VOLUME-BASED DIFFUSE GLOBAL ILLUMINATION

Pavlos Mavridis
Athens University of Economics and Business, Dept. of Informatics, Greece

Athanasios Gaitatzes
University of Cyprus

Georgios Papaioannou
Athens University of Economics and Business, Dept. of Informatics, Greece

ABSTRACT

In this paper we present a novel real-time algorithm to compute the global illumination of scenes with dynamic geometry and arbitrarily complex dynamic illumination. We use a virtual point light (VPL) illumination model on the volume representation of the scene. Light is propagated in void space using an iterative diffusion approach. Unlike other dynamic VPL-based real-time approaches, our method handles occlusion (shadowing and masking) caused by the interference of geometry and is able to estimate diffuse inter-reflections from multiple light bounces.

KEYWORDS

Real-Time Global Illumination, Spherical Harmonics, Voxels.

1. INTRODUCTION

In order to synthesize photorealistic images we need to capture the complex interactions of light with the environment. Light follows many different paths distributing energy among the object surfaces. This interplay between light and object surfaces can be classified as *local illumination* and *global illumination*. *Local illumination* algorithms take into account only the light which arrives at an object directly from a light source. *Global Illumination* algorithms, on the other hand, take into account the entire scene, where the light rays can bounce off the different objects in the environment or be obstructed and absorbed. Reflections, refractions, occlusions and shadows are examples of complex light interactions with a high computational cost that is usually not available for real-time applications.

In this paper we propose a method that produces photorealistic images of diffuse, dynamic environments in real time, by estimating the illumination at discrete locations in the environment and applying the results on the scene geometry. This way, we can achieve shadowing effects as well as diffuse inter-reflections from the light bounces. The method we propose uses a discretization approach of the scene objects, incorporating geometry information in the discretization structure. Instead of using the shadow map data as virtual point lights (VPLs) [Dachsbacher et al. 2005] [Dachsbacher et al. 2006] [Kaplanyan 2009], our method performs a complete scene voxelization and is thus able to include occlusion information along with any direct, indirect and self-emitted illumination. Furthermore, it is capable of calculating global illumination from multiple light bounces.

2. PREVIOUS WORK

Radiosity based methods in voxel space have addressed the illumination problem, like [Chatelier et al. 2005] but their results were not computed in real-time and had large storage requirements. Modern advances of the same approach, [Kaplanyan 2009], yielded much faster results than before, but ignored indirect occlusion and secondary light bounces.

The Irradiance Volume, which was first introduced in [Greger et al. 1997], regards a set of single irradiance samples, parameterized by direction, storing incoming light for a particular point in space (i.e. the light that flowed through that point). The method had large storage requirements as neither an environment map nor spherical harmonics were used for the irradiance storage. With a set of samples they approximated the irradiance of a volume, which was generally time-consuming to compute but trivial to access afterwards. With an irradiance volume they efficiently estimated the global illumination of a scene.

Instant radiosity methods, introduced by Keller [Keller 1997], approximate the indirect illumination of a scene using a set of VPLs. A number of photons are traced into the scene and VPLs are created at surface hit points, then the scene is rendered, as lit by each VPL. The major cost of this method is the calculation of shadows from a potentially large number of point lights but since it does not require any complex data structures it is a very good candidate for a GPU implementation. Lightcuts [Walter et al. 2005] reduce the number of the required shadow queries by clustering the VPLs in groups and using one shadow query per cluster, but the performance is still far from real time.

Reflective shadow maps [Dachsbacher et al. 2005] consider the pixels of a shadow map as VPLs, but the contribution of these lights is gathered without taking scene occlusion into account. To achieve interactive frame rates, screen space interpolation is required and the method is limited to the first bounce of indirect illumination. An extension of this method by the same authors [Dachsbacher et al. 2006] reorganizes the computation of the indirect light to achieve better performance, but it still ignores occlusion for the indirect lighting. Imperfect shadow maps [Ritschel et al. 2008] use a point based representation of the scene to efficiently render extremely rough approximations of the shadow maps for all the VPLs in one pass. They achieve interactive frame rates but indirect shadows are smoothed out considerably by the imperfections and the low resolution of the shadow maps.

[Jensen 1996] introduced the concept of photon mapping, where in a first pass photons are traced from the light sources into the scene and stored in a k-d tree and in a second pass the indirect illumination of visible surface points is approximated by gathering the k nearest photons. [McGuire et al. 2009] computes the first bounce of the photons using rasterization on the GPU, continues the photon tracing on the CPU for the rest of the bounces and finally scatters the illumination from the photons using the GPU. Since part of the photon tracing still runs on the CPU, a large number of parallel cores are required to achieve interactive frame-rates.

[Ritschel et al. 2009] extends previous methods for screen space ambient occlusion calculation [Shanmugam et al. 07] and introduces a method to approximate the first indirect diffuse bounce of the light by only using information in the 2D frame buffer. This method has a very low computational cost but the resulting illumination is hardly accurate since it depends on the projection of the (visible only) objects on the screen.

The concept of interpolating indirect illumination from a cache was introduced by [Ward et al. 1988]. Accurate irradiance estimates are computed using ray tracing on a few surface points (irradiance sample points) and for the remaining surface points fast interpolation is used. Wang [Wang et al. 09] presents a method to calculate the irradiance sample points in advance and implements the algorithm on the GPU. The method is accurate but it achieves interactive frame rates only in very simple scenes.

[Nijasure et al. 05] uses spherical harmonics to store the incoming radiance of the scene in a uniform grid structure. The surfaces are rendered by interpolating the radiance from the closest grid points. This method supports multiple bounces and indirect occlusion but it's very expensive because it requires the complete scene to be rendered in a cube map for the radiance estimation on each grid point.

[Kaplanyan 2009] also uses a regular grid to store the scene radiance, but for its calculation he uses a propagation scheme to calculate the radiance distribution on the scene. Radiance is initially injected in VPL positions and then it is iteratively propagated through empty space. The method achieves very high performance but completely ignores occlusion for the indirect light and secondary bounces.

3. METHOD OVERVIEW

Our method consists of three stages, first the scene is discretized to a voxel representation. Next the radiance of each voxel is iteratively propagated in the scene and finally, during image rendering, the irradiance of each surface point is calculated taking into account the radiance from the closest voxels.

3.1 Real-Time Voxelization

Our algorithm operates on a uniform grid data structure. A uniform data structure is preferable over an hierarchical one, like sparce voxel octrees, because it is more suitable for fast generation by the GPU. This is essential to our method since we consider our scenes fully dynamic and the complete data structure must be rebuilt in every frame.

Each grid cell contains information about space occupation and the average normal of the surfaces inside the cell. For each color band, it also stores the propagated radiance distribution inside the cell, which is the radiance that is going to be propagated in the next iteration step. The cumulative sum of the incoming radiance over all the iteration steps is also stored for each color band, and is going to be visualized during the final scene rendering, as detailed in section 3.3. The propagated radiance (propagation buffer) is initialized with the first bounce VPLs (direct illumination), as described in the following paragraph and the cumulative radiance (accumulation buffer) is initialized with zero. All the radiance distributions are approximated using a compact spherical harmonic representation.

To voxelize the scene, instead of applying one of the fast binary GPU voxelization methods, such as [Eisemann et al. 08], we use a variant of Chen's algorithm [Chen et al. 98] because we need to store multi-channel scalar data in each voxel. In brief, the scene is uniformly sliced, and each slice is scan converted. Each pixel of a slice corresponds to a voxel in the final volume. Chen assumed that all the objects are watertight and he used the XOR operator to keep track of the void/non-void regions. This assumption often does not hold for typical scenes encountered in real-time graphics applications and also Chen's method cannot store any information about surfaces that are perpendicular to the slicing buffers, since they are never scan-converted. We solve this problem by repeating the scan-conversion process 3 times, using 3 perpendicular slicing directions, once for each primary axis. This way, we ensure that the depth-discontinuity in one orientation of the view plane will be remedied in one of the two others. Three intermediate buffers on the GPU are needed to store the temporary volume results of the three slicing procedures (one for each different orientation of the object). Finally, those three volumes are combined into one buffer, using the MAX frame buffer blending operation.

During the scan conversion of the slices, the average direct illumination of each grid cell is computed, complete with shadows and emissive illumination. Then a new hemispherical VPL is created towards the direction of the average normal with the same intensity as the computed illumination. The spherical harmonic representation of this VPL gives the initial radiance distribution in the propagation buffer for the iterative radiance propagation.

3.2 Iterative Radiance Propagation

Once we have initialized the grid cells with the radiance distribution of the first bounce VPLs, we need to propagate this radiance to their neighboring voxels. Similar to Kaplanyan [Kaplanyan 2009], we use an iterative diffusion approach on the GPU to propagate energy within space. In contrast to [Kaplanyan 2009] though, since each grid cell contains space occupation information from the voxelization (not-just VPLs), energy is propagated only in void space, from one voxel boundary to the next. The propagated radiance is *reflected* on occupied (voxelized) volume grid points and accumulated at these locations in the accumulation buffer, as detailed below.

Each grid cell contains a radiance distribution $L(x,\omega)$ *that* is going to be propagated in neighboring cells. We assume that all the radiance inside a cell is originating from the center of the cell, thus it can be seen as a function $L(\omega)$ over the sphere and can be encoded using spherical harmonic coefficients. Given a cell i with radiance distribution L_i, we must compute how much radiance L_j a neighboring cell j receives from i. We observe that the radiance between the two cells is going to be propagated through the boundary that connects them, and it's easy to see that

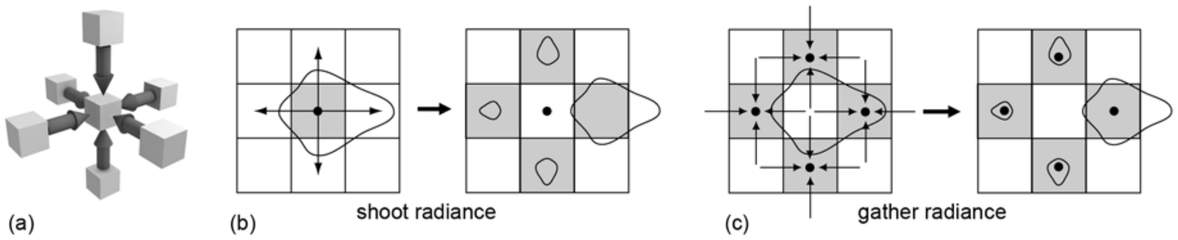


Figure 1. a) Radiance Gathering Illustration. The radiance for the center voxel is gathered from the values stored at the voxels of the surrounding cells. b) and c) Radiance propagation. Radiance shooting is equivalent to radiance gathering

$$\mathrm{L}_{\mathrm{j}}(\omega) = \mathrm{L}_{\mathrm{i}}(\omega) \cdot \mathrm{T}_{\mathrm{j}}(\omega)\,,\ \textit{where}\ \ \mathrm{T}_{\mathrm{j}}(\omega) = \begin{cases} 1, \ \omega \in \Omega_{\mathrm{j}} \\ 0, \ \omega \notin \Omega_{\mathrm{j}} \end{cases} \tag{3.2.1}$$

where we define Ω_j as the solid angle subtended by the boundary that connects the two voxels. T_{j} is represented using spherical harmonics and is essentially a filter that keeps only the radiance that has a direction towards the neighboring cell *j*. Computing the product of two spherical harmonics is too expensive for real-time applications, so in the above equation we replace radiance distribution $L_i(\omega)$ with a constant term, the average radiance $L_{avg}(\omega)$ over the solid angle Ω_{j} which can be computed by the following integral

$$\mathrm{L}_{\mathrm{avg}}(\omega) = \int_{\Omega'} \mathrm{L}_{\mathrm{i}}(\omega) \cdot \mathrm{T}_{\mathrm{j}}(\omega) \mathrm{d}\omega \tag{3.2.2}$$

where the integration domain Ω' is the full sphere. According to the spherical harmonic properties, the above integral can be computed as a simple dot product of the spherical harmonics representation of the two terms, and by replacing $L_i(\omega)$ with $L_{avg}(\omega)$ in equation (3.2.1) we get the final equation for our propagation scheme

$$\mathrm{L}_{\mathrm{j}}(\omega) = \mathrm{T}_{\mathrm{j}}(\omega) \cdot \int_{\Omega'} \mathrm{L}_{\mathrm{i}}(\omega) \mathrm{T}_{\mathrm{j}}(\omega) \mathrm{d}\omega \tag{3.2.3}$$

For each grid cell *i* the above equation can be applied to each of the six neighboring cells *j*, to find the radiance that is scattered from *i* to its neighbors. This corresponds to a shooting/scattering propagation scheme and for each iteration one cell is read and six are written. Alternatively we can change the order of the computations and for each grid cell *i* we can calculate the amount of radiance that it receives from the six neighboring cells *j*. This corresponds to a gathering scheme, as shown in figure 1, and for each iteration, it requires six reads and one write, so it's much more preferable for a parallel implementation on the GPU. Of course, the scattering and the gathering schemes are computationally equivalent, only the order of the operations is changing.

When radiance is propagated towards a cell that is marked as occupied (according to the voxel data), then the incoming radiance of this cell is accumulated in a separate accumulation buffer. This buffer stores the cumulative incident radiance distribution of each cell and is going to be used in the final rendering step, as will be detailed in section 3.3. Also occupied voxels should diffusely reflect any incoming radiance, so in this case a new hemispherical VPL is created towards the direction of the average normal in this voxel and its radiance distribution is injected in the propagation buffer of this cell. The algorithm runs iteratively, so this new radiance is going to contribute in the next propagation step.

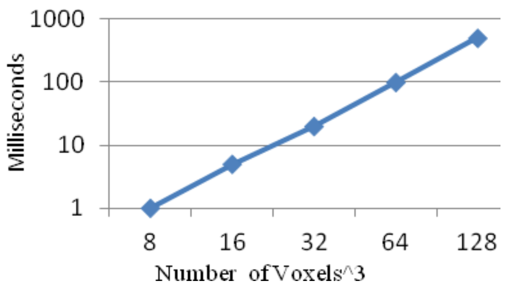
3.3 Final Irradiance Reconstruction

During the final scene rendering, the irradiance at each surface point must be computed from the radiance *L* that is stored in our uniform grid structure. The irradiance *E* of a point **x** is given by the following equation [Kajiya 1986]:

$$E(\mathbf{x}) = \frac{\rho(\mathrm{x})}{\pi} \int_{\Omega} L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta \, d\vec{\omega}_i \tag{3.3.1}$$

	T	I	V (ms)	P(ms)	T(ms)
boxes	48	11	10	12	22
room	704	64	3	61	69
sponza	66454	11	10	11	28
arena	10219	12	3	13	21

a)



b)

Figure 2. a) Measurements of our test scenes with a grid size of 32^3. T:triangles, I:iterations, V:voxelization, P:propagation, T:total time. b) Scalability with grid resolution (logarithmic scale)

where θ is the angle between the surface normal and the incident radiance direction ω_i. The integration domain is the hemisphere Ω defined by the surface normal $\mathbf{n}_x$ at point $\mathbf{x}$. If we change the integration domain to the full sphere Ω', the previous equation can be rewritten as follows:

$$E(\mathbf{x}) = \frac{\rho(\mathrm{x})}{\pi} \int_{\Omega'} L_i(\mathbf{x}, \vec{\omega}_i)\, T(\mathbf{n}_\mathrm{x}, \vec{\omega}_i)\, d\vec{\omega}_i \text{ , } \textit{where} \quad T(\mathbf{n}_\mathrm{x}, \vec{\omega}_i) = \begin{cases} \cos\theta, & \theta < \pi/2 \\ 0, & \theta > \pi/2 \end{cases} \tag{3.3.2}$$

This change of the integration domain is necessary because we are going to use spherical harmonics, which are defined over the sphere and not the hemisphere.

Equation (3.3.2) is directly evaluated per pixel to give the final indirect lighting. In our algorithm the radiance L is tri-linearly interpolated from the stored values in the uniform grid structure. From the eight closest grid points only the ones corresponding to occupied voxels are considered for interpolation. L is already stored and interpolated in spherical harmonic representation. We also build a spherical harmonic representation for the function T, as described in [Sloan 2008] and the integral is calculated per pixel as a simple dot product, according to the spherical harmonic properties.

4. RESULTS

We have integrated the above algorithm in a real time deferred renderer using OpenGL and GLSL. Our proof of concept implementation uses a 2nd order spherical harmonic representation, since the four SH coefficients, map very well to the four component buffers supported by the graphics hardware. All figures presented here are rendered on an nVIDIA GeForce GTX285 at 900x900 pixels with a 32^3 grid size, unless otherwise noted. It should be noted here that, excluding the final interpolation stage, the performance of the indirect lighting computation in our method does not depend on the final screen resolution, but only on the voxel grid size and the number of propagation steps. This is a big advantage over instant radiosity methods, like imperfect shadow maps.

Figure 2a shows the comprehensive time measurements for all the scenes detailed below. Please note that only the voxelization and propagation times are relevant to our work. The total rendering time includes the direct lighting computation and other effects and is given as a reference. All scenes are considered to have fully dynamic geometry and lighting conditions. In all cases our algorithm achieves real time frame rates and sufficient accuracy in the reproduction of the indirect diffuse illumination, even though our implementation is not optimized in any way. Figure 2b shows that our method scales linearly with the total number of voxels. In these tests the propagation distance is fixed and equals the maximum side of the scene (when the grid size is doubled the iteration steps are also doubled).

We have found that the propagation stage of our method is limited by the available memory bandwidth and not the computational speed. This is to be expected, since the propagation kernel requires 52 floating point reads and 8 floating point writes per color band. To save memory bandwidth we don't store the diffuse color of the surfaces in the voxel structure, but after the first light bounce we consider it constant and equal to 0.5 for each color band.

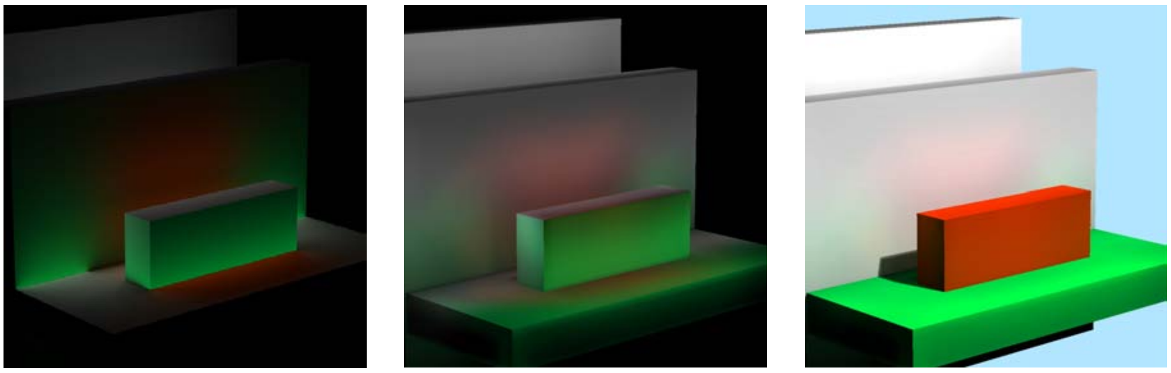


Figure 3. From left to right: reference solution computed with ray tracing (indirect illumination only), our solution (indirect illumination only) and final image with direct and indirect lighting

Figure 3 shows a direct comparison of our algorithm with a reference solution on a simple test scene. We can see that our method reproduces the shape and the properties of the indirect illumination in the reference solution.

Figure 4 shows a room lit through a set of open windows. This is a challenging scene for global illumination algorithms, because only a small region on the left wall is directly lit by the sun and the majority of the lighting in the room is indirect. We can see that the simple light propagation method [Kaplanyan 2009] completely fails to reproduce the indirect lighting in the room, since it is not taking into account secondary light bounces and occlusion. At least two bounces of indirect lighting are required to get meaningful results in this case. In our method, when a grid of size Y is used, the distance between the walls of the room is also Y, so n*Y propagation steps are required to compute n bounces of indirect illumination. This is a worst case scenario, as in typical scenes light interaction from one end of the scene to the other is not required. In this particular case we have used 64 propagation steps to simulate two bounces of light on a 32^3 grid. The resulting illumination is visually pleasing, giving high contrast on the edge of the walls and the staircase. Since our algorithm takes indirect occlusion in consideration, the area below the staircase is correctly shadowed. We observe some artifacts below the windows, due to the imprecision of the spherical harmonics and the fact that the grid cell on this area covers both the edge of the wall and the empty space inside the window.

Even with a number of propagation steps this high, our method maintains easily an interactive frame-rate since the propagation stage takes only *61 ms* to complete. A nice characteristic of our method is the predictable performance of the propagation stage. We can easily calculate that one propagation step for each individual voxel takes $2.8*10^{-5}$ *ms* and this time is constant and independent from the scene complexity. It should be noted of course that the performance may be constant and predictable, but the same is not true for the accuracy and the quality of the resulting illumination.

Figure 5 shows the sponza atrium, a typical scene in the global illumination literature. The right part of the scene is directly lit by the sun, the left one is lit only indirectly. As we can see, using only eleven

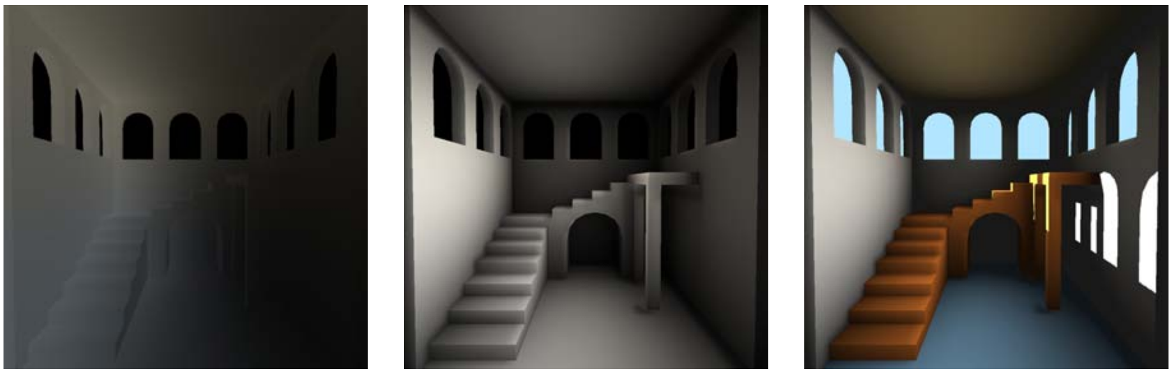


Figure 4. From left to right: the indirect illumination using light propagation volumes [Kaplanyan 2009], Indirect lighting using our method with 64 iterations and final image with direct and indirect lighting

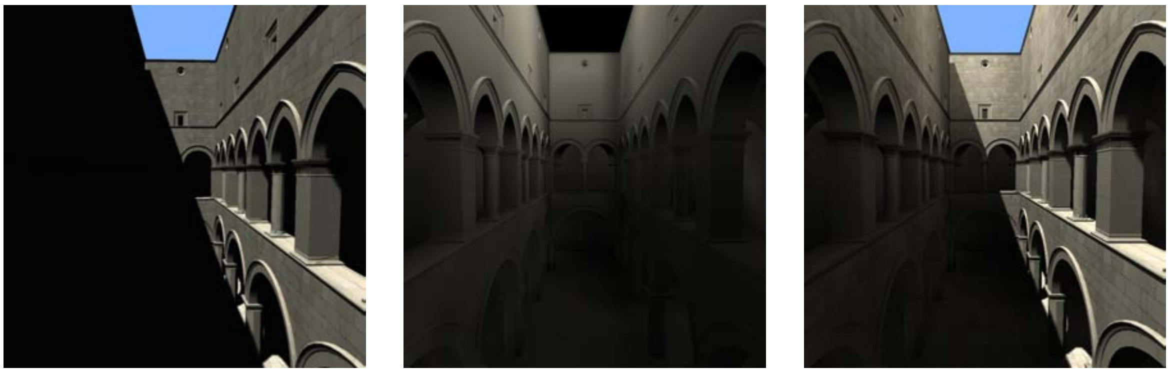


Figure 5. From left to right: the sponza atrium scene lit with direct lighting, indirect lighting only and the final image with direct and indirect lighting

propagation steps our method successfully reproduces the low-frequency indirect illumination which is dominant on the left part of the scene with very few visible artifacts.

Figure 6 shows an arena, a typical outdoor scene in video games. Twelve propagation steps are used in this case and we can see that the resulting indirect illumination greatly improves the visual quality of the final image.

4.1 Discussion

A nice characteristic of our method is that the indirect lighting can be easily updated at a different rate than the direct one. So, for scenes with static or smoothly changing geometry and lighting conditions the cost of the indirect illumination can be amortized among many frames without introducing any visible artifacts. In other words, the rate of indirect lighting updates can be reduced to meet the final performance goals.

Since classic voxelization is a rough discretization of the scene geometry, secondary shadows from small scale geometric details cannot be reproduced accurately by our method. Higher voxel resolutions can always be used, but with a performance hit. Also, due to graphics hardware limitations, we only used second order spherical harmonics, which they don't have sufficient accuracy to represent high frequency indirect light. This is not crucial if the direct illumination covers large parts of a scene yielding only very low-frequency indirect shadows in the first place. Interestingly, imperfect shadow maps have exactly the same issue (but for different reasons) but we think that our method is preferable since it does not require the maintenance of a secondary point based scene representation and the performance is mostly independent from final image resolution.

The performance and quality of our method depends on two parameters: the volume resolution and the number of iterations. Empirically, we have found that a grid size of 32 is sufficient in most cases. For outdoor scenes we have found that a low iteration count (around 12) is sufficient but for indoor ones a much higher iteration count is required (around 64) to accurately simulate the bouncing of the light inside the building.

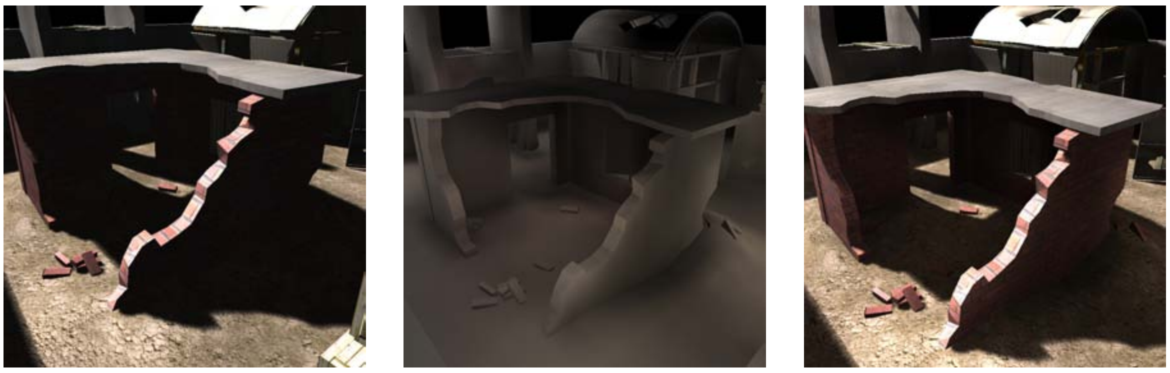


Figure 6. From left to right: The arena scene lit with direct lighting, indirect lighting only, final image with direct and indirect lighting

5. CONCLUSION AND FUTURE WORK

We have presented a new method for the computation of indirect diffuse light transport in large and fully dynamic scenes in real-time. Unlike previous work, our method takes in to account indirect occlusion and secondary light bounces. We have demonstrated that our method gives good results in a variety of test cases and always maintains a high frame rate.

Since the test results showed that the voxelization step is relatively costly, in the future we intent to introduce a much faster voxelization scheme. Another interesting direction of research is to extend this method to take specular light transport in to account.

REFERENCES

Chen, H. and Fang, S., 1998. Fast voxelization of 3D synthetic objects. In *Journal of Graphic Tools*, Vol. 3, No. 4, pp 33–45.

Chatelier, P., Malgouyres, R., 2005. A Low Complexity Discrete Radiosity Method. In Proc. of Discrete Geometry for Computer Imagery. pp 392–403

Dachsbacher, C. and Stamminger, M., 2005. Reflective shadow maps. In Proc. of ACM SI3D 2005, ACM, New York, NY, USA, pp 203–231.

Dachsbacher, C. and Stamminger, M., 2006. Splatting indirect illumination. In Proc. of ACM SI3D 2006, ACM, New York, NY, USA, pp 93–100.

Eisemann, E. and Décoret, X., 2008. Single-pass GPU Solid Voxelization for Real-time Applications. In GI ’08: Proceedings of Graphics Interface 2008, Information Processing Society, pp. 73–80.

Greger, G., Shirley, P., Hubbard, P., Greenberg, D. 1997. The Irradiance Volume. In *IEEE Computer Graphics and Applications*, Vol 18, pp. 32-43.

Jensen, H. W., 1996. Global illumination using photon maps. In Proceedings of the Eurographics Workshop on Rendering techniques, Springer-Verlag, London, UK, pp. 21–30.

Kajiya, J. T., 1986. The rendering equation. In Proceedings of *ACM SIGGRAPH Comput. Graph.* Vol 20, No 4, pp. 143-150.

Kaplanyan, A., 2009. Light Propagation Volumes in CryEngine 3. In *ACM SIGGRAPH Courses*, August 03, 2009.

Keller A., 1997. Instant radiosity. In Proceedings of SIGGRAPH, pp. 49–56.

McGuire, M. and Luebke, D., 2009. Hardware-accelerated global illumination by image space photon mapping. In Proceedings of the Conference on High Performance Graphics, New Orleans, Louisiana.

Nijasure, M., Pattanaik, S., Goel, V., 2005.: Real-time global illumination on the GPU. In Journal of Graphics Tools, 10(2):55--71.

Ritschel, T., Grosch, T., Kim, M. H., Seidel, H.-P., Dachsbacher, C. and Kautz, J., 2008. Imperfect shadow maps for efficient computation of indirect illumination. ACM Trans. Graph. (Proc. SIGGRAPH Asia) 27, 5, 129:1–129:8.

Ritschel, T. and Grosch, T., Seidel, H.-P., 2009. Approximating dynamic global illumination in image space. In Proceedings of the 2009 symposium on Interactive 3D graphics and games. New York, NY, USA: ACM, pp. 75-82.

Shanmugam, P. and Arikan, O., 2007. Hardware Accelerated Ambient Occlusion Techniques on GPUs. In Proceedings of ACM Symposium in Interactive 3D Graphics and Games, ACM, B. Gooch and P.-P. J. Sloan, Eds., pp. 73–80.

Sloan, P, 2008. Stupid Spherical Harmonics (SH) Tricks. In Game Developers Conference 2008.

Walter B., Fernandez S., Arbree A., Bala K., Donikian M., Greenberg D., 2005. Lightcuts: A scalable approach to illumination. In Proceedings of SIGGRAPH, pp. 1098 – 1107.

Ward, G. J., Rubinstein, F. M., Clear, R. D, 1988. A ray tracing solution for diffuse interreflection. In Proceedings of SIGGRAPH, pp. 85–92.

Wang, R., Wang, R., Zhoun, K., Pan, M. and Bao, H., 2009. An efficient GPU-based approach for interactive global illumination. ACM Trans. Graph. (SIGGRAPH) Vol 28, No 3.