GLOBAL ILLUMINATION USING IMPERFECT VOLUMES

Pavlos Mavridis and Georgios Papaioannou
Department of Informatics, Athens University of Economics & Business, Athens, Greece

Keywords: Real-Time Diffuse Global Illumination, Spherical Harmonics, Voxelization, Ray Marching.

Abstract: This paper introduces the concept of imperfect volumes, a fast one-pass point-based voxelization algorithm, and presents its applications to the global illumination problem. As often noted, diffuse indirect illumination has the characteristics of a low frequency function, consisting of smooth gradations. We exploit this by performing the indirect lighting computations on a rough approximation of the scene, the imperfect volume. The scene is converted on the fly to a dense point cloud, and each point is directly rendered to a volume texture, marking the corresponding voxel as occupied. A framebuffer reprojection scheme ensures that voxels visible to the main camera will get more points. Ray-marching is then used to compute the ambient occlusion or the indirect illumination of each voxel, and the results are stored using spherical harmonics. We demonstrate that the errors introduced by the imperfections in the volume are small and that our method maintains a high frame rate on scenes with high geometric complexity.

1 INTRODUCTION

In computer graphics the goal of global illumination algorithms is to produce convincing images of an artificial world. Given a scene description, including the geometry, surface properties and light source descriptions, they simulate the complex interactions of the light with the world, like diffuse and specular interreflections, in order to generate realistic and accurate images. Such accuracy is desired for architectural visualization, feature film production and even for real-time applications, but is often omitted due to the high cost associated with the calculation of global illumination effects.

High quality global illumination at interactive speed is still an unsolved problem for large and dynamic scenes. In this paper we propose a method that produces realistic images of diffuse, dynamic environments in real time, by estimating the diffuse indirect light transport at discrete locations in the environment and applying the results on the scene geometry. To do so, we introduce the *imperfect volume*, a rough approximation of the scene, storing occupancy and lighting information in a uniform grid data structure. The way this volume is constructed ensures that visible surfaces will get a nearly perfect voxelization, through a frame buffer reprojection scheme, while the rest of the scene may contain inaccuracies, like missing voxels. Radiance or ambient occlusion (AO) is computed at the center of each voxel using volume ray-marching and the results are stored in the volume using spherical harmonics. Finally the diffuse indirect illumination of each visible point is reconstructed by sampling the radiance of the closest voxels in the volume. We demonstrate that the low frequency nature of the diffuse indirect illumination tends to mask the introduced imperfections, while the performance gains from such an approximation are substantial.

2 PREVIOUS WORK

Instant radiosity methods (Keller, 1997) approximate the indirect illumination of a scene using a set of *Virtual Point Lights* (VPLs). VPLs are created by tracing photos on the scene, then the scene is rendered, as lit by each VPL. The major cost of this method is the visibility queries for a large number of point lights. Lightcuts (Walter et al., 2005) reduce the number of the required queries by clustering the VPLs in groups, but the performance is still far from real-time.

Reflective shadow maps (Dachsbacher and Stamminger, 2005)(Dachsbacher and Stamminger, 2006) 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

Mavridis P. and Papaioannou G..
GLOBAL ILLUMINATION USING IMPERFECT VOLUMES.
DOI: 10.5220/0003318601600165
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2011), pages 160-165
ISBN: 978-989-8425-45-4

frame rates, screen space interpolation is required and the method is limited to the first bounce of indirect illumination. Imperfect shadow maps (ISM) (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 of the shadow maps. Like ISMs, our method also uses an imperfect representation of the scene, but our method has much better scalability with the final image resolution.

Micro-Rendering (Ritschel et al., 2009a) gathers illumination by rasterizing a point based representation of the scene from many different viewpoints in parallel. The results are accurate but the frame rate is interactive only in relatively simple scenes.

McGuire (McGuire and Luebke, 2009) adapts the photon mapping (Jensen, 1996) algorithm on the GPU. The first bounce of the photons is computed using rasterization on the GPU, but the rest of the bounces are computed on the CPU, making the algorithm impractical unless a large number of parallel cores are used.

Directional occlusion (Ritschel et al., 2009b) extends previous screen space methods (Shanmugam and Arikan, 2007) and approximates the first indirect diffuse bounce of the light by only using information in the frame buffer. The computational cost is low but the resulting illumination is hardly accurate since it depends on the projection of the visible objects on the screen.

The irradiance cache (Ward et al., 1988) stores accurate irradiance estimates on a few surface points and for the remaining ones fast interpolation is used. Radiance caching (Křivánek et al., 2005) extends the concept to store radiance instead of irradiance, using spherical harmonics. (Wang et al., 2009) presents a method to calculate the radiance sample points in advance and implements the algorithm on the GPU. The method is accurate but achieves interactive frame rates only in very simple scenes. (Nijasure et al., 2004) places the sample points on a grid, but to estimate the radiance on each point, the complete scene must be rendered on a cube map, making the method expensive. (Kaplanyan and Dachsbacher, 2010) avoids this cost by using an iterative propagation scheme to calculate the radiance distribution on the scene. The method achieves high performance but indirect occlusion is only limited to surfaces visible in the camera and the shadowmaps. Compared to that, the *imperfect volume* has the advantage that occlusion information is available for the whole scene.

(Papaioannou et al., 2010) computes AO by tracing rays on a volume representation of the scene from visible points, but to achieve real time performance AO is computed at a lower resolution. Our method also traces rays in a volume, but can achieve better performance by doing it at regular intervals in world space and not for every visible surface point.

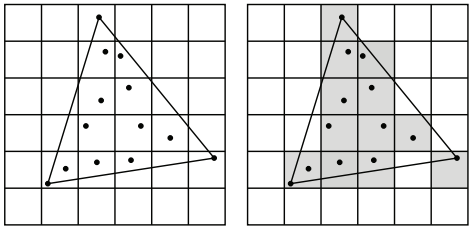

Figure 1: For each input triangle, random points are created, marking the corresponding voxels as occupied.

3 METHOD OVERVIEW

Our method first creates the imperfect volume from a dense point cloud representation of the scene, created on the fly using geometry shaders. After that, the imperfect volume is refined using the visible points from the main camera. Next, the incoming radiance or the AO of each voxel is computed using ray-marching and monte carlo integration and finally, during image rendering, the irradiance of each surface point is calculated taking into account the radiance of the nearest voxels.

3.1 Imperfect Volume Creation

Our goal is to create a volume representation of the full scene, storing occupancy and illumination information. Previous fast one-pass voxelization algorithms, like (Eisemann and Décoret, 2008), cannot be used because our algorithm requires multi-channel floating point data rather than binary values in each voxel. Since we consider our scenes fully dynamic, the complete volume representation must be rebuilt in every frame, so a high performance algorithm is essential for our method.

The volume is created from a dense point-cloud representation of the scene. Although we can precalculate the point cloud in advance, we chose to generate it on the fly using the geometry shader functionality of the latest graphics cards. This approach has the advantage that an additional point-based representation and storage of the scene is not required, something that simplifies the integration with the rendering pipeline of typical real-time applications.

For each input triangle with area A, N random

points are generated with probability density $p(x) = 1/A$. The geometry shader takes as input a triangle and emits N points. For each emitted point, the appropriate slice of the volume is calculated and the point is directly rendered to this slice, marking the corresponding voxel as occupied. For each point, the direct illumination is calculated using the interpolated normal from the input triangle and the result is stored in the volume using a spherical harmonics representation.

The obvious flaw in the above algorithm is that there is guarantee that every occupied voxel will receive a random point, but as the density of the generated points increases, the probability that an occupied will not receive a random point tends to zero.

It should be noted that geometry shaders are not fast doing data amplification, so when producing more random points the performance quickly drops. It's more preferable to tessellate big triangles at load or content creation time, than to generate more points at run-time. We have also experimented with adaptively adjusting the number of generated points depending on the size of the triangle, but the performance turned out to be slower.

3.2 Visible Point Reprojection

In this step the visible regions of the imperfect volume are refined with data from the framebuffer. Before this step, the depth, normals and albedo of the surface points visible to the camera are recorded in a g-buffer (Akenine-Möller et al., 2008) using deferred rendering. Direct illumination is calculated for all fragments and the visible points from the main camera are reprojected in the imperfect volume and the corresponding voxels are updated.

To perform this reprojection, an array of points is used, each point corresponding to a pixel in the framebuffer. Each point reads the depth of the corresponding pixel in the vertex shader and is projected from clip coordinates back to world space, to be reprojected in the imperfect volume. A geometry shader routes the points to a slice of the 3D volume according to their depth in world space coordinates. Finally, the fragment shader reads the albedo, normal and the direct lighting of the point, already computed and stored by the deferred renderer, and stores it to the volume using a spherical harmonics representation according to the normal of the point.

Since the number of the visible points that are projected in the volume is considerably larger than the number of the visible voxels, this operation results in an almost perfect voxelization of the visible surfaces, eliminating most of the potential gaps in the visible

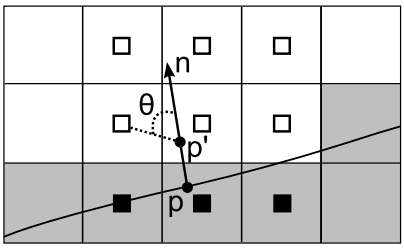

Figure 2: Computing the radiance or visibility distribution at surface point p. To avoid bias from self intersections, the point is displaced half a voxel along the normal n to the position p', and a more accurate radiance distribution is recomputed from the radiance of neighboring voxels. Voxels behind the surface, marked with a black square, do not contribute to the computation.

portion of the imperfect volume.

3.3 Volume Sampling and Radiance Caching

The incoming radiance distribution of each voxel L_i is stored in the volume as a vector of spherical harmonic coefficients λ_l^m, such as

$$L_i(\theta,\phi) = \sum_{l=0}^{n-1} \sum_{m=-l}^{l} \lambda_l^m Y_l^m(\theta,\phi) \qquad (1)$$

where n is the order of the SH representation and Y_l^m are the spherical harmonic basis functions(Ramamoorthi and Hanrahan, 2001). Our implementation uses a 2^{nd} order spherical harmonic representation, since the four SH coefficients map very well to the four component buffers supported by the graphics hardware. The coefficients λ_l^m can be computed with the following integral

$$\lambda_l^m = \int_0^{2\pi} \int_{-\pi/2}^{\pi/2} L_i(\theta,\phi) Y_l^m(\theta,\phi) \sin\theta d\theta d\phi \qquad (2)$$

Since we don't have an analytical form for L_i, but we can take samples of this function using raycasting, we compute λ_l^m using monte carlo quadrature with uniform sampling

$$\lambda_l^m = \frac{4\pi}{N} \sum_{j=1}^{N} L_i(\theta_j,\phi_j) Y_l^m(\theta_j,\phi_j) \qquad (3)$$

where L_i is the incoming radiance from the (θ_j,ϕ_j) direction of the sphere and N is the total number of samples.

In order to compute the L_i term in equation 3, we sample the radiance that is stored in the volume representation of the scene. For every voxel, N random directions on the sphere are created using stratified sampling, and rays starting from the center of the voxel

are traced using ray-marching, a process where the volume is sampled in regular intervals along the ray, until an occupied cell is found or the extends of volume are reached.

In the case of AO calculation, we sample and store the directional visibility instead of the incoming radiance. We also limit the ray-marching distance to a short range, making the AO calculation much faster than GI.

Even though the rays start from the center of each voxel, to avoid self intersections the actual ray-marching should start outside the originating voxel. To do this an initial distance d_s along each ray should be skipped. For cubic voxels of size s_v the distance that we skip is $d_s = \frac{\sqrt{3}}{2} s_v$, which is the radius of the bounding sphere of the voxel. This scheme does not avoid self intersections with neighboring voxels potentially generated from the same polygon, but we skip these intersections when doing the final per-pixel irradiance reconstruction, as described in the next section.

Our method samples the illumination from one volume buffer and writes the results to another one. By alternating between those two buffers in successive passes we compute multiple bounces of the light.

3.4 Irradiance Reconstruction

The diffusely reflected light, or *radiosity B*, of a point x on a surface with diffuse reflectivity ρ is given by the following equation (Kajiya, 1986):

$$B(x) = \frac{\rho(x)}{\pi} \int_{\Omega} L_i(x,\omega) \cos\theta d\omega \qquad (4)$$

where θ is the angle between the surface normal and the incident radiance direction ω.

During the final scene rendering, equation 4 must be evaluated for every visible surface point in order to determine the color of each pixel. To determine the incident radiance L_i of a surface point in the scene, we don't take in to account the radiance of the corresponding voxel in the volume, because the stored radiance distribution will be biased from intersections with neighboring voxels, potentially generated from the same surface. Instead, we shift the surface point outside the current voxel by moving it half a voxel along the normal, and then we recalculate a more accurate distribution of the radiance, taking in to account the surface orientation and the radiance of the N closest voxels, using the following equation

$$\dot{L}_i = \frac{\sum_{j=1}^{N} w_j \dot{L}_{ij}}{\sum_{j=1}^{N} w_j}, \quad where \quad w_j = \begin{cases} \cos\theta, & \theta < \pi/2 \\ 0, & \theta > \pi/2 \end{cases} \qquad (5)$$

Table 1: The performance of our method (times in milliseconds on a NVIDIA GTX460). G: number of triangles, T_i: time when emitting i vertices per triangle, T_{AO}, T_{GI}: Ray-marching time for AO and GI(per bounce) respectively. The framebuffer reprojection time for 512^2 points is 0.7ms on all scenes.

	G	T_3	T_9	T_{AO}	T_{GI}
sponza	262267	3.8	11.6	4.4	31.2
knossos	109168	1.5	4.5	4.3	32.9
arena	10219	0.2	0.6	4.6	35.7
room	8760	0.3	0.8	4.3	35.0

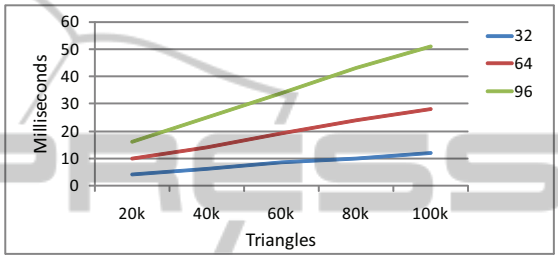

Figure 3: Volume creation scaling with the number of triangles and the volume size. When the volume resolution is doubled, the number of generated points per triangle are also doubled, to keep the quality of the voxelization identical. Times measured in milliseconds on a ATI Radeon 3650.

where $\dot{L}$ denotes the spherical harmonic representation of L. The weights w_i, as illustrated in Figure 2, guarantee that voxels behind the surface will not contribute to the radiance computation, and that voxels facing the normal of the surface will contribute the most. For performance reasons, in the actual implementation of the algorithm we only consider the six nearest voxels.

Finally, since the radiance $\dot{L}_i$ is represented using spherical harmonics, we compute the irradiance integral in equation 4 as a simple dot product with an analytically computed spherical harmonic representation of the cosine lobe(Ramamoorthi and Hanrahan, 2001), directed towards the surface normal.

4 RESULTS

We have integrated the above algorithm in a traditional real-time deferred renderer using OpenGL and GLSL. Unless otherwise noted, all figures presented here are created with 12 random points per triangle, 100 rays per voxel, 64^3 voxels and a ray marching step size of one voxel.

Table 1 shows comprehensive time measurements for our method. All scenes are considered to have

Figure 4: Left: Diffuse GI from one point light source. Notice the indirect light on the crates. Right: Occlusion using a) 9 points per triangle, b) 12 points per triangle, c) 3 points per triangle plus voxelization with framebuffer reprojection.

fully dynamic geometry and lighting conditions. We can see that even on the most complex scenes, our algorithm maintains an interactive framerate and, as expected, the AO is much faster than GI, since the ray-marched distance is shorter and only the coverage information is fetched from the volume.

As shown, imperfect volumes can be created extremely fast, even for very large datasets. The performance of our method, but also the quality of the resulting volume, depends on the number of the generated points per triangle. In most scenes we get nearly perfect voxelization with 12 to 19 points. But since the framebuffer reprojection scheme guarantees a nearly perfect voxelization of the visible portion of the imperfect volume, in practice we rarely need to use more than 6 points per triangle.

Figure 3 shows how the volume creation algorithm scales with the number of input triangles and the volume resolution. When the volume resolution is doubled we also double the number of the generated points, to keep the density of the points per voxel constant. We observe sub-linear scaling with the number of triangles and linear scaling with the volume size.

Figure 4 shows how the imperfections of the volume are refined by the framebuffer reprojection. These imperfections result in smoothed out contact shadows, but on the visible part of the volume are corrected by the reprojection of the framebuffer. Any further imperfections in the invisible parts of the scene do not produce any objectionable errors in the illumination, and they are mostly unnoticeable in the final textured image.

Figure 5 illustrates how the volume resolution affects the final image quality on the room scene. We can see that when the volume resolution is insufficient, small scale details like the contact shadows of the table, are lost. Figure 6 shows several shots of the knossos scene, using ambient occlusion and diffuse

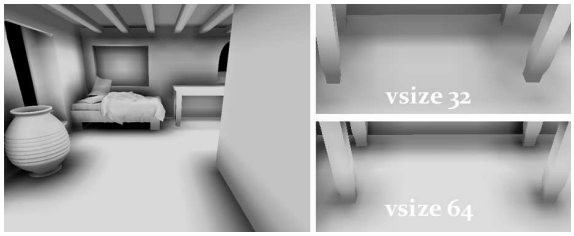

Figure 5: Left: Ambient occlusion on the room scene. Right: Small scale details are lost when using insufficient volume resolution.

global illumination.

4.1 Discussion and Limitations

Since our method operates in voxel space, the performance is mostly independent of the final image resolution. The performance and quality depends on the number of points generated per triangle, the number of rays traced per voxel, the step size when traversing the volume and the volume resolution.

Since even perfect voxelization is a rough approximation of the scene geometry, GI from small scale geometric details cannot be reproduced accurately. A multi-scale approach like the cascaded volumes (Kaplanyan and Dachsbacher, 2010) could be applied to alleviate this problem. Also, we have observed some temporal aliasing in the resulting illumination, as the state of the voxels changes from occupied to unoccupied. One way to counter this phenomenon is to sample the radiance at more positions inside the voxels, and not just the centers.

5 CONCLUSIONS

We have presented a method for the real-time computation of ambient occlusion and diffuse indirect light transport in fully dynamic scenes. Our method operates on a rough approximation of the entire scene, created by a fast point-based vozelization algorithm. Although this algorithm does not give any hard guarantees about the quality of the voxelization, we have demonstrated that potential errors introduced by the imperfections in the volume are small and that our method always maintains an interactive frame rate in a variety of test cases.

Figure 6: Several shots from of the knossos scene, showcasing ambient occlusion (left and right) and global illumination using three bounces (middle).

REFERENCES

Akenine-Möller, T., Haines, E., and Hoffman, N. (2008). *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA.

Dachsbacher, C. and Stamminger, M. (2005). Reflective shadow maps. In *Proceedings of the 2005 ACM Symposium on Interactive 3D Graphics and Games*, pages 203–231. ACM SIGGRAPH.

Dachsbacher, C. and Stamminger, M. (2006). Splatting indirect illumination. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pages 93–100. ACM SIGGRAPH, ACM Press.

Eisemann, E. and Décoret, X. (2008). Single-pass gpu solid voxelization for real-time applications. In *GI '08: Proceedings of graphics interface 2008*, pages 73–80. Canadian Information Processing Society.

Jensen, H. W. (1996). Global Illumination Using Photon Maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, pages 21–30. Springer-Verlag/Wien.

Kajiya, J. T. (1986). The Rendering Equation. In *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150.

Kaplanyan, A. and Dachsbacher, C. (2010). Cascaded light propagation volumes for real-time indirect illumination. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 99–107, New York, NY, USA. ACM.

Keller, A. (1997). Instant radiosity. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, volume 31, pages 49–56.

Křivánek, J., Gautron, P., Pattanaik, S., and Bouatouch, K. (2005). Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics*, 11(5).

McGuire, M. and Luebke, D. (2009). Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the 2009 ACM SIGGRAPH/EuroGraphics conference on High Performance Graphics*, New York, NY, USA. ACM.

Nijasure, M., Pattanaik, S., and Goel, V. (2004). Real-time global illumination on the GPU. *Journal of Graphics Tools*, 10(2).

Papaioannou, G., Menexi, M. L., and Papadopoulos, C. (2010). Real-time volume-based ambient occlusion. *IEEE Transactions on Visualization and Computer Graphics*, 99(RapidPosts).

Ramamoorthi, R. and Hanrahan, P. (2001). An efficient representation for irradiance environment maps. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 497–500, New York, NY, USA. ACM.

Ritschel, T., Englehardt, T., Grosch, T., Seidel, H.-P., Kautz, J., and Dachsbacher, C. (2009a). Micro-rendering for scable, parallel final gathering. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2009)*, 28(5).

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 Transactions on Graphics*, 27(5).

Ritschel, T., Grosch, T., and Seidel, H.-P. (2009b). Approximating dynamic global illumination in image space. In *Proc. ACM Symposium on Interactive 3D Graphics and Games 2009 (I3D '09)*.

Shanmugam, P. and Arikan, O. (2007). Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 73–80. ACM.

Walter, B., Fernandez, S., Abree, A., Bala, K., Onikian, M., and Greenberg, D. P. (2005). Lightcuts: A scalable approach to illumination. In *ACM SIGGRAPH 2005 Full Conference DVD-ROM*, pages 1098–1107.

Wang, R., Wang, R., Zhou, K., Pan, M., and Bao, H. (2009). An efficient GPU-based approach for interactive global illumination. volume 28.

Ward, G. J., Rubinstein, F. M., and Clear, R. D. (1988). A Ray Tracing Solution for Diffuse Interreflection. In *Computer Graphics (ACM SIGGRAPH '88 Proceedings)*, volume 22, pages 85–92.