GPU Radiosity for Triangular Meshes with Support of Normal Mapping and Arbitrary Light Distributions

Günter Wallner
University of Applied Arts Vienna, Austria
Oskar Kokoschka Platz 2
1010 Vienna
wallner.guenter@uni-ak.ac.at

ABSTRACT

This paper describes an implementation of a progressive radiosity algorithm for triangular meshes which works completely on programmable graphics processors. Errors due to the rasterization of triangles are fixed in a post-processing step or with a fragment shader during runtime. Adaptive subdivision to increase the accuracy of the radiosity solution can be performed during render-time. Since we found that the gradient is not very robust to determine whether triangles should be subdivided or not, we propose a new technique which uses hardware occlusion queries to determine shadow boundaries in image space. The GPU implementation facilitates the simple integration of normal mapping into the radiosity process. Light distribution textures (LDTs) enable us to simulate a variety of real world light sources without much computational overhead. The derivation of such an LDT from a EULUMDAT file is described.

Keywords Radiosity, Global Illumination, GPU, Normal Mapping, Shadow Boundary Detection, Light Distribution Texture

1 INTRODUCTION

Computer image generation has been driven by two major factors: realism and interactivity. The former has led to a variety of global illumination algorithms such as radiosity. Radiosity was first introduced to computer graphics by Goral et al. [Gor84] to simulate the light interaction in strictly diffuse environments. The fraction of the radiant light energy leaving one particular surface which strikes a second surface is defined as the so-called form factor. These form factors can be obtained by computing the coifficients of a set of linear equations. Cohen et al. [Coh85] introduced the hemicube to support scenes with occluded surfaces, which where not considered in the original implementation. In [Coh88], Cohen et al. presented a progressive refinement approach which eliminated the $O(n^2)$ storage requirements of former methods by calculating the form factors on-the-fly. Further speed ups can be gained by implementing the substructuring approach from Cohen et al. [Coh86] where light is shot from a courser mesh to a finer sets of elements. Smits et al. [Smi92] published a radiosity implementation which focuses on those parts of the scene which affect an image most. Although radiosity is usually restricted to diffuse surfaces, generalizations of the radiosity method which can handle general reflectance (e.g. by Sillion et al. [Sil91]) and volumetric scattering due to participating media like smoke ([Rus87]) have been proposed. A comprehensive treatment of the radiosity method can be found in [Sil94] and [Coh95]. However, for completeness the essential features are reviewed in Section 2.

Figure 1: GPU radiosity solution of a scene with 6534502 elements distributed over 13627 triangles.

Modern GPUs opened up a whole new research area, allowing researchers to compute a radiosity solution in much faster time or even at interactive rates. Keller [Kel97] generates a particle approximation of the diffuse radiance in the scene using quasi-Monte Carlo integration. Afterwards, the graphics hardware renders an image with shadows for each particle which are considered as point light sources. Martin et al. [Mar98] calculated a hierarchical radiosity solution on the CPU and refined the result by generating textures that represent the diffuse illumination. Nielsen and Christensen

[Nie02] accelerated the hemicube method using graphics hardware. Carr et al. [Car03] used floating point textures to store the result of the radiosity computation. Gautron et al. [Gau05] adapted the irradiance cache ([War88]) to graphics hardware. However, all of these publications used graphics hardware to accelerate certain elements of the radiosity solution. Coombe et al. [Coo03] finally proposed a progressive radiosity implementation which worked solely on the GPU.

This paper follows the approach by Coombe et al. but extends it to arbitrary triangular meshes. Further contributions are the inclusion of normal mapping into the radiosity process, the support of arbitrary light distributions due to the use of light distribution textures (LDTs) and new way to determine shadow boundaries for adaptive subdivision. Figure 1 shows a radiosity solution obtained with our method. The reminder of this paper is structured as follows. Section 2 reviews in short the basics of radiosity and the progressive radiosity approach and Section 3 describes our implementation. Adaptive subdivision is explained in Section 4. We conclude the paper by presenting results and sample images (Section 5) as well as future work (Section 6).

2 PROGRESSIVE RADIOSITY

The radiosity method evaluates the intensity (or radiosity) of discrete points and surface areas in an diffuse environment. The radiosity B_i of an element i is given by [Gor84]

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} \tag{1}$$

where E_i is the emission, ρ_i the reflectivity and F_{ij} the form factor between element i and j. F_{ij} is purely geometrical in nature and describes the fraction of energy leaving element j impinging on element i. If using the disc approximation of Wallace et al. [Wal89] to the differential form factor equation (Figure 2), F_{ij} is given by

$$F_{dA_i,A_j}(=F_{ij}) = A_j \sum_{i=1}^{m} \frac{\cos(\phi_i)\cos(\phi_j)}{d^2\pi + \frac{A_j}{m}} \tag{2}$$

where m is the number of sampling points on A_j. As noted by Coombe et al. [Coo03] this disc approximation reduces artifacts between adjoining faces exhibited by the original form factor formulation [Gor84] when used in conjunction with projection methods, like the hemicube approach by Cohen et al. [Coh85]. To assure conversation of energy in a closed environment the sum of all form factors for a given element i is equal to unity:

$$\sum_{j=1}^{n} F_{ij} = 1 \quad \text{for } i = 1 \ldots n \tag{3}$$

Contrary to the conventional radiosity algorithm, where all the form factors for the entire scene are precalculated, form factors are calculated on-the-fly in a progressive radiosity solver. Furthermore, shooting is always performed from the element radiating the most light energy, since those typically have the greatest impact on the illumination, leading to a solution which converges quickly in regard to accuracy. Additionally an ambient radiosity term

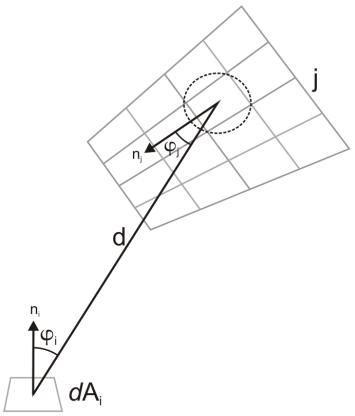

Figure 2: The form factor between a differential area dA_i and a polygon j which is divided into m sections. Each section gets approximated by a disc.

$$A = R \sum_{j=1}^{n} \Delta B_j F'_{ij} \quad \text{for any } i \tag{4}$$

was introduced by Cohen et al. [Coh88] to estimate reflected light in the earlier iterations, yielding a more adequate illumination during early stages. ΔB_j represents the unshot radiosity. F'_{ij} is a first approximation to the form factor and is given by

$$F'_{ij} = \frac{A_j}{\sum_{k=1}^{n} A_k} \quad \forall i \tag{5}$$

and the interreflection factor R is defined as

$$R = \left(1 - \frac{\sum_{k=1}^{n} \rho_k A_k}{\sum_{k=1}^{n} A_k}\right)^{-1} \tag{6}$$

3 IMPLEMENTATION

This section describes our radiosity implementation in detail. For each triangle two 32bit RGBA floating point textures are stored which hold the radiosity and residual energy respectively. The RGB components are used to store the illumination and the alpha channel is used to determine if a particular texel of the texture is occupied by the triangle ($A = 1$) or not ($A = 0$).

In a preprocessing step each triangle is rendered orthographically into a framebuffer of size $(2^n - 2) \times (2^n - 2)$. During rendering an occlusion query is issued to retrieve the number of texels occupied by the triangle. The area of a single element of a triangle can then be obtained by dividing its area with the result of the occlusion query. Note that due to partially covered pixels the area of an element is slightly underestimated. However, we found that no significant error is caused by this.

The texture coordinates are retrieved by multiplying the vertex coordinates with the modelview-projection matrix used for rendering and shifting the values to the range $[0,1]$. The result is then centered in a texture of size $2^n \times 2^n$ to allow for interpolation in the post-processing step. Furthermore all textures are placed in a texture atlas of size $2^m \times 2^m$ to reduce the number of texture switches during the radiosity process and the number of readbacks during the next shooter selection. All textures have power-of-two dimensions to allow for mipmapping.

After the preprocessing step the progressive radiosity solver starts until the result has converged or a maximum number of iterations has been reached. At the beginning of each iteration the next shooter is determined. To find the triangle with the highest residual power the nth level of the mipmap pyramid is constructed from each residual texture atlas using a fragment program and a ping-pong rendering scheme. This results in a texture where each texel corresponds to the averaged residual intensity ($I = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B$) of a triangle. The reasons for a fragment program are twofold. First, graphics hardware may or may not support hardware mipmapping for floating point textures. Second, only texels which are occupied by the triangle may influence the average. Therefore our fragment shader only averages texels whose alpha channel equals one. The alpha channel of the new texel is set to one if one of the four original texels alpha value is one. The values are read back and multiplied by the area of the corresponding triangle to retrieve the residual power. The resulting values are compared and the triangle with the highest residual power is chosen as next shooter. During the next shooter selection the ambient radiosity term from Equation 4 is calculated. Since the average residual energy of a triangle is evaluated nevertheless and the overall interreflection factor can be precalculated, the computational expense is negligible.

Once a shooter has been selected, all the elements of the triangle shoot their energy in turn. The selected triangle is rendered orthographically into a framebuffer with two color attachments. A fragment shader outputs the interpolated normals and world positions of this triangle. As suggested by Coombe et. al [Coo03], substructuring ([Coh86]) can be supported by constructing a lower resolution mipmap of the residual texture. In our case, we also construct the mipmap from the normal and the world position map. The resulting residual mipmap gets sampled and each texel whose alpha channel equals one shoots its energy.

To determine the visibility from the current shooter, we follow the approach of Coombe et. al [Coo03] and render the scene from the point of view of the shooter using a stereographic projection into a visibility texture. The position and orientation of the shooter are retrieved from the world position and normal map. However, instead of using color-encoded IDs of the polygons, we store the depth values as proposed by Barsi and Jakab [Bar04]. Based on the front (fp) and back clipping distances (bp) the depth value is calculated in a vertex shader as shown in Algorithm 1.

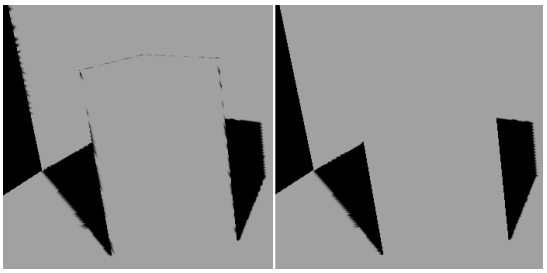

Figure 3: Light Grey parts of the image are visible from the shooter, black ones aren't. If only one texel in the visibility texture is evaluated for depth correspondence artifacts appear near silhouette edges (left). Comparing also the neighboring texels removes those artifacts (right).

```
pos = mul(modelView, position);
float z = (-2*pos.z-bp-fp)/(bp-fp); // [-1..1]
float zDepth = -z/bp // [0..1]
```

Algorithm 1: Vertex shader code for calculation of the depth value

Since only vertices are affected by the stereographic projection, several errors are introduced, especially near the equator of the hemisphere. For example, convex quads may get concave after the projection, which leads to rasterization artifacts. Working with depth values instead of polygon IDs eliminates dot artifacts[1], since a tolerance value can be used when the visibility checks are performed later in the process. Triangles behind the hemisphere are culled away be checking against the plane defined by the position and normal of the shooter. For the remaining triangles, an occlusion query is issued.

Every triangle that might have received energy (triangles which pass the occlusion query test) is rendered orthographically to a framebuffer of size $(2^n - 2) \times (2^n - 2)$. However, instead of back-projecting the texels into the shooter's viewpoint, as done by Coombe et al. [Coo03], the back projection is done in a vertex shader and the resulting position is passed to the fragment shader. This way the same error occurs during back projection as observed in the creation of the visibility texture. The fragment shader compares the depth value of the texel with the depth value stored in the visibility texture. We found that we can further reduce artifacts – mainly in areas of silhouette edges from

[1] Due to the limited resolution of the visibility map and errors introduced by the projection, nearby elements of the scene may be mapped to the same texel.

the shooter's point of view – if we also check the neighboring texels in the visibility texture for correspondence with the – currently examined – texels depth (see Figure 3).

If the texel is declared visible, the form factor equation from Equation 2 is evaluated by the fragment program. The radiosity value is gained by multiplying the form factor, the shooters energy and the color as well as the reflectivity r of the receiver and adding it to the radiosity texture. Respectively the residual texture is updated by taking $1-r$. After all texels of the shooter have shot their energy, the residual texture of the shooter is set to zero.

After the post-process (described in Section 3.1), the floating point textures are tone mapped using either a simple exposure function or a GPU implementation of the global tone mapping operator from Reinhard [Rei02].

3.1 Rasterization of Triangles

According to the OpenGL specification [Seg03] polygons and line segments are rasterized differently. For lines OpenGL uses a "diamond-exit" rule. This means that for each fragment f with center at window coordinates x_w and y_w a diamond shaped region R_f is defined as

$$R_f = \{(x,y) | \|x-x_w\| + \|y-y_w\| < \frac{1}{2}\} \qquad (7)$$

A good description of OpenGL's line rasterization can be found in [Sun03]. For polygons OpenGL follows the point-sampling rule. Only fragments which centers lie inside the polygon are produced by rasterization. Special treatment is given to a fragment whose center lies on a polygon boundary edge (see [Seg03] for details). However, we are not concerned about the exact details because those fragments get rasterized by line-rasterization anyway. Figure 4 shows the rasterization of a triangle. Since not all fragments – which are needed for texturing – are rasterized (these are shown red in Figure 4), the missing fragments are interpolated from the neighbor intensities in a post-processing step. To reduce artifacts due to rasterization, two steps are taken. First, every triangle is rendered twice. One time the polygon itself and next the outline with a line width of 1. It should be noted that using a line width greater than 1 leads to artifacts, since more than one fragment of the line has the same texture coordinate assigned, therefore pointing to the same location in the radiosity map. Second, after the radiosity solver has finished a textured quad is rendered orthographically to a framebuffer at the same resolution as the assigned radiosity texture to establish a one-to-one correspondence with the fragments of the framebuffer. A fragment program linearly interpolates the intensities for fragments which neighbor at least one fragment whose alpha channel is one. Only fragments occupied by the triangle are considered for interpolation. Since the textures are interpolated linearly for rendering, this is done twice, using a ping-pong technique. Fragments produced by this step are marked with black (first iteration) and blue (second iteration) dots in Figure 4. These fragments are only used for display purposes, therefore they are neither considered in the radiosity process nor do they alter the size of a triangle.

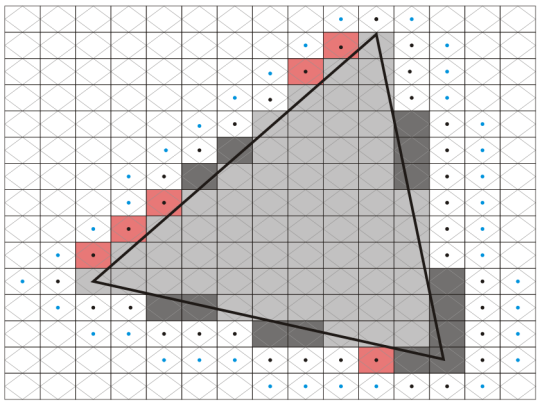

Figure 4: Rasterization of a triangle with OpenGL. Light gray fragments are produced by polygon-rasterization. Dark gray rectangles depict fragments which were produced additionally by line-rasterization. Red fragments represent fragments which would be needed for GL_NEAREST texture sampling but have not been rasterized.

3.2 Light Distribution Textures

To include arbitrary light distributions into the radiosity process, we propose a so called light distribution texture (LDT). These textures can be derived from a EULUMDAT file or any other similar photometric file format. An English translation of the EULUMDAT specification can be found at [Ash]. Concordant do the specification we denote the number of C-planes as m_c. The number of light intensities in a C-Plane (vertical planes through the light distribution) is designated as n_g. Figure 5 shows the light distribution curve of a luminaire and it's 3D representation.

Although the EULUMDAT file stores photometric values and the radiosity method works with radiometric values, the normalized light distribution can be used as is. We can show that the radiant intensity $I_e = kI_v$, where k is some constant and I_v the luminous intensity (an in-depth treatment of lighting engineering can be found, for example, in [Gal04]).

Proof. The radiant intensity can be written as

$$I_e = \frac{d\phi_e}{d\Omega} \qquad (8)$$

where ϕ_e is the radiant flux and Ω the solid angle, and the luminous intensity can be written as

$$I_v = \frac{d\phi_v}{d\Omega} \qquad (9)$$

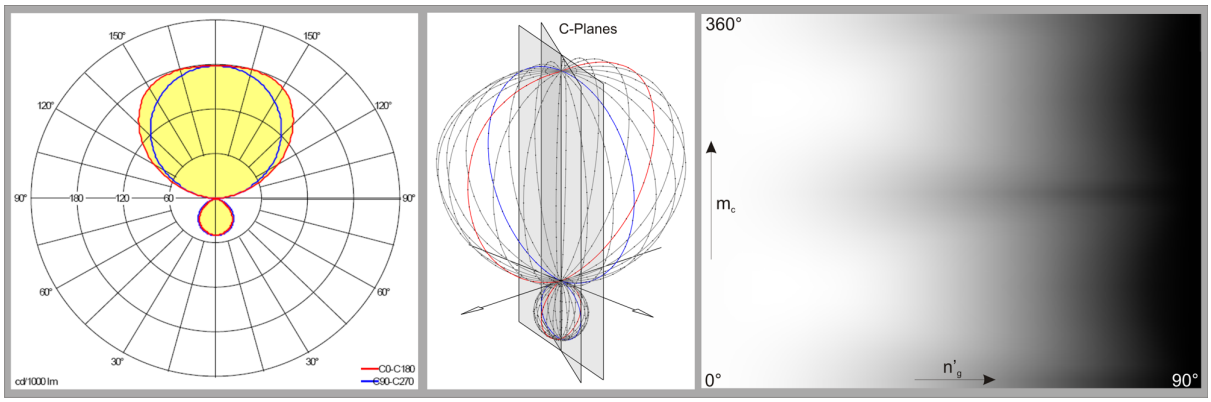

Figure 5: The light distribution of a Zumtobel KAREA-S luminaire (left) and its 3D representation with $m_c = 24$ (middle). The red line depicts the intersection of the light distribution with the plane $C0/180$ and the blue line with plane $C90/270$ respectively. The texture derived from the luminaire's light distribution is shown on the right.

Furthermore the luminous flux ϕ_v is defined as

$$\phi_v = K_m \int_{\lambda=380nm}^{780nm} \phi_{e_\lambda} V(\lambda) d\lambda \tag{10}$$

For a monochromatic lightsource we can reduce Equation 10 to

$$\phi_v = K_m V(\lambda) \phi_e \tag{11}$$

where $K_m = 683\ lmW^{-1}$ is the sensitivity of the eye at 555 nm and $V(\lambda) = 1$ for photopic vision (these values can be gained from the photopic vision curve $V(\lambda)$). For values of $V(\lambda)$ refer, e.g. to [Gal04]. Substituting into Equation 8 we get

$$I_e = \frac{\frac{1}{683} d\phi_v}{d\Delta\Omega} = \frac{\frac{1}{683} I_v d\Delta\Omega}{d\Delta\Omega} = \frac{1}{683} I_v \tag{12}$$

□

An LDT stores the light distribution of a luminaire of one half space and has dimension $n'_g \times m_c$ where n'_g is the number of intensities of a C-Plane in one half-space. The intensity values are retrieved from the light distribution and divided by the maximum intensity value I_{max} to normalize the values to the range $[0..1]$. These values are written into the texture, where each horizontal line represents the intensity values of a C-Plane. The relationships are shown in Figure 5. Texture sampling is set to linear to automatically interpolate between the discrete measurements. To assure continuity at the boundary of 0° and 360°, the texture wrap mode in v-direction is set to GL_REPEAT. By storing only the normalized intensity distribution the texture can be reused for luminaires with the same light distribution but different intensities.

To access the LDT, the azimuth ϕ_r and elevation ϕ_n of the vector $\mathbf{d}$ with respect to the reference system of the light source given by $(\mathbf{n_0}, \mathbf{r_0}, \mathbf{u_0})$ is determined. We use the subscript 0 to denote unit vectors. Figure 6 shows a geometrical representation of the problem. Normalizing the angles to the range $[0,1]$ yields

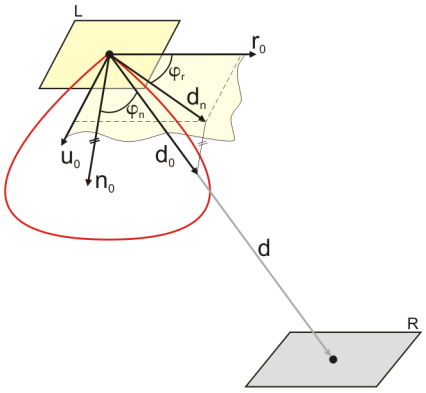

Figure 6: The texture coordinates of the LDT for a given direction vector $\mathbf{d}$ between the lightsource L and receiver R depend on the azimuth ϕ_r and elevation ϕ_n of this vector. The light distribution is depicted as red curve.

$$x_t = 1 + \frac{\min(\phi_n - \pi/2, 0)}{(\pi/2)} \tag{13}$$

$$y_t = \begin{cases} 1 - \frac{0.5\phi_r}{\pi} & u_0 \cdot \mathbf{d} \leq 0 \\ \frac{0.5\phi_r}{\pi} & \text{otherwise} \end{cases} \tag{14}$$

as texture coordinates (x_t, y_t). According to Sillion et al. [Sil91] the energy d^2E emitted by a differential area dA_i around a point T_i in the direction of unit vector $\mathbf{d_0}$ and falling on a differential area dA_j around a point T_j is then given by

$$d^2E = I(T_1, \mathbf{d_0}) \frac{cos(\phi_j) cos(\phi_i)}{\mathbf{d} \cdot \mathbf{d}} dA_j dA_i \tag{15}$$

where $I(T_1, \mathbf{d_0})$ is the intensity leaving the surface. In our case the intensity is retrieved by sampling the LDT at position (x_t, y_t) and multiplying it with I_{max}.

3.3 Normal Mapping

Inclusion of normal mapping [Coh98] into the radiosity process is straightforward. Instead of taking the interpolated vertex normals for calculation of ϕ_j the per-

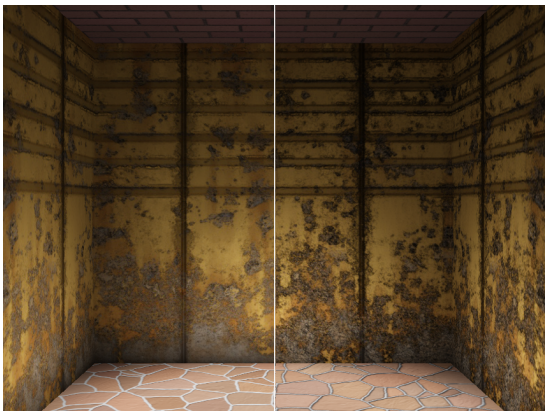

Figure 7: Radiosity solution of a simple box. Once without normal mapping (left) and one time with normal mapping enabled (right).

turbed normal is used. If the normals stored in the normal map are given in tangent space and since the light calculation is handled in world space, the vector **d** has to be transformed appropriately into tangent space. For static scenes the required tangent vectors can be calculated during the preprocessing step. Figure 7 and Figure 9 show results obtained with normal mapping.

However, it is worthy to note that if normal mapping and multitexturing is used simultaneously the resolution of the radiosity texture should correspond to the resolution of the normal map. Otherwise artifacts will appear because the result of light calculation does not overlay correctly with the texture of the object. Note also that illumination may change if normal mapping is used, because the shooting order must not necessarely be the same as without normal mapping.

4 ADAPTIVE SUBDIVISION

The accuracy of the radiosity solution depends very much on the underlying mesh. As noted by several authors (e.g. [Coh95]) uniform subdivision is not the best approach for radiosity, since some areas may be undersampled and others oversampled. Furthermore a too coarse mesh can introduce shadow leakage ([Bul89, Cam90]). Several techniques to identify elements that require subdivision have been proposed (see [Coh95] for an overview). For example, Vedel et al. [Ved91] subdivides if the gradient of the radiosity values varies more than a certain threshold. Campbell [Cam92] splits an element perpendicularly to the line connecting the maximum and minimum points of an element, if the difference between the extrema exceeds a certain threshold. Campbell et al. [Cam90] suggested a geometrical approach where the receiver polygon is tested against the shadow volume, generated by the light source and the occluding surfaces. However, the method is computationally expensive and does not scale well to complex scenes. We therefore propose the following method to determine if an element should be subdivided or not, by rendering the scene three times from the point of view of the shooter with a stereographic projection.

Step 1 Render the scene without depth testing and with occlusion queries enabled. This gives the complete number of rasterized fragments n_r for a triangle (independent from rendering order).

Step 2 Render the scene with depth testing enabled to initialize the depth buffer.

Step 3 Render the scene with depth testing and GL_LEQUAL as depth function and occlusion queries enabled. This yields the number of visible fragments n_v.

If $n_r \neq n_v$ there has to be a shadow boundary on this triangle. The triangle is subdivided if $b_l \leq n_v/n_r \leq b_u$ where $0 \leq b_l \leq 1$ and $0 \leq b_u \leq 1$ are the lower and upper threshold respectively. This avoids subdivision of triangles where the shadow boundary is short.

In our implementation, we account for subdivision before shooting the first time from a triangle. If the area of the shooter is small, we found that a sufficient tradeoff. In such a case, the rendering of the visibility texture can be combined with the steps outlined above. We follow the suggestion of Baum et al. [Bau91] and use regular refinement for subdivision of triangles. Newly introduced triangles are tested again for subdivision until a maximum subdivision level has been reached or the subdivision criteria is not fulfilled. To avoid linear interpolation artifacts due to introduced T-vertices, these vertices are fixed with bisection refinement in regard to the balance criterion of Baum et al. [Bau91]: the subdivision level of the neighboring elements should not differ more than one. If a triangle is subdivided, the radiosity and residual texture of the parent triangle is copied down to the child triangles using linear interpolation. Since subdivision is done before actually shooting, no reshooting as for example in [Coh88] is performed.

5 RESULTS

The presented method was implemented with C++, OpenGL and the Cg shading language from NVidia. Table 1 shows information about the examples used throughout this paper. It lists the used radiosity texture size TS along with the mipmap level used for shooting (in brackets), the time consumed by the radiosity solver including the post-process and simple exposure tone mapping t_r and the time for setup and preprocessing (loading of scene geometry, calculation of tangent vectors, initializing of the texture atlas etc.) t_{pp}. Furthermore, the number of triangles n_t and the number of elements n_e as well as the number of iterations IT (an iteration includes shooting from all elements of a triangle) are listed. The subscript nm denotes that normal mapping has been used. The time in brackets

Figure 8: The scene consists of 9012 triangles which are divided into 4259072 elements. The street lamp is simulated with a standard Lambertian light, the head and taillights are simulated with four spotlights. Each triangle was assigned a 32×32 radiosity texture and shooting was done from the third mipmap level.

Figure 9: A scene illuminated by a Zumtobel wallwasher. There are 18436 triangles in the scene yielding 8440590 elements. The left image shows the scene without normal mapping, the middle and right image where rendered with normal mapping. The right image is a close-up view of the statue showing the reflecting light from the wall.

Uniform	TS	t_{pp} [sec]	t_r [sec]	n_t	n_e	IT
box$_{nm}$	256(4)	0.96	28.37	42	1365280	16
box	32(3)	0.5	0.92	42	20120	16
bus	32(3)	8.84	58.8	8798	4192282	16
museum	32(3)	8.78	89.98	13627	6534502	10
statue	32(3)	13.45	104	16028	7683931	8

Adaptive	TS	t_{pp} [sec]	t_r [sec]	n_t	n_e	IT
bus	32(3)	8.84	72.26 (2.14)	9012	4259072	16
statue	32(3)	13.61	138.49 (14.43)	18288	8392967	8
statue$_{nm}$	32(3)	27.43	164.94 (33.0)	18436	8440590	8

Table 1: Performance for uniform and adaptive meshing for different scenes

represents the portion of t_r required for adaptive subdivision of the mesh. Except of the statue scene, all scenes have reached more than 88% convergence for the given number of iterations. All measurements where taken on a Intel Core2 CPU with 2.13 GHz with a Geforce 8800GTS with 640MB DDR3 Ram.

Performance analysis of the code showed that most time was consumed for rendering the receiver triangles. This is evident since this function is dependent on the result of the occlusion query to determine visibility. Additionally, setting the appropriate parameters of the orthographic projection for each triangle requires context switches of the fragment program. The performance of the current implementation is mainly limited by the available texture memory of the GPU. Once too many textures have to be maintained, texture memory thrashing can be noticed.

6 CONCLUSION AND FUTURE WORK

We presented a GPU implementation of progressive radiosity for triangular meshes. The rasterization of triangles is the major problem to overcome. We solve this by rendering the triangle itself and the outline of the triangle. The remaining artifacts are eliminated in a post-processing step or can be fixed during runtime with a fragment shader. Furthermore, we demonstrated the inclusion of normal mapping into the radiosity process, which yields more sophisticated results. Arbitrary light distributions can also be simulated with the help of light distribution textures.

The ample use of occlusion queries for determining visibility and shadow boundaries requires an elaborate algorithm to avoid stalling of the graphics pipeline. We are currently optimizing our implementation in this regard. Currently only one texture size is used for all triangles in the scene – independent from the actual size of a triangle. However, for scenes consisting of objects with rather coarse and fine meshes, this is subop-

timal, since some triangles inevitable get undersampled or oversampled respectively.

We aim to include general reflectance distributions by means of BRDFs, as published by Sillion et al. [Sil91], in our method. To allow for efficient reconstruction of the BRDF during runtime we are investigating the approach by NVidia [Wyn00]. To account for diffuse transmission the inclusion of a backward diffuse form factor [Rus90], which denotes the fraction of energy leaving a surface from its back side and impinging on another surface, is considered.

REFERENCES

[Ash] Ian Ashdown. English translation of the eulumdat specification. Available online: `http://www.helios32.com/Eulumdat.htm`.

[Bar04] Attila Barsi and Gabor Jakab. Stream processing in global illumination. *Proceedings of 8th Central European Seminar on Computer Graphics*, 2004.

[Bau91] D. R. Baum, S. Mann, K. P. Smith, and J. M. Winget. Making radiosity usable: automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. In *SIGGRAPH '91: Proc. of the 18th annual conference on Computer graphics and interactive techniques*, pp. 51–60. ACM Press, New York, NY, USA, 1991.

[Bul89] J. M. Bullis. *Models and Algorithms for Computing Realistic Images Containing Diffuse Reflections*. Master's thesis, Dept. of Computer Science, Univ. of Minnesota, 1989.

[Cam90] A. T. Campbell and Donald S. Fussell. Adaptive mesh generation for global diffuse illumination. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pp. 155–164. ACM Press, New York, NY, USA, 1990.

[Cam92] Alvin T. Campbell. *Modeling global diffuse illumination for image synthesis*. Ph.D. thesis, Austin, TX, USA, 1992.

[Car03] N. A. Carr, J. D. Hall, and J. C. Hart. Gpu algorithms for radiosity and subsurface scattering. In *HWWS '03: Proc. of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pp. 51–59. Eurographics Association, Aire-la-Ville, Switzerland, 2003.

[Coh85] Michael F. Cohen and Donald P. Greenberg. The hemicube: a radiosity solution for complex environments. In *SIGGRAPH '85: Proc. of the 12th annual conference on Computer graphics and interactive techniques*, pp. 31–40. ACM Press, New York, NY, USA, 1985.

[Coh86] M.F. Cohen, D.P. Greenberg, D.S. Immel, and P.J. Brock. An efficient radiosity approach for realistic image synthesis. *Computer Graphics and Applications*, vol. 6(3):pp. 26–35, 1986.

[Coh88] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pp. 75–84. ACM Press, New York, NY, USA, 1988.

[Coh95] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Morgan Kaufmann, 1995.

[Coh98] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 115–122. ACM Press, New York, NY, USA, 1998.

[Coo03] G. Coombe, M. Harris, and A. Lastra. Radiosity on graphics hardware, 2003.

[Gal04] Dietrich Gall. *Grundlagen der Lichttechnik - Kompendium*. Pflaum, 2004.

[Gau05] Pascal Gautron, Jaroslav Krivanek, Kadi Bouatouch, and Sumanta Pattanaik. Radiance cache splatting: A gpu-friendly global illumination algorithm. *Eurographics Symposium on Rendering*, 2005.

[Gor84] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84: Proc. of the 11th annual conference on Computer graphics and interactive techniques*, pp. 213–222. ACM Press, New York, NY, USA, 1984.

[Kel97] Alexander Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 49–56. ACM Press/Addison-Wesley, New York, NY, USA, 1997.

[Mar98] I. Martin, X. Pueyo, and D. Tost. A two-pass hardware-based method for hierarchical radiosity. *Computer Graphics Forum*, vol. 17(3):pp. 159–164, 1998.

[Nie02] Kasper H. Nielsen and Niels J. Christensen. Fast texture-based form factor calculations for radiosity using graphics hardware. *J. Graph. Tools*, vol. 6(4):pp. 1–12, 2002.

[Rei02] Erik Reinhard. Parameter estimation for photographic tone reproduction. *J. Graph. Tools*, vol. 7(1):pp. 45–52, 2002.

[Rus87] H. E. Rushmeier and K. E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. In *SIGGRAPH '87: Proc. of the 14th annual conference on Computer graphics and interactive techniques*, pp. 293–302. ACM Press, New York, NY, USA, 1987.

[Rus90] H. E. Rushmeier and K. E. Torrance. Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Trans. Graph.*, vol. 9(1):pp. 1–27, 1990.

[Seg03] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 2.0)*. 2003.

[Sil91] Françis X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A global illumination solution for general reflectance distributions. *SIGGRAPH Comput. Graph.*, vol. 25(4):pp. 187–196, 1991.

[Sil94] François X. Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann, 1994.

[Smi92] Brian E. Smits, James R. Arvo, and David H. Salesin. An importance-driven radiosity algorithm. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pp. 273–282. ACM Press, New York, NY, USA, 1992.

[Sun03] Chengyu Sun, Divyakant Agrawal, and Amr El Abbadi. Hardware acceleration for spatial selections and joins. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 455–466. ACM Press, New York, NY, USA, 2003.

[Ved91] C. Vedel and C. Puech. A testbed for adaptive subdivision in progressive radiosity. *2nd Eurographics Workshop on Rendering*, 1991.

[Wal89] J. R. Wallace, K. A. Elmquist, and E. A. Haines. A ray tracing algorithm for progressive radiosity. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pp. 315–324. ACM Press, New York, NY, USA, 1989.

[War88] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pp. 85–92. ACM Press, New York, NY, USA, 1988.

[Wyn00] Chris Wynn. Nvidia corporation. real-time brdf-based lighting using cube-maps. 2000.