use it for presentation and inspection purposes (interactive walkthrough) and is also acceptable for the web environment. The finite element representation of the scene reduces usual high-frequency noise of stochastic methods in environments with prevalent diffuse reflection.

Possible extensions of the system are: output of the animation of light proportions during the day, further speeding-up of the computation process by some spatial subdivision method, clustering, implementation of the discontinuity meshing, participation of the media. The system will be extended to calculate thermal load of the buildings and to take into account other effects influencing the habitant's environment and their feelings.

Acknowledgements

This project has been partly supported by the Ministry of Education, Youth and Sports of the Czech Republic under research program No. Y04/98: 212300014 (Research in the area of information technologies and communications).

8. REFERENCES

[Aut02a] Autodesk, Inc. Lightscape. http://www.autodesk.com/lightscape, 2002

[Bek99a] Bekaert, P. Hierarchical and Stochastic Algorithms for Radiosity. PhD thesis, Katholieke Universiteit Leuven, 1999.

[Bek00a] Bekaert, P., Dutre P. and Suykens, F. Renderpark a Photorealistic Rendering Tool. http://www.renderpark.be, 2000.

[Coh88a] Cohen, M., Chen, S. E., Wallace, J. R. and Greenberg, D. P. A progressive refinement approach to fast radiosity image generation. Computer Graphics (SIGGRAPH '88 Proceedings), 1998.

[Coh93a] Cohen, M. F. and Wallace, J. R. Radiosity and Realistic Image Synthesis. Academic Press Professional, Boston, MA, 1993.

[Dau97a] Daubert, K., Schirmacher, H., Sillion, F. and Drettakis, G. Hierarchical Lighting Simulation for Outdoor Scenes. In Rendering Techniques'97, NY, 1997.

[Gla90a] Glassner, A. S. Graphics Gems. Academic Press, Inc., San Diego, CA, 1990.

[Gor93a] Gortler, S. J., Schröder, P., Cohen, M. F. and Hanrahan, P. M. Wavelet radiosity. Computer Graphics (SIGGRAPH '93 Proceedings), 1993.

[Gra01a] Granier, X. and Drettakis, G. Incremental Updates for Rapid Glossy Global Illumination. In Proceedings of EUROGRAPHICS'01, 2001.

[Han91a] Hanrahan, P., Salzman, D., and Aupperle, L. A rapid hierarchical radiosity algorithm. Computer Graphics (SIGGRAPH '91 Proceedings), 1991.

[Hav00a] Havran, V. Heuristic Ray Shooting Algorithms. PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.

[Int02a] Integra, Inc. Inspirer. http://www.integra.co.jp/eng, 2002

[J3d02a] The Java3D Community Site. http://www.j3d.org, 2002

[ISO97a] ISO. 14772-1. Information Technology – Virtual Reality Modelling Language. International standard ISO/IEC, 1997.

[Jen02a] Jensen, H. W. and Buhler, J. A Rapid Hierarchical Rendering Technique for Translucent Materials. (SIGGRAPH'02 Proceedings), 2002.

[Kaj86a] Kajiya, J., T. The Rendering Equation. In Computer Graphics (SIGGRAPH '86 Proceedings), 1986.

[Lar98a] Larson, G. W., and Shakespeare, R. Rendering with Radiance: The Art and Science of Lighting Visualization. Morgan Kaufmann, San Francisco, CA, 1998.

[Mul95a] Müller, S., Kresse, W., Gatenby, N. and Schöffel, F. A Radiosity Approach for the Simulation of Daylight. In Rendering Techniques'95, NY, 1995.

[Smi00a] Smits, B. and Jensen, H. W. Global Illumination Test Scenes. Technical Report UUCS-00-013, University of Utah, 2000.

[Whi80a] Whitted, T. An improved illumination model for shaded display. Communications of the ACM 23 (6): 343-349, 1980.

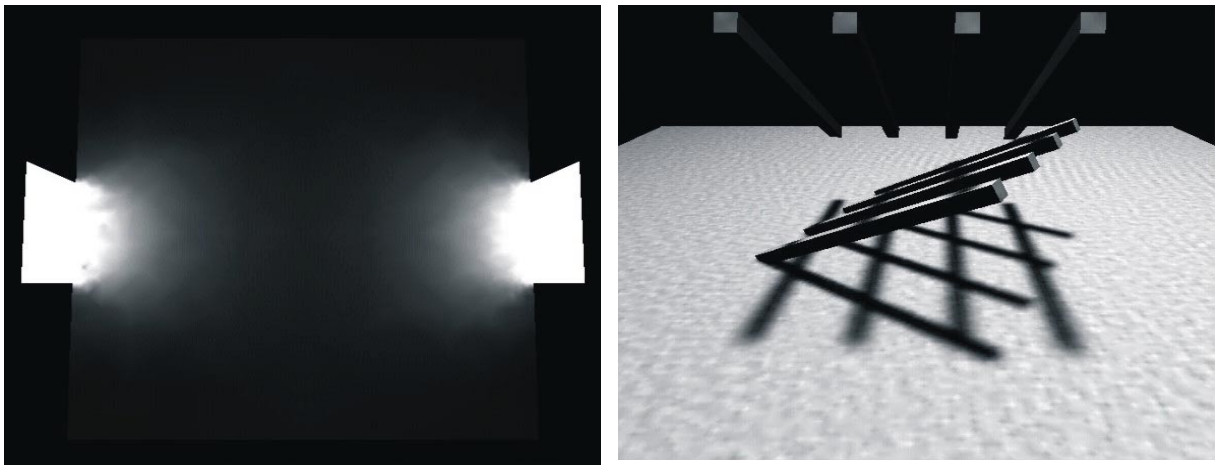

Figure 7. Global Illumination Test Scenes.
Left image shows scene “Emission”, right image shows scene “Shadow”. Both were computed with total amount of approximately 10^6 rays per iteration and exhibit high frequency noise.

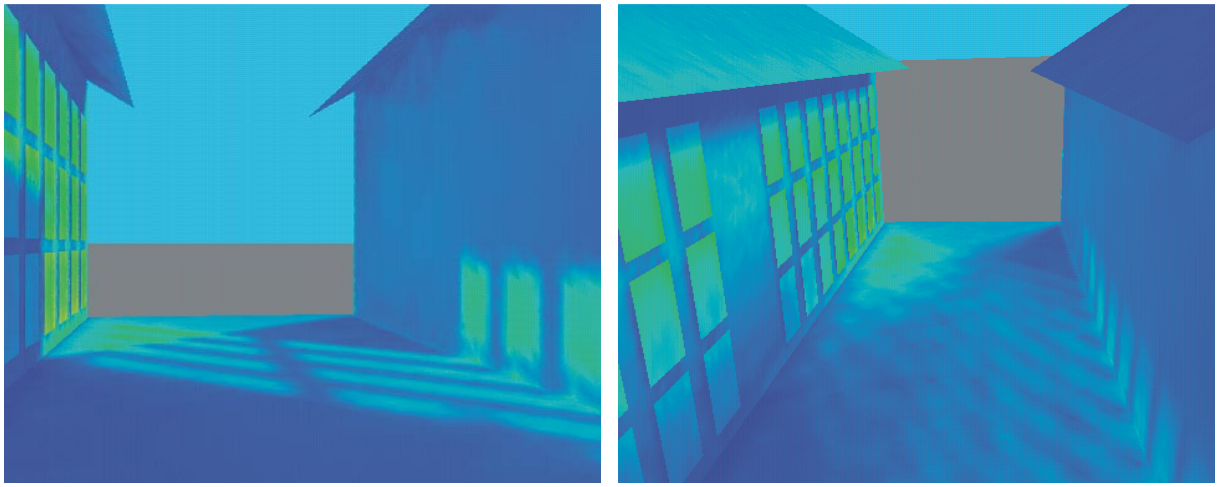

Figure 8. Light load for two simple architectural scenes. On the left we can see results for a single light source, the right image shows a cumulative result for 1 hour during a sunny day.

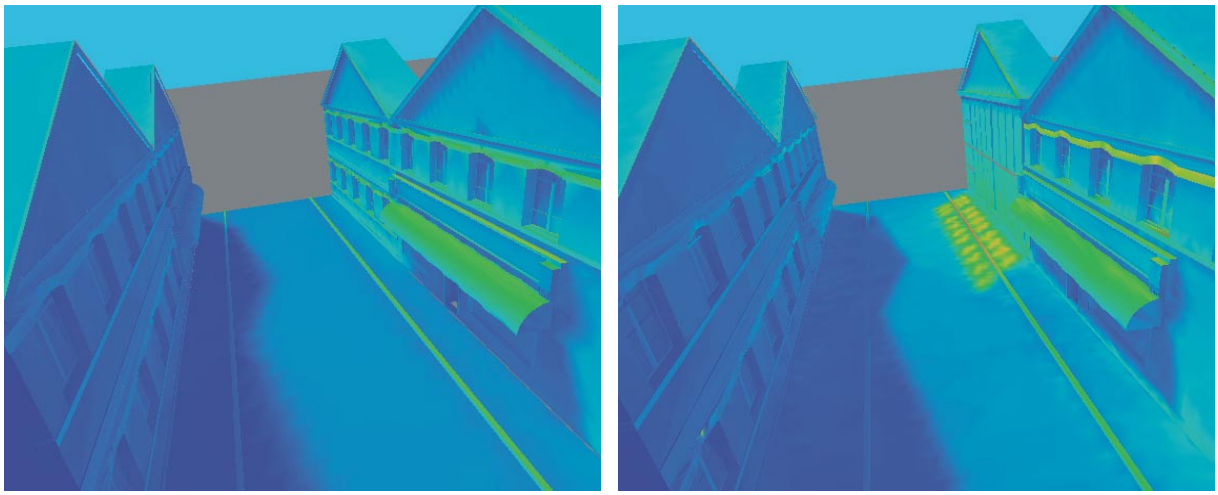

Figure 9. Visualization of the light load in a street. On the left image we can see light load in an original street, the right image shows light load in this street after replacing an old building on the right side by the new one made of more reflective materials.