REAL-TIME DYNAMIC RADIOSITY RELIGHTING OF VIRTUAL ENVIRONMENTS

Kasper Høy Nielsen **Niels Jørgen Christensen**

Informatics and Mathematical Modelling
The Technical University of Denmark
DK 2800 Lyngby, Denmark
{khn,njc}@imm.dtu.dk

ABSTRACT

This paper describes a method for dynamic changing of colours and intensities of light sources in a radiosity-lit environment. We introduce a fast radiosity sampling approach where energy is sorted with respect to the emitting lights. The idea is to tag energy with a light source identifier in order to determine from which light source energy is coming from, either directly or indirectly. Based on this information the subsequent reconstruction allows for interactive changing of light source emissions, as long as the geometry in the environment remains static. We illustrate this concept by developing a modified progressive refinement algorithm that performs efficient concurrent sampling of separate light source solutions. We show that the result is useful for real-time animation of realistic lighting in virtual environments. Furthermore, we describe how the method can be adapted to handle near real-time animation of moving lights.

Keywords: Radiosity, relighting, progressive refinement, dynamic lighting.

1 INTRODUCTION

Radiosity is a popular method for illuminating environments for use in interactive walkthroughs. However, because radiosity is computationally expensive, such environments are traditionally inherently static. Different methods have been developed for handling dynamic environments, where radiosity solutions are updated to match the changed environment. It is inefficient to simply recompute a new radiosity solution. Instead, the challenge is to use information already calculated by taking advantage of scene and illumination coherence.

Previous work has mainly focused on methods for repropagation of light in completely dynamic environments [Baum90, Chen90, Georg90, Puech90, Dorse95a, Schof95]. However, such method can be computationally expensive in environments where only light source properties are dynamically changed.

This paper specifically explores the interactive changing of light source intensities and colours in a radiosity-lit environment. That is, an environment where the illumination is dynamically changed, but where the geometry remains static. We present a sampling approach that allows for efficent precomputation of illumination, and, during reconstruction, allows for interactive lighting changes based on the precomputed solution. Although limited to static geometry, dynamic lighting is useful for updating diffuse illumination when using hybrid rendering approaches, e.g. where illumination of static and dynamic geometry is handled differently, or simply to bring life to a diffuse walkthrough.

1.1 Background Information

Dynamic radiosity methods, such as [Chen90, Georg90, Puech90, Schof95], handle lighting changes by shooting ”negative light”. However, this is expensive if it affects a large part of the scene [Chen90]. Other methods, such as [Baum90, Besui98], assume that the animation is given *a priori*, which makes them unsuitable for interactive rendering. Lighting changes can also be performed given a full matrix radiosity solution. However, a full matrix solution is generally expensive in both time and space and is

normally not used in practice. Several authors [Airey90, Cohen93, Devil94, Dorse95b, Paul95] have also described how the contributions of different light sources may be computed independently and then linearly combined: Airey et al. [Airey90] have demonstrated that this allows for interactive brightening and dimming of lights. Similarly, Dorsey et al. [Dorse95b] have simulated time-variant lighting using linear combinations of static images, each representing a radiosity solution for a single bank of light sources. However, in [Airey90, Dorse95b] each solution was computed separately for each bank of lights which is computationally expensive.

1.2 Our Contribution

We propose a faster radiosity sampling approach where energy is sorted with respect to the emitting light sources. We illustrate this concept by developing a modified progressive refinement algorithm. However, the concept may just as well be adapted to other radiosity methods (e.g. hierarchical radiosity).

Based on the precomputed solution, and the principles of [Airey90, Dorse95b], the subsequent reconstruction allows for real-time re-illumination of static scenes on low-end graphics hardware. We demonstrate this, and also describe how the method can be adapted to handle moving light sources.

We refer to the dynamic changing of light source properties as *radiosity relighting* [Niels00]. In the next section we describe our method in detail. Section 3 discusses the results, and section 4 draws the conclusion and points out directions of future work.

2 RADIOSITY RELIGHTING

Radiosity relighting is a simple approach for re-illumination of a static radiosity scene in which properties of fixed light sources are dynamically changed. In the following subsection we review the basic concept of decomposing a radiosity solution into several simpler solutions. In section 2.2 we describe how these solutions can be obtained concurrently by using our new approach. Section 2.3 explains how the solutions are combined to allow dynamic lighting. Finally, section 2.4 describes how the method can be adapted for use in non-static environments.

2.1 Decomposing the Radiosity Solution

Assuming we have a radiosity solution with k contributing light sources, it can be split into k independent radiosity solutions, one for each light source.

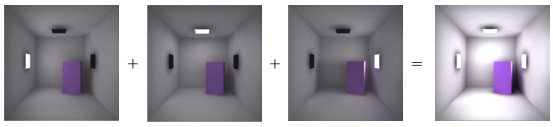


Figure 1: A radiosity solution composed of a set of independent solutions for each contributing light source. In this example, three contributing light sources form the final solution.

Hence, by solving a set of radiosity solutions for the contributing lights, we can linearly combine these into the final solution [Airey90, Cohen93, Devil94, Dorse95b, Paul95]. See Fig. 1. Presuming we have computed k such solutions for a static environment we can form the final solution. More importantly, we can change any of these solutions independently, without recomputing any of the other solutions. Because only one light source is active in each solution, it is possible to dim or change the colour of a light source by directly modulating the calculated energies. A light source can be turned off by completely disregarding its contribution.

By only solving the individual radiosity solutions for bright white light (unit emission) [Cohen93], we ”pull” the colour and intensity of a light source outside the radiosity solution. This can then be applied in the final reconstruction step, solely considering the contribution from each light source. The resulting radiosity solution can be thought of as element energies, that tell us in what way light is changed or modulated as it propagates through the scene. In this respect, the calculated energies are only a function of reflectance and form factors, from which we determine in what way light is altered, from it leaves a light source, until it is received by an element.

Since radiosity is solved with unit emission, all lights are assumed to be equally important. However, if this is not the case, small emission values can be used for less important light sources. Finally, it can be advantageous to group several lights into a single contribution, or bank of light sources, to limit the number k [Dorse95b].

2.2 Our Modified Radiosity Sampling Method

Solving a separate radiosity solution for each light source is expensive compared to solving only one solution, because more form factors consequently have to be calculated. Instead of solving k radiosity solutions independently, we want to find the solutions concurrently. This avoids the complexity of handling k independent radiosity solutions and reduces the number of form factor calculations.

We observe that standard radiosity methods treat energies alike, and that there is no distinction as to from where light originates. For example, by looking at an element, we can see that it receives a fraction of energy, but we cannot determine which light sources the energy is coming from, either directly or indirectly. To solve this, we propose a scheme where energy is tagged with a light source identifier. This information enables us to sort received fractions of energy for each element with respect to the light source from which the energy originates, thereby solving k separate radiosity solutions concurrently.

We illustrate this concept by developing a modified version of the progressive refinement algorithm with substructuring. Similar to a normal progressive refinement algorithm, we initialize emitters with maximum energy and receivers (non-emitters) with zero energy. At the same time we tag emitters with a light source id. During successive progressive refinement iterations, light is shot from the patches with most energy as usual. When distributing energy to the elements, fractions of energy are added separately with respect to the light source id. The pseudo code for the modified progressive refinement algorithm is shown in Fig. 2. In a scene with k light sources, the algorithm computes k separate radiosity solutions. A non-substructured algorithm is shown for clarity.

The time-complexity for one shot in the algorithm is $O(n * k)$, where n is the number of elements, as energy is distributed separately with respect to the originating light sources. Thus, compared to the time-complexity of standard progressive refinement, $O(n)$, this is a factor of k more expensive. This is identical to the time-complexity for solving each solution separately, $O(n * k)$, for one iteration. Hence, at the first glance this does not seem like an improvement. However, when solving each solution separately, form factors to every element in the scene are calculated $k * i$ times, for i number of iterations. For the same number of shots, this is reduced to i times, regardless of k, in the algorithm shown in Fig. 2. Since the determination of form factors is computationally expensive, this can improve performance.

```
for (each light source m∈[1:k]) {
    for (each element i) {
        if (i is light source m) B_{i,m} = 1
        else B_{i,m} = 0
        B_{unshot,i,m} = B_{i,m}
    }
}
while (not converged) {
    Pick i, so Σ_m B_{unshot,i,m}*A_i is largest
    Calculate all form factors F_{ij}
    for (each light source m∈[1:k]) {
        for (each element j) {
            ΔB = B_{unshot,i,m}*ρ_j*F_{ij}
            B_{unshot,j,m} += ΔB
            B_{j,m} += ΔB
        }
        B_{unshot,i,m} = 0
    }
}
```

Figure 2: Pseudo code for the modified progressive refinement algorithm.

2.3 Reconstruction with Dynamic Lighting

After the progressive refinement has converged or been terminated, the result is a scene where each element has a list of energy amounts received directly or indirectly from the k contributing light sources. The final solution is computed by determining the combined energy received by each element j in the scene. This energy is found by summing the energy contributions from the k light sources multiplied by the actual emitted energy from each light source [Airey90, Cohen93, Devil94, Dorse95b, Paul95]:

$$B_{j,combined} = \sum_{i=1}^{k} B_{j,i} E_i \quad (1)$$

The time-complexity for computing the final solution in a scene with n elements and k contributing light sources is thus $O(n * k)$. Compared to a full matrix solution, $O(n^2)$, it is therefore both faster and less memory intensive, but also more limited in that we can only change emission properties for selected emitters instead of for arbitrary elements. Thus, the method is only advantageous if the number of light sources is small compared to the number of elements.

As in normal progressive refinement, an ambient contribution can be added to the solution. This contribution is calculated by summing k separate ambient contributions, one for each light source solution, to form the global ambient term. As with elements, these terms can be modulated directly by the properties of the light sources.

The reevaluation and update of n elements can be computationally expensive for interactive scenes of moderate complexity. An alternative method is to take advantage of radiosity textures [Myszk94] and graphics hardware. As noted by Dorsey et al. [Dorse95b], hardware-blending can be used for fast scaling and accumulation of an image-based solution. By storing each of the k solutions in separate sets of radiosity textures the final solution can be reconstructed using texture mapping, and rendered in k passes. The modulation of colours can be achieved by using simple vertex shading. However, since graphics hardware is used to store and combine element colour contributions from individual light sources, this method can suffer from quantization errors due to the limited dynamic range of the graphics hardware, especially in scenes with many lights.

2.4 Sampling and Reconstruction of Dynamic Scenes

The sampling algorithm can be extended to categorize energy with respect to direct/indirect illumination, storing these separately. This can be advantageous for dynamic radiosity methods, e.g. where direct light is changed while indirect light remains fixed (as in [Dorse95a]).

The method can also be adapted to render radiosity scenes with moving light sources: By regarding patches as light sources during sampling, we obtain a solution in which the emitted energy from each patch can be changed individually. During rendering, energy is shot from the moving lights into the scene, and the emitted energy of each patch is adjusted to match the reflected energy. The final radiosity solution is then found as the sum of the direct illumination from the moving lights and the relighted solution based on the emitting patches. Although the method is fast compared to a recomputation of radiosity, it can be memory intensive for scenes with many patches.

Dynamic objects can also be inserted into the scene. Although these will cast shadows, they will not transmit or receive indirect illumination, nor cast shadows due to the indirect illumination.

3 RESULTS AND DISCUSSION

Results are split into two parts: We first conduct a comparison of our modified progressive refinement method to standard progressive refinement (sampling). We then examine the results for interactive rendering with dynamic lighting (reconstruction).

3.1 Sampling

Our modified algorithm is compared to a standard progressive refinement method. Both use uniform meshing and fixed substructuring, and the algorithms are terminated after 100 shots. A 128×128 hemicube was used for form factor computations. Fig. 3 shows the used test scenes. Without light sources the test scenes consist of 12 polygons, meshed into 60 patches and 960 elements. For fast radiosity sampling our implementation uses texture maps to discretize the scene using graphics hardware [Niels00]. All tests were conducted on an Intel Pentium III 450 MHz PC, with 128 MB RAM, equipped with an Nvidia GeForce 256 graphics accelerator. The results are shown in Table 1. The sampling tests show that our modified algorithm is slightly (4.07/3.2 = 27%) slower than standard progressive refinement for a combined solution. This is due to the fact that

(a) (b) (c)

Figure 3: The test scenes with (a) 1, (b) 3 and (c) 8 light sources, shown after 100 shots.

Number of lights	1	3	8
(1) Standard P.R. (combined)	3.20s	3.20s	3.20s
(2) Standard P.R. (separate)	3.20s	9.62s	25.65s
(3) Modified P.R. (separate)	3.29s	3.54s	4.07s

Table 1: The table shows the measured runtimes for obtaining each solution: (1) standard progressive refinement solved for one scene using all light sources. (2) solving a progressive refinement solution for each light source separately, combined by superpositioning the resulting images. (3) solutions found using our modified progressive refinement algorithm.

the modified algorithm distributes energy for each of the k contributing lights separately, that is, $O(n * k)$ instead of $O(n)$. Yet, the speed-difference is not very large: 0.09s, 0.34s and 0.87s for each test scene, which corresponds to the k dependence. Hence, it is the total number of form factor calculations that accounts for the majority of the computation time, and these are exactly the same in the two methods. The methods otherwise produce identical results. Our modified algorithm produces a solution where each light source contribution is stored separately. We therefore also compare it to an equivalent method, where a progressive refinement solution is found separately for each light source. Because an isolated solution is found for each light source, the method is expensive compared to our proposed alternative, e.g. in the solution using 8 light sources, the modified algorithm is up to (25.65/4.07) ≈ 6 times faster than the standard method. This is a result of the form factor calculations involved in 100*8 unsorted shots compared to only 100 sorted shots.

3.2 Reconstruction

After sampling, the lighting can be changed interactively during reconstruction. The reconstruction is performed for each frame. The final solution is stored in radiosity textures for fast rendering. The measured results are shown in Table 2 and different images from the scenes with changed lighting settings are shown in Fig. 4 [1]. The images have been verified to match standard progressive refinement solu-

[1] Animated movie sequences can be downloaded from *http://www.imm.dtu.dk/∼khn/rrl/*

Number of lights	1	3	8
Software Reconstruction:			
Rec. time (1 frame)	1.0 ms	2.6 ms	7.3 ms
Frames per second	90 fps	78 fps	59 fps
Used memory	12 KB	35 KB	95 KB
Hardware Reconstruction (radiosity textures):			
Rec. time (1 frame)	0.9 ms	2.6 ms	8.3 ms
Frames per second	142 fps	120 fps	66 fps
Used memory	12 KB	35 KB	95 KB

Table 2: The table shows the measured time used for software- and hardware-based reconstruction, the average frame rate, and their memory consumption.

tions with identical lighting settings. The tests show that the reconstruction time responds linearly to the number of lights, i.e. the expected $O(n * k)$, where n is the number of elements and k the number of light sources. When using software reconstruction, the reevaluation of n colours is expensive to perform for each frame. A faster runtime is achieved by storing the k solutions in radiosity textures and utilizing hardware texture mapping for reconstructing the solution in k passes (see Table 2). This corresponds to $O(t * k)$, where t is the number of radiosity textures. We perform these passes in texture space rather than screen space to minimize the number of drawn pixels. Because our hardware uses a 24 bit texel representation, the method causes visible quantization artifacts when using many lights, e.g. more than 10 lights in the tested scenes. Yet, such artifacts can be diminished by using a larger texel representation if this is supported by the hardware.

Using our test system a user can move around in a scene and change lighting settings in real-time. Fig. 5 shows relighting of more complex scenes. Finally, we have also used the method to render simple radiosity scenes with moving light sources. See Fig. 6. Here, the scene is subdivided into 30 patches. Since patches are regarded as light sources, the per frame reconstruction performs a weighted blend of 30 radiosity solutions for each element in the scene. We observe that the method does allow for realistic lighting of simple environments in near real-time. Yet, the method is expensive for scenes with many patches, and rendering of more complex scenes will clearly not be real-time.

4 CONCLUSION AND FUTURE WORK

We have shown that the proposed modified progressive refinement algorithm works as expected. Compared to a standard progressive refinement, our method executes illumination changes in fractions of the time it takes to recompute a complete progressive refinement solution, at the cost of a slightly slower sampling time. Both runtimes depend linearly on the number of light sources k, and uses $O(n * k)$ in both time and memory. The method is therefore advantageous for scenes with relatively few lights.

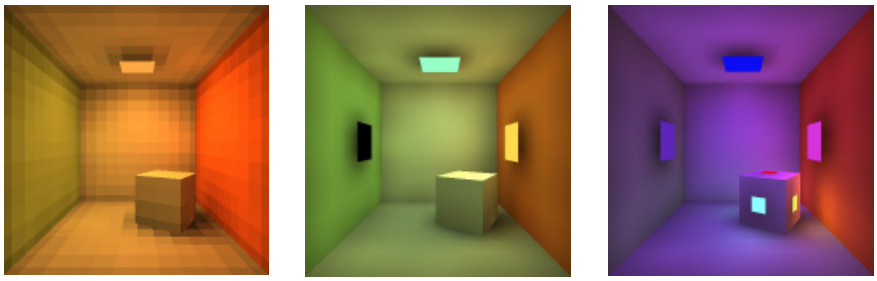
We have compared the method to an alternative approach, where a separate progressive refinement solution is found for each light source. Our method can be used to achieve the same results, but was found to be up to 6 times faster than the alternative sampling method. The reason for the increased speed is simply that our method uses the same form factors for distributing energy from k different light sources for each iteration. This reduces the number of form factor calculations, thereby improving performance. We have shown that the method can be used for fast reconstruction of scenes with changing lighting conditions, and verified the expected runtimes. Finally, we have shown that the method can be used for near real-time rendering of radiosity scenes with moving light sources.

The radiosity method is modified so that energy is tagged and sorted according to the originating light source. We have demonstrated that this can be integrated into a regular progressive refinement algorithm. However, it should be possible to integrate this concept into more advanced radiosity methods, including dynamic radiosity methods. This should be investigated in future work.

The time-complexity of both sampling and reconstruction depends linearly on the number of light sources. This makes real-time lighting changes possible, and is in our opinion advantageous compared to alternative methods, since fast lighting changes are achieved by an inexpensive recalculation of the energy distribution. This is done at the cost of a larger memory consumption. Thus, if memory is not critical but speed is, we believe that this approach is a useful alternative to traditional dynamic radiosity methods.

5 ACKNOWLEDGEMENTS

The authors would like to thank Tomas Akenine-Möller for his encouragement and helpful comments. Special thanks to Andreas Bærentzen for suggesting that the method could be adapted to handle moving lights. Thanks also to Bent Dalgaard Larsen for proof-reading the draft. This work was supported in part by the STVF project DMM and the Nordunit2 project NETGL.



(a) (b) (c)

Figure 4: Images for interactive relighting of the scenes using (a) one, (b) three, and (c) eight light sources. Image (a) is shown without bilinear texture filtering.

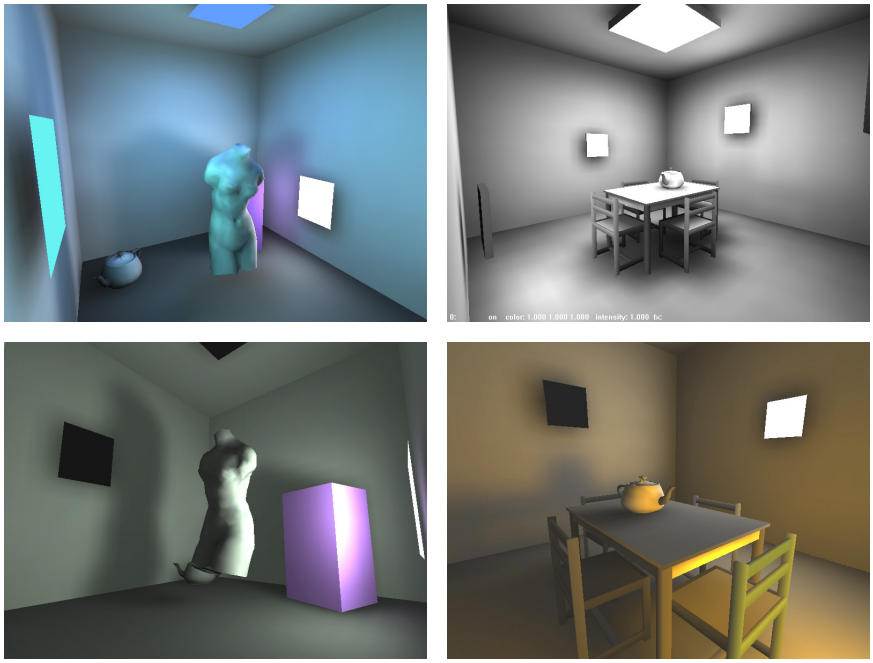
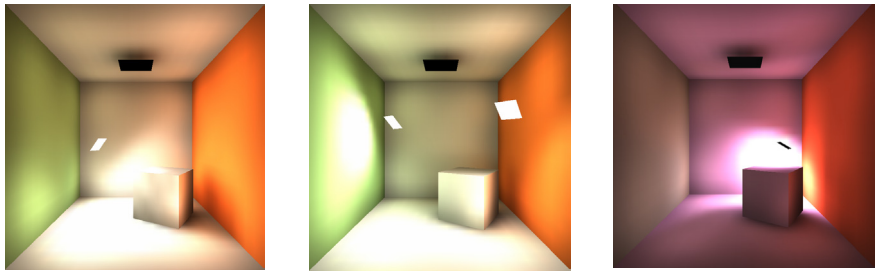


Figure 5: Images from interactive relighting sessions on two scenes: (left) The ”Venus” scene contains 3 light sources and 3517 triangles meshed into 2306 patches and 6221 elements (300 KB). The solution (100 shots) was found in 22 seconds, and relighting ran at 24 fps. (right) The ”Table” scene contains 5 light sources and 1111 triangles meshed into 1344 patches and 5874 elements (423 KB). The solution was found in 16 seconds, and relighting ran at 15 fps.



(a) (b) (c)

Figure 6: Images for relighting of a test scene with moving light sources: (a) using one light source (5 fps), (b) two lights (3 fps), and (c) one light with changing colors (5 fps). The images are rendered using software reconstruction. The scene consists of 30 patches and 1920 elements, and the total memory consumption for the 30 solutions was 690 KB. The light sources themselves do not cast shadows.

REFERENCES

[Airey90] Airey, J. M., Rohlf, J. H., Brooks, F. P.: Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments, Computer Graphics (ACM Workshop on Interactive Graphics Proc.), pp. 41-50, 1990.

[Baum90] Baum, D. R., Winget, J. M.: Real Time Radiosity Through Parallel Processing and Hardware Acceleration, Computer Graphics (SIGGRAPH'90 Proceedings), Vol.24, No.2, pp. 67-75,1990.

[Besui98] Besuievsky, G., Pueyo, X.: A Dynamic Light Sources Algorithm for Radiosity Environments, EGCAS (Eurographic Workshop on Computer Animation and Simultaion), Springer, 1998.

[Chen90] Chen, S. E.: Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System. Computer Graphics (SIGGRAPH'90 Proceedings), pp. 135–144, 1990.

[Cohen88] Cohen, M. F., Chen, S. E., Wallace, J. R., Greenberg, D. P.: A Progressive Refinement Approach to Fast Radiosity Image Generation. Computer Graphics (SIGGRAPH'88 Proceedings), pp. 75-84, 1988.

[Cohen93] Cohen, M. F., Wallace, J. R.: Radiosity and Realistic Image Synthesis, Academic Press, 1993.

[Devil94] Deville, M., Merzouk, S., Paul, J. C.: Modeling light source for accurate simulations. In Proceeding of the International Conference Computer Graphics International'94, 1994.

[Dorse95a] Dorsey, J., Nimeroff, J., Rushmeier, H.: A Framework for Global Illumination in Animated Environments, Rendering Techniques'95, Springer, pp. 92-103, 1995.

[Dorse95b] Dorsey, J., Arvo, J., Greenberg, D. P.: Interactive design of complex time-dependent lighting, IEEE Computer Graphics and Applications, 15(2), pp. 26-36, 1995.

[Georg90] George, D. W., Sillion, F. X., Greenberg, D. P.: Radiosity Redistribution for Dynamic Environments, IEEE Computer Graphics and Applications, Vol. 10, No. 4, pp. 26-34, 1990.

[Myszk94] Myszkowski, K., Junii, T. L.: Texture mapping as an alternative for meshing during walkthrough animation. Photorealistic Rendering Techniques, pp. 389–400. Springer, 1994.

[Niels00] Nielsen, K. H.: Real-Time Hardware-Based Photorealistic Rendering. Master's Thesis, Informatics and Mathematical Modelling (IMM), Technical University of Denmark, 2000.

[Paul95] Paul, J. C., Deville, P. M., Winkler, C.: Modeling radiative properties of light sources and surfaces. The Journal of Visualization and Computer Animation, 1995.

[Puech90] Puech, C., Sillion, F. X., Vedel, C.: Improving Interaction with Radiosity-based Lighting Simulation Programs, Computer Graphics (1990 Symposium on Interactive 3D Graphics), pp. 51-57, 1990.

[Saxe96] Saxe, E., Hughes, M., Lastra, A. A.: Higher-order color interpolation for real-time radiosity display, UNC-CH Department of Computer Science Technical Report TR96-023, 1996.

[Schof95] Schöffel, F., Müller, S.: Fast Radiosity Repropagation for Interactive Virtual Environments using a Shadow-Form-Factor-List, G. Sakas et al. (Editor), Photorealistic Rendering Techniques, Springer, pp. 339-356, 1995.