Accelerating Radiosity on GPUs

Alexandr Shcherbakov
Lomonosov Moscow State University
GSP-1, Leninskie Gory
119991, Moscow, Russia
alex.shcherbakov@graphics.cs.msu.ru

Frolov Vladimir
Lomonosov Moscow State University
Keldysh Institute of Applied Mathematics
(Russian Academy of Sciences)
GSP-1, Leninskie Gory
119991, Moscow, Russia
vfrolov@graphics.cs.msu.ru

ABSTRACT

We propose a novel approach to implement radiosity on GPU with specific optimizations via form-factor matrix transformations. The proposed transformations enable to reduce the amount of computations for multiple-bounce global illumination and apply DXT compression (with subsequent hardware decompression when reading form-factors on GPU). Our implementation is 10 times faster running and requires 3 times less memory than the naive radiosity GPU implementation.

Keywords
Global illumination, radiosity, real-time applications.

1 INTRODUCTION

The main difficulty of real time global illumination involves accurate evaluation of reflected light. Precise value can be computed only through the lighting integral evaluation, which becomes much more complicated with each new reflection. Therefore, in practice different approximate methods are used. Today, there are a variety of popular techniques for global illumination evaluation. Each of them represents the evolution of some basic method of global illumination with several modifications, which make it more suitable for specific conditions and hardware.

2 INTERACTIVE GLOBAL ILLUMINATION METHODS

2.1 Instant Radiosity

Despite its name, instant radiosity [1] is not a variation of radiosity method [4]. The idea behind this method is to approximate indirect illumination with the direct light from a large number of "secondary" point light sources placed on the surface. The Reflective Shadow Maps (RSM) algorithm [3] is the most popular extension of instant radiosity method for GPU applications. RSM creates a set of secondary light sources from mip-mapped texture of shadow map [9]. These lights are used further in the fragment shader just like any other lights. The shadows from indirect light sources are not included in this case. The RSM method is the fastest and requires the least amount of memory over all existing methods of real-time global illumination solutions. However, its main disadvantage is poor accuracy.

2.2 Light Propagation Volumes

The main idea of Light Propagation Volumes (LPV) is to represent the lighting in a scene sampled on the lattice or grid. The algorithm consists of the four main steps:

1. Generate a radiance point set by rendering the scene into the reflective shadow map;
2. Inject virtual light sources into the radiance field;
3. Propagate radiance by iteratively solving a differential scheme inside the grid;
4. Apply a result radiance volume to the scene.

The disadvantage of LPV is $O(n^3)$, where n is the size of grid, complexity and memory consumption. The significant performance disadvantage appears in the propagation light through large empty spaces. Through, the Cascaded LPV [2] algorithm amortizes some of these problems, the accuracy of this method is not enough for many cases (especially for architecture-related applications).

2.3 Voxel Cone Tracing

Voxel Cone Tracing (VCT) [5] allows for voxelized approximation of the scene and trace cones via ray marching through the different mip-map levels of 3D texture. The key idea of VCT is to pre-integrate the incoming light via mip-mapping: gather the nearby light from the detailed mip level and the far light from the coarse mip level which averaged the emitted light from many surfaces. Therefore, distant areas are used with less precision.

The main disadvantage of VCT is a computational cost, because it traces several cones per pixel. The algorithm also suffers from light leaks due to coarse approximation of geometry by voxels.

2.4 Spherical Harmonics

The method of Spherical Harmonics [7] is a popular global illumination method based on the approximation of complicated functions by decomposition into a sum of simple spherical functions. For each vertex in scene polygons (or grid positions called light probes [8]) its own representation of lighting function is computed and approximated by spherical harmonics. The next stage of evaluating the global illumination is relatively cheap. Thus, the large part of computation is executed on a precomputing stage.

It generates acceptable global illumination. On the edge of light and shadow, it can create some artifacts caused by rough approximation of illumination functions.

2.5 Radiosity

Despite the considerable effort of researchers in real-time global illumination, an original radiosity method [4] has some advantages over the previously discussed methods. The first advantage is the conservation of energy and sufficiently high accuracy of the solution. The second one is low computational cost for a small number of patches, because the main evaluation runs on the precompute stage. However, it is difficult to use radiosity directly due to the fact that in modern 3D-scenes there are millions of polygons.

One of the modern versions of radiosity is the "Enlighten" [11] graphics engine in which simplified geometry is used for global illumination computing. Simplification is provided manually by 3D artists using some tools in 3D content modeling programs. Their implementation uses CPU for computing and updates indirect illumination once per 5-10 frames.

3 PROPOSED SOLUTION

3.1 GPU Radiosity

There were several works related to the implementation of radiosity on GPU. In [6] the progressive refinement radiosity algorithm running completely on GPU is presented. The work is mostly focused on form-factors computation with adaptive subdivision and using fixed functionality of GPU (like Rasterizer). In [13] an extended GPU progressive radiosity is presented. Their solver integrates ideal diffuse as well as specular transmittance and reflection and is capable to handle multiple specular reflections with correct mirror-object-mirror occlusions. In [14] another adaptive subdivision implementation on GPU is presented.

Unlike those discussed above, the work [10] is focused on the efficient linear system equation solution and matrix operations. It deals with the performance of different floating point format for matrix of form-factors and investigates the differences between GPU and CPU performance on matrix operations; it also explores a hierarchical radiosity approach via multiresolution meshed atlas. The authors of [10] pack 4 sequential form factors to a single color of texture. However, they didn't explore hardware compression and didn't propose algorithmic optimizations (except hierarchical radiosity). The work [10] also suggests sub surface scattering computation possibility via radiosity.

We believe that all the GPU radiosity works discussed above can be significantly improved. In this paper, we propose a new multibounce method based on special modifications of the radiosity algorithm [4] for global illumination computing. Our extensions are aimed to accelerate radiosity, reduce the required memory for form-factors storing and increase the accuracy of computation in comparison to popular global illumination methods.

3.2 Automatic Geometry Simplification

It is impractical to use radiosity for scenes that consist of millions of polygons, due to the high computational complexity of this problem. Therefore, in practice, radiosity is applied to a simplified scene and the produced result is used for original scene. We used the geometry simplification method based on the voxel representation of original scene [12].

3.3 Key Terms and Definitions

Global illumination computing is performed according to the following scheme. For each patch, initial luminance is computed or set. These values form vector *emission*. Each element of this vector is a three-component vector, one component per color.

$$emission_i = (red, green, blue) \tag{1}$$

Vector *excident* consists of colors that are sent from patches. It can be computed using *emission*.

$$excident_i = emission_i * color_i \tag{2}$$

$color_i$ is the color of i-th patch. Then, form-factors matrix F is multiplied by $excident$. The result of this operation is the lighting received by patches from light sources $incident$.

$$incident_i = \sum_{j=0}^{n} F_{ij} \cdot excident_j \tag{3}$$

n is the number of patches.

$$excident_i^{(1)} = incident_i^{(0)} \cdot color_i \tag{4}$$

$excident$ is the lighting sent from patches after first reflection.

Then we repeat multiplication of form-factors matrix and $excident$ vector for computing lighting after first reflection. We can repeat these operations for an arbitrary number of reflections.

3.4 Optimizing radiosity for multi-bounce global illumination

First, we define color form-factors matrix F^c. Element of this matrix on row i and column j are defined in the following way:

$$\begin{aligned} F_{ij}^c =& F_{ij} \cdot color_j = \\ &= (red_j \cdot F_{ij}, green_j \cdot F_{ij}, blue_j \cdot F_{ij}) \end{aligned} \tag{5}$$

Then, we can change the computing of patches lighting after the first reflection using equations (3), (4) and (5).

$$incident^{(1)} = F^c \cdot emission \tag{6}$$

Furthermore, we can generalize this equation for computing lighting received by patches after arbitrary reflection.

$$incident^{(h)} = F^c \cdot incident^{(h-1)} = (F^c)^h \cdot emission \tag{7}$$

Total lighting of the patch for k reflections is summarized from values of lighting received after each reflection.

$$\begin{aligned} incident_{total} &= \sum_{h=0}^{k} incident^{(h)} = \\ &= \sum_{h=0}^{k} \left((F^c)^h \right) \cdot emission) = \\ &= \left(\sum_{h=0}^{k} (F^c)^h \right) \cdot emission = \\ &= F^{k-reflection} \cdot emission \\ incident_{total} &= F^{k-reflection} \cdot emission \end{aligned} \tag{8}$$

So, we can use matrix $F^{k-reflection}$ to perform multiplication only once for k reflections. However, this matrix needs 3 times more memory than the original one.

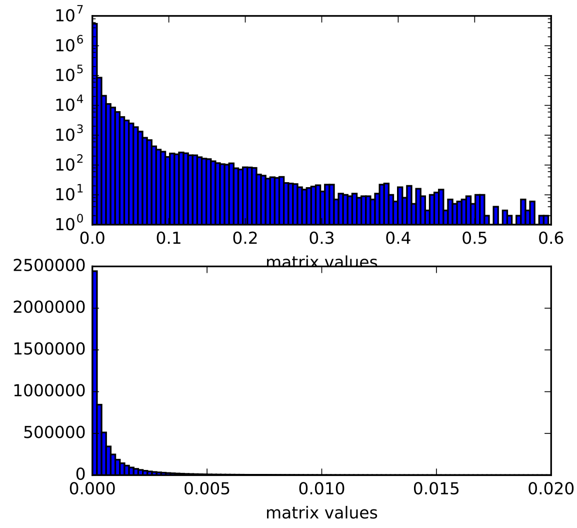

Figure 1: Distribution of form-factors values on the test scene (logarithm and linear scales).

3.5 Using DXT1 compression for form-factor matrix compression

For the form-factor matrix values less than 0.005 are prevailed (see Fig. 1). Since the contribution of these values in result is less than the others, they can be effectively compressed with losses.

For this reason, we split a form-factor matrix in two parts.

The first part contains 4% biggest numbers for each color channel for each row in the matrix. This value is based on the experimental results shown on figure 2. It provides a high quality of image.

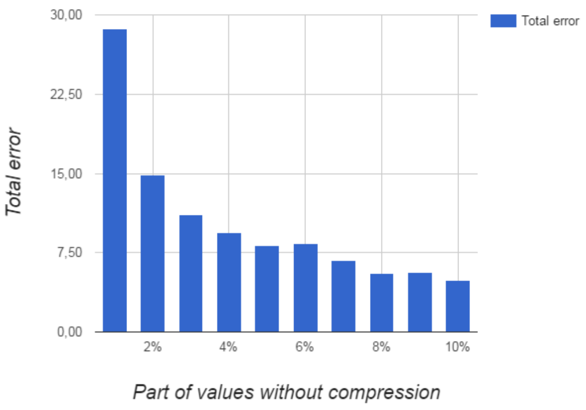

Figure 2: Total compression error for different parts of uncompressed values.

The second part has the same shape as the original matrix, but values from the first part are set to zero. Since these values fail to make significant contribution, accuracy for exponent is more important than accuracy for mantissa. Therefore, we apply some transformations to them.

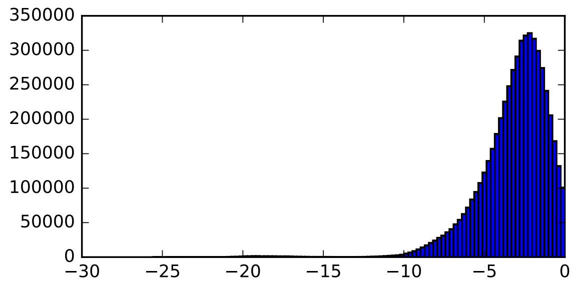

Figure 3: Distribution of lower form-factors values after normalization and logarithmization.

Firstly, values for each channel are divided on the maximum value for this channel. Secondly, the logarithm function is applied to the values (see Fig. 3).

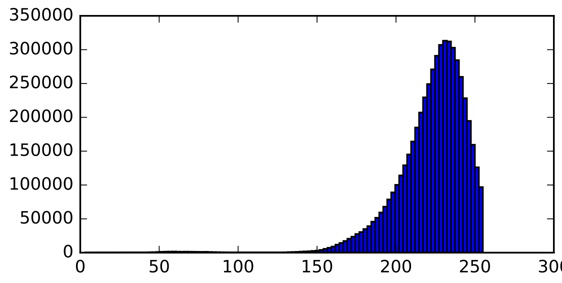

Figure 4: Result distribution of values in form-factors matrix.

Since the computed values are lie between -30 and 0 (see Fig. 3), we add a positive constant to them (in this case we add 25, because the values that remain negative are not important for computation). Then these values become to range from 0 to 255 (see Fig. 4).

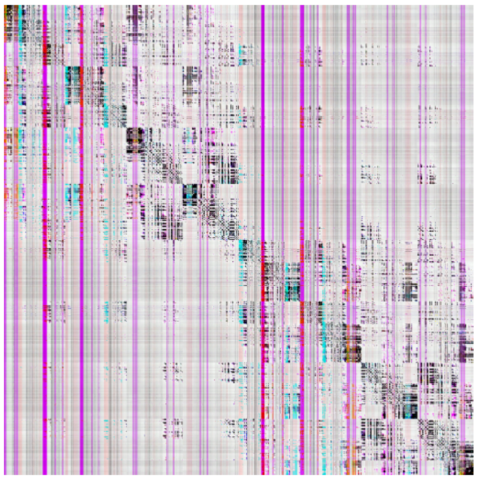

Figure 5: Form-factor matrix before reordering.

DTX1 compression is applied to a transformed form-factor matrix (see Fig. 5). To reduce the losses, we swap some rows of matrix. In this process, we also swap columns and patches matched these rows (see Fig. 6).

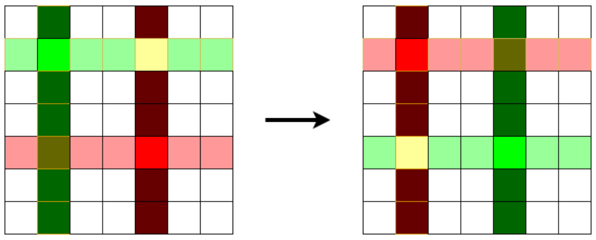

Figure 6: Scheme of columns/rows swapping.

Since we make matrix reorganization to reduce compress losses, we want to store similar values closer and hence we strive to place similar rows and columns next to each other. We use Euclidean distance as a measure of similarity.

In general case, if we represent columns or rows of matrix as vertices of full graph and set the measure of similarity as edges weights, then the problem of finding of the optimal order of rows and columns is the problem of finding of the shortest path in the graph, which includes all vertices. This problem is assigned to NP class.

Since the number of patches is the number of the order of several thousand, we use a heuristic approach. The first row always stays in the place. Most similar to the first row is put in the second place. Most similar to the second of the remaining row is put in the third place, etc. These matrix transformations decrease compression losses about 5 times (see Fig. 7).

	Without reordering	Reordering by rows	Reordering by rows & columns
Avg. error	1,28E-06	8,13E-07	5,88E-07
Max error	0,4622	0,4684	0,1524
Total error	47,12	18,86	9,40

Figure 7: Comparison of error for reordering.

Then, we apply second reordering, but measure is computed for columns.

After all reorderings, resulting texture is saved using DXT1 compression. Matrix after compression is presented in the figure 8. It is difficult to tell the difference, so the difference for the red component of the texture is shown in the figure 9. For green and blue channels, difference is similar in structure.

3.6 Implementation Details

All computations are executed on GPU using OpenGL framework. Form-factors matrix stored as a compressed texture. Other data stored in array buffers. The order of computation is the following:

1. Rendering of shadow maps for all light sources;
2. Running compute shader. This shader uses shadow maps for computing emission on the patches. Emission is square of illuminated part of patch;

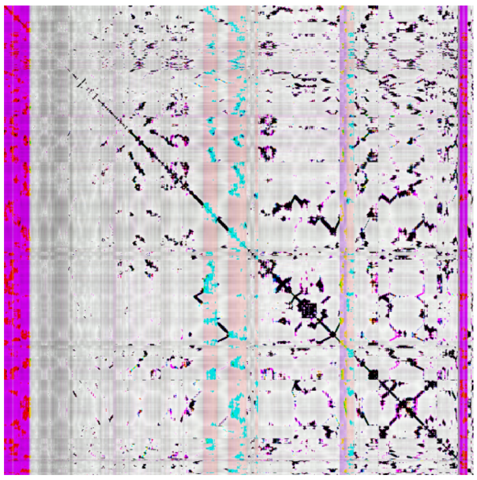

Figure 8: Compressed form-factor matrix after two reordering.

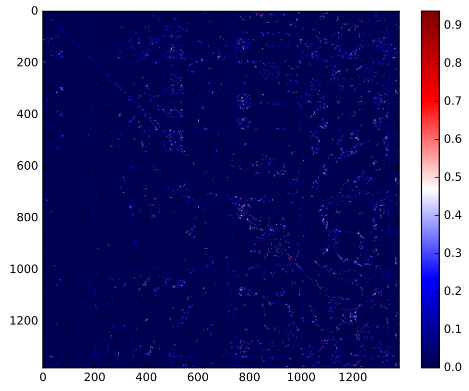

Figure 9: Form-factor matrices difference in red component.

3. Next, we run another compute shader for radiosity. On this step compressed form-factor matrix is used. The form-factor matrix is multiplying by emission vector;

4. And the last compute shader transfers indirect illumination from the simplified scene to the original;

5. The final step is the scene rendering.

4 EXPERIMENTAL RESULTS

In this paper, Naive Radiosity and Path Tracing are compared with our approach.

4.1 Comparison with naive radiosity implementation

4.1.1 *Computation speed comparison*

Our multibounce radiosity implementation is asymptotically faster than both naive radiosity and the algorithm described in [10]. Presented method can be used with other optimization techniques for radiosity (for example, hierarchical radiosity).

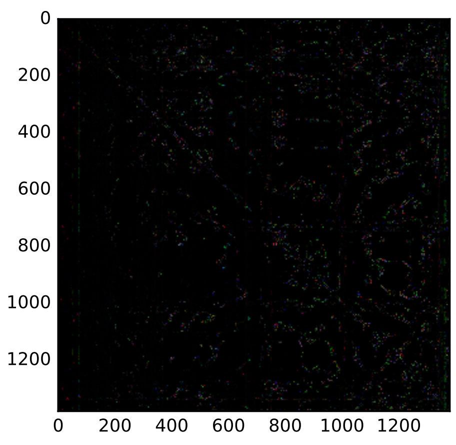

Figure 10: Form-factor matrices difference in all components.

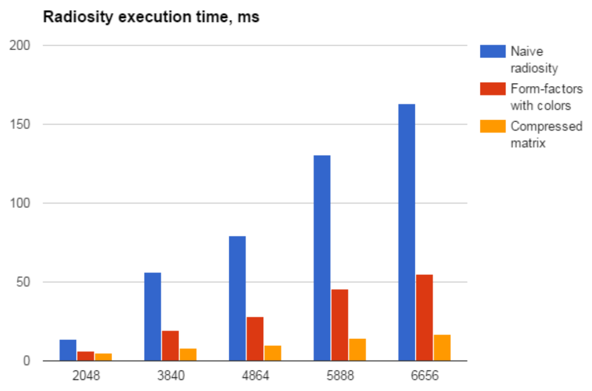

Figure 11: Speed comparison.

It can be seen in the figure 11, our method significantly superiors the naive implementation of radiosity. In addition, using the compressed by DXT1 form-factor matrix is faster due to decreased memory amount that we need to transfer between DRAM and GPU multiprocessor.

4.1.2 *Memory Requirements Comparison*

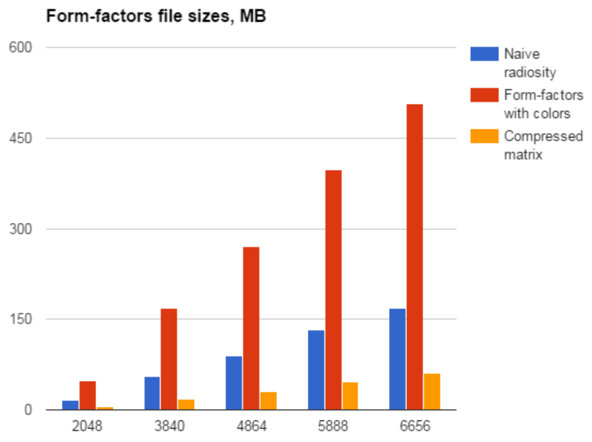

Figure 12: Memory comparison.

In figure 12 we show the sizes of files containing form-factor matrices. After compression, matrix requires significantly less memory. Thus, our modifications reduce the amount of data being stored and loaded into video memory.

4.1.3 Images Comparison

Our approach generates an image with similar quality as naive radiosity implementation, but we use less amount of memory and execute the radiosity algorithm faster.

4.2 Comparison with Light Propagation Volumes

Light Propagation Volumes generates image with visually perceptible inaccuracy as can be seen on figure 13. A column in selection (green rectangle) is pink, but on reference image this column painted with a gradient. Both images (our approach and LPV) rendered with 40 FPS. The size of voxel grid for LPV was chosen to reach the same fps with our implementation of radiosity.

4.3 Comparison with Path Tracing

In comparison with Path Tracing (see Fig. 13) we show that our method generates a similar image. Meanwhile, our image was generated less than 17ms. Path Tracing needs more than 5 minutes to generate a reference image on the same GTX670.

5 CONCLUSION

In this paper, we present a practical approach for real-time global illumination on GPU using radiosity. First, we reduced the multiple bounce computation cost by introducing color information to a form-factor matrix and powering it. Second, we apply rows/columns reordering and DXT compression to both save memory and increase speed when reading form-factors from DRAM on GPU (because radiosity is memory-bound). Our implementation runs 10 times faster than naive radiosity and requires 3 times less memory. The resulting image hardly differs from the path-traced reference and has comparable FPS to other real-time global illumination algorithms.

6 ACKNOWLEDGMENTS

This work is supported by RFBR 16-31-60048 mol_a_dk.

7 REFERENCES

[1] Alexander Keller. 1997. Instant radiosity. In Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 49-56. DOI=http://dx.doi.org/10.1145/258734.258769

[2] Anton Kaplanyan and Carsten Dachsbacher. 2010. Cascaded light propagation volumes for real-time indirect illumination. In Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games (I3D '10). ACM, New York, NY, USA, 99-107. DOI=http://dx.doi.org/10.1145/1730804.1730821

[3] Carsten Dachsbacher and Marc Stamminger. 2005. Reflective shadow maps. In Proceedings of the 2005 symposium on Interactive 3D graphics and games (I3D '05). ACM, New York, NY, USA, 203-231. DOI=http://dx.doi.org/10.1145/1053427.1053460

[4] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. 1984. Modeling the interaction of light between diffuse surfaces. In Proceedings of the 11th annual conference on Computer graphics and interactive techniques (SIGGRAPH '84), Hank Christiansen (Ed.). ACM, New York, NY, USA, 213-222. DOI=http://dx.doi.org/10.1145/800031.808601

[5] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. 2011. Interactive indirect illumination using voxel-based cone tracing: an insight. In ACM SIGGRAPH 2011 Talks (SIGGRAPH '11). ACM, New York, NY, USA, , Article 20 , 1 pages. DOI=http://dx.doi.org/10.1145/2037826.2037853

[6] Greg Coombe, Mark J. Harris, and Anselmo Lastra. 2004. Radiosity on graphics hardware. In Proceedings of Graphics Interface 2004 (GI '04). Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 161-168.

[7] Ian G. Lisle and S.-L. Tracy Huang. 2007. Algorithms for spherical harmonic lighting. In Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia (GRAPHITE '07). ACM, New York, NY, USA, 235-238. DOI=http://dx.doi.org/10.1145/1321261.1321303

[8] Jaroslav Krivanek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. 2008. Radiance caching for efficient global illumination computation. In ACM SIGGRAPH 2008 classes (SIGGRAPH '08). ACM, New York, NY, USA, , Article 75 , 19 pages. DOI: https://doi.org/10.1145/1401132.1401228

[9] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. 2004. Light space perspective shadow maps. In Proceedings of the Fifteenth Eurographics conference on Rendering Techniques (EGSR'04). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 143-151.

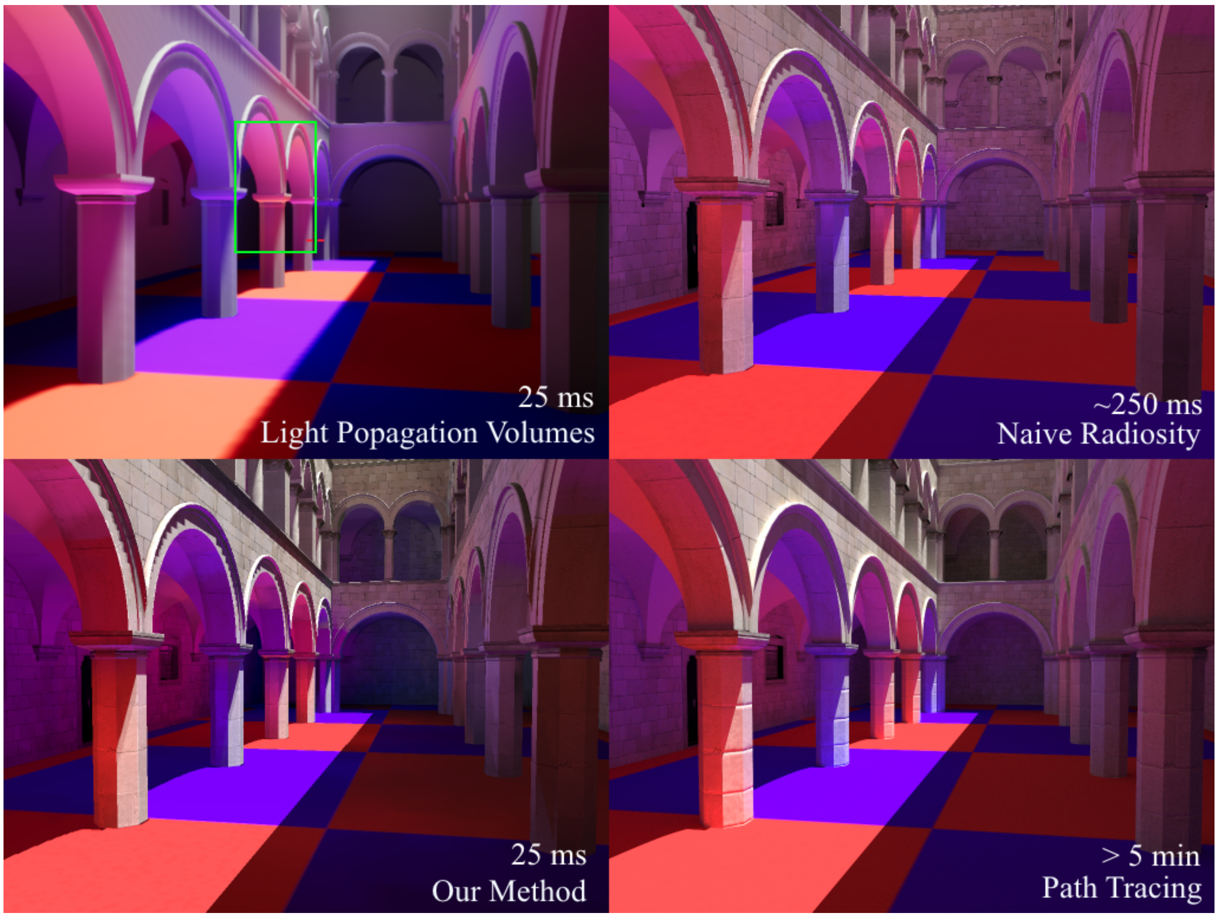

Figure 13: Our method (left-bottom), Light Propagation Volumes (Unreal Engine 4) (left-top), naive radiosity (right-top) and Path Tracing (right-bottom).

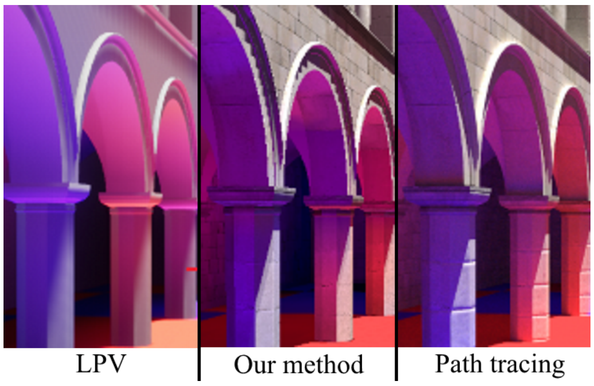

Figure 14: Comparison with Light Propagation Volumes and Path Tracing

DOI=http://dx.doi.org/10.2312/EGWR/EGSR04/143-151

[10] Nathan A. Carr, Jesse D. Hall, and John C. Hart. 2003. GPU algorithms for radiosity and subsurface scattering. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (HWWS '03). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 51-59.

[11] SamMartin, Per Einarsson. A Real Time Radiosity Architecture for Video Games. Siggraph 2010. http://advances.realtimerendering.com/s2010/Martin-Einarsson-RadiosityArchitecture(SIGGRAPH%202010%20Advanced%20RealTime%20Rendering%20Course).pdf

[12] Shcherbakov, A., and Frolov, V. Automatic geometry simplification for computation of indirect lighting using radiosity. In Graphicon-2016 (2016), NNGASU, pp. 34-38

[13] Wallner, G. Vis Comput (2009) 25: 529. doi:10.1007/s00371-009-0347-z

[14] Wallner, G. GPU radiosity for triangular meshes with support of normal mapping and arbitrary light distributions. J. WSCG 16(1-3), 1-8 (2008)