GLOBAL ILLUMINATION USING IMPERFECT VOLUMES

Pavlos Mavridis , Georgios Papaioannou
Department of Informatics, Athens University of Economics & Business, Greece
pmavridis@aueb.gr, gepap@aueb.gr

Keywords: Real-Time Global Illumination, Spherical Harmonics, Voxelization

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. 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.

Our main contributions include:

- An approximate representation of the scene, the *imperfect volume*, facilitating the calculations of global illumination effects in real-time.
- A method to reconstruct the irradiance of the scene surfaces by sampling the radiance stored in the volume, avoiding bias from self intersections.
- An analysis on the influence *imperfect volumes* have on the resulting illumination.

We demonstrate that our method achieves interactive speed on complex, fully dynamic scenes, even on mainstream graphics hardware.

2 PREVIOUS WORK

Instant radiosity methods, introduced by Keller (Keller, 1997), approximate the indirect illumination of a scene using a set of *Virtual Point Lights* (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. Lightcuts (Walter et al., 2005) reduce the number of the required shadow 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 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 and the low resolution of the shadow maps. The concept of ISMs is similar to our work, but our method has some advantages, as detailed in section 4.1, like much better scalability with the final image resolution.

Micro-Rendering (Ritschel et al., 2009a) is an efficient method to perform final gathering, by rasterizing a point based representation of the scene from many different viewpoints in parallel. The method gives accurate results but achieves interactive frame rates only in relatively simple scenes.

Photon mapping (Jensen, 1996) traces photons from the light sources into the scene and stores them 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 (McGuire and Luebke, 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.

Directional occlusion (Ritschel et al., 2009b) extends previous methods for screen space ambient occlusion calculation (Shanmugam and Arikan, 2007) 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 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 points) and for the remaining ones fast interpolation is used. Radiance caching (Křivánek et al., 2005) extends the irradiance cache to store and interpolate a radiance representation of the indirect illumination instead of irradiance, using spherical harmonics. This method needs fewer samples in the cache, and can handle specular surfaces. Wang (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. The *imperfect volume* can be seen as a radiance cache, but also stores occupancy information to perform the raytracing calculations directly on the volume and not the scene geometry and uses a uniform grid arrangement of the cache points to facilitate fast computation on the GPU.

Nijasure (Nijasure et al., 2004) 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 (Kaplanyan and Dachsbacher, 2010) also uses a regular grid to store the scene radiance and 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 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 (Papaioannou et al., 2010) uses a volume representation of the scene where rays are traced from surface points through the volume to determine the occlusion of each visible point, but to achieve real time performance ambient occlusion is rendered 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.

3 Method Overview

Our method consists of following stages: first a dense point cloud representation of the scene is created on-the-fly using geometry shaders, and is projected in the *imperfect volume*, storing the occupancy and the direct illumination of each voxel. After that, the imperfect volume is refined by reprojecting the visible points of the framebuffer into the volume, creating a nearly perfect voxelization of the visible surfaces. Next, the incoming radiance of each voxel is calculated by sampling the volume using ray-marching and monte carlo quadrature and the results are stored using a vector of spherical harmonic coefficients. 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 the storage of arbitrary data in each voxel and should work on any object, not necessarily watertight. 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.

We observe that diffuse indirect illumination has the characteristics of a low frequency function, consisting of smooth gradations, which tend to mask errors due to incorrect visibility. To exploit this we compute the indirect illumination on a rough approximation of the scene, an *imperfect volume*. While *imperfect volumes* may contain missing voxels (holes), the resulting errors in the indirect illumination are small but the computational gains are significant, as we will demonstrate in detail in Section 4.

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 the maintenance of an alternate point-based representation of the scene is not required, something that simplifies the integration with the rendering pipeline of typical real-time applications.

For each input triangle, N random points v_{rand} are generated over the surface A of the triangle with probability density $p(x) = 1/A$, using the following equation(Turk, 1990):

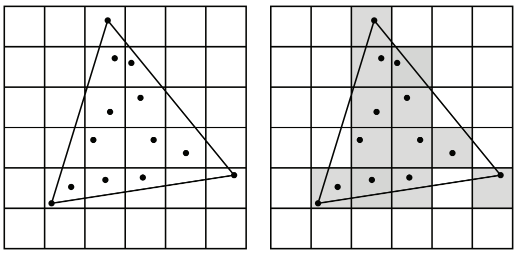

Figure 1: For each input triangle, N random points are created, marking the corresponding voxels as occupied. Each point is directly rendered to the appropriate volume layer, according to its depth.

$$v_{rand} = (1-\sqrt{r_1})v_1 + (1-r_2)\sqrt{r_1}v_2 + r_2\sqrt{r1}v_3$$

where v_1, v_2 and v_3 are the vertices of the triangle and r_1, r_2 are random numbers, uniformly generated over the interval $[0,1]$. 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 no guarantee that every voxel containing a triangle will receive a random point, as shown in figure 1. As the density $d = N/A$ of the generated points increases, the probability that an occupied voxel will not receive a random point decreases. The density of the generated points is proportional to the number N of the generated points per triangle, and inversely proportional to the area of the input triangles. In other words our algorithm gives better results when the number of the generated points increases and when the size of the input triangles is small relatively to size of the voxels.

It should be noted that geometry shaders are not fast doing data amplification (generating new geometry), 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. Also we have tried to generate a more uniform distribution of points by doing a regular tessellation of the input triangle. Both approaches turned out to be slower by a very wide margin. We also experimented with the dedicated hardware tessellation units, to produce tessellated triangles and then convert them to points in the geometry shader, but interestingly this approach turned out to be slower as well.

3.2 Framebuffer Reprojection

In this step the visible regions of the imperfect volume are refined with data from the framebuffer. Before this step, a typical deferred renderer renders the scene on a g-buffer structure (Akenine-Möller et al., 2008), containing the depth, normals and albedo of the visible materials and using those buffers calculates the direct lighting of the visible surfaces. The visible points of the depth buffer are then projected back in world space, and then are reprojected in the voxels of the imperfect volume.

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 reprojected in the imperfect volume. A geometry shader routes the points to a slice of the 3d volume according to their depth. Finally, the fragment shader reads the albedo, normal and the direct lighting of the point, already computed and stored by the deferred renderer, and injects 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 imperfections in the visible portion of the imperfect volume. And because the direct illumination is already computed by the deferred renderer, this operation is extremely fast (about $0.6ms$ for 512^2 points on a nvidia gtx460). Using this technique, the number of random points per triangle needed in the previous step is considerably reduced, and the overall performance and the quality of the method increases.

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.

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. Papaioannou (Papaioannou et al., 2010) proposes an elaborate variable guard distance to avoid self intersections when ray-marching volume data, but this measure can only be used when the ray-marching begins from the surfaces, because the actual surface normal is required for the calculations. In our case we use a variation of this measure. For cubic

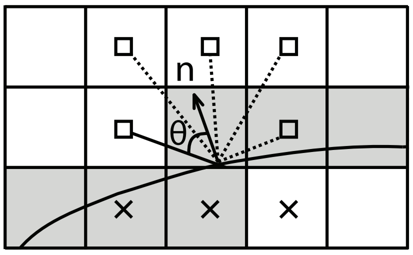

Figure 2: Irradiance reconstruction for a surface point from 8 neighboring voxels. To avoid bias from self intersections we ignore the values in the current occupied voxel, and we reconstruct the irradiance from the neighboring voxels. The voxels behind the surface, marked with an “x” do not contribute to the computation.

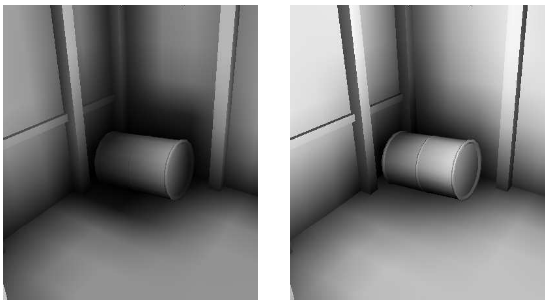
Figure 3: On the left direct visualization of the radiance in the volume. Notice how the occlusion effect is exagerated and flat surfaces appear too dark. On the right proper reconstruction with the proposed scheme.

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 (or the distance from the center to the corners). 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. The integral in this equation computes the power that x receives and is called *irradiance* and the integration domain Ω is the hemisphere defined by the surface normal at point x.

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

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

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. Finally the computed irradiance is linearly interpolated by the graphics hardware. Figure3 demonstrates the effectiveness of this method. 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 the spherical harmonic representation of the cosine lobe(Ramamoorthi and Hanrahan, 2001), directed towards the surface normal.

3.5 Ambient Occlusion Calculation

Ambient occlusion is a rough approximation of the global illumination, where the radiance distribution of the whole scene is considered constant. Equation 4 can then be rewritten as

$$A(x) = \frac{1}{\pi} \int_{\Omega} V_i(x,\omega) \cos\theta d\omega \qquad (6)$$

where the incoming radiance L_i has been replaced with the visibility term V_i.

In the case of AO calculation, when an occupied voxel is reached instead of returning its stored radiance, we return zero. If no obstacle is reached, we return one. We also limit the maximum traversed distance along the rays to a short distance. The rest of the algorithm remains the same, but these simplifications make the AO calculation much faster than GI.

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 19 random points per triangle, 100 rays per voxel, 64^3 voxels and a ray marching step size of one voxel. All the time measurements are on a nvidia gtx460, unless otherwise noted.

	G	T_3	T_7	T_{12}	T_{19}	T_{rp}
sponza	262267	22	52	86	125	0.7
knossos	109168	11	22	35	55	0.7
arena	10219	2	4	5	7	0.7
room	8760	4	5.4	8	10	0.7

Table 1: The performance of imperfect volume creation. G: number of triangles, T_i: times in milliseconds when emitting i vertices per triangle, T_{rp}: times for framebuffer reprojection using 512^2 points.

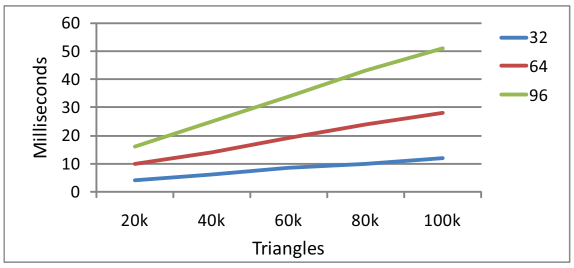

Figure 4: Volume creation scaling with the number of triangles and the volume size. When the volume resolution is doubled, the number of generated points per triangle are also doubled, to keep the quality of the voxelization identical. Times measured in milliseconds on a ATI Radeon 3650.

Table 1 shows the time required to create the imperfect volume on various scenes. 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. The framebuffer reprojection scheme guaranties that the visible portion of the imperfect volume will always get a nearly perfect voxelization at extremely low cost, independently from the number of the generated random points per triangle. But since we are using the imperfect volume to compute global illumination effects, the invisible parts of the volume are also needed. If we skip the tessellation step, and we operate only on the visible voxels created by the framebuffer reprojection, then our method has the same disadvantages as the screen space methods(Ritschel et al., 2009b)(Shanmugam and Arikan, 2007), as shown in figure 5. By using the information in the invisible parts of the volume, our method maintains consistency between frames, unlike previous methods. In this spirit, our method can be seen as an extension of screen space methods to include information not visible in the framebuffer.

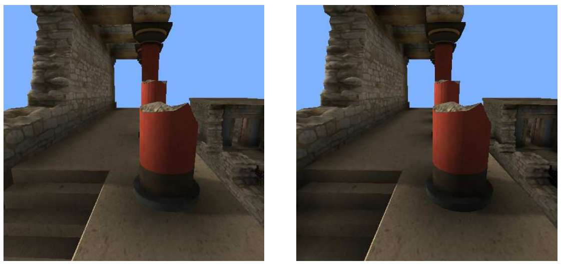

Figure 5: Left: AO with framebuffer reprojection only. Right: AO with 3 random points per triangle and framebuffer reprojection. Notice how the contact shadows from the invisible geometry is missing in the first case. All screenspace algoritms inevitably have this problem. In the second case the contact shadows are correctly reproduced, due to information available in the imperfect volume.

	G	R_{AO}	R_{GI}	I
sponza	262267	12	58	7
knossos	109168	12	52	6
arena	10219	16	60	5
room	8760	15	70	7

Table 2: Time measurements for our method (in milliseconds). G: number of triangles, R_{AO}: Ray-marching time for AO, R_{GI}: Ray-marching time for GI (per bounce), I: Reconstruction/Interpolation time.

Figure 4 shows how the 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.

Table 2 shows comprehensive time measurements for our method when calculating ambient occlusion and diffuse indirect illumination. All scenes are considered to have 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 raymarched distance is shorter and only the coverage information is fetched from the volume.

Figure 6 shows how the imperfections of the volume affect the final image and showcases the importance of framebuffer reprojection. We can see that when using few random points, the contact shadows in ambient occlusion are smoothed out. Imperfect shadow maps suffer from the same problem, but in our method this is corrected on the visible surfaces with 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 7 illustrates how the volume resolution af-

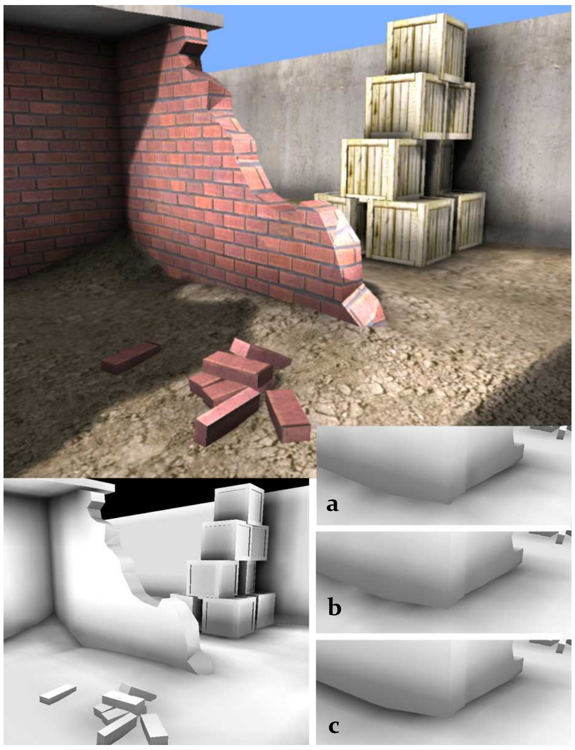

Figure 6: The arena scene with ambient occlusion. Bottom left: the ambient occlusion buffer. Bottom right: The resulting occlusion using a) 9 points per triangle, b) 12 points per triangle, c) 3 points per triangle plus voxelization with framebuffer reprojection.

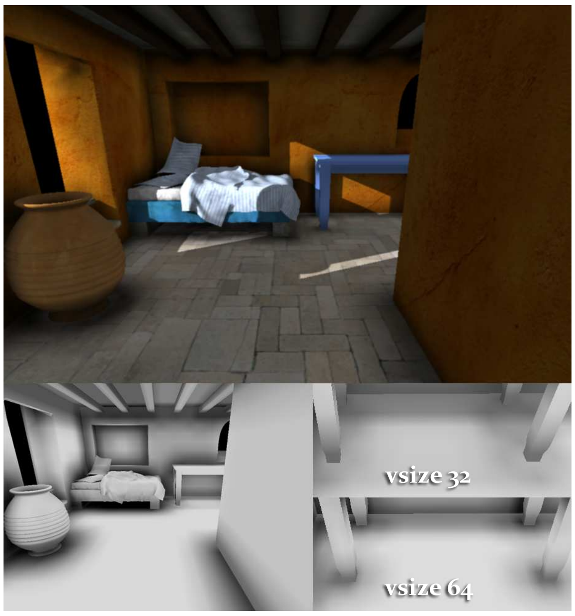

Figure 7: The room scene with ambient occlusion. Bottom left: The ambient occlusion render buffer. Bottom right: Small scale details are lost when using insufficient volume resolution.

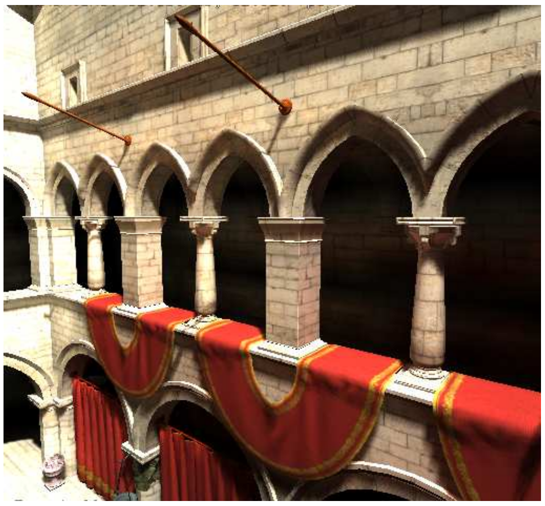

Figure 8: Global illumination in the sponza palace scene.

fects 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. On the other hand even a low resolution volume is enough to capture the variations in the shading of thin complicated objects like the vase, which appears dark from the inside, lit from the outside. This is achieved because each voxel stores the radiance distribution of the corresponding area, and not just a constant illumination value.

Figure 8 shows global illumination on the sponza palace model. The contribution of the GI has been scaled in the final image to better illustrate the effect of color bleeding. Figure 9 shows several shots of the knossos scene, using ambient occlusion and diffuse global illumination.

4.1 Discussion and Limitations

The performance of our method is mostly independent of the final image resolution, because the actual global illumination is calculated at regular intervals in world space. This is a big advantage over instant radiosity methods, like ISMs that need to compute the illumination in lower resolutions to achieve interactive framerates. Also, compared to ISMs our method does not require the maintenance and the constant update of a secondary point-based scene representation, making the algorithm more practical.

An interesting observation is that the performance of ray-marching vastly improved once we removed the branching from our implementation. The branchless implementation continues to march along the rays even when the first obstacle is found and uses conditional moves to discard any further results. This is

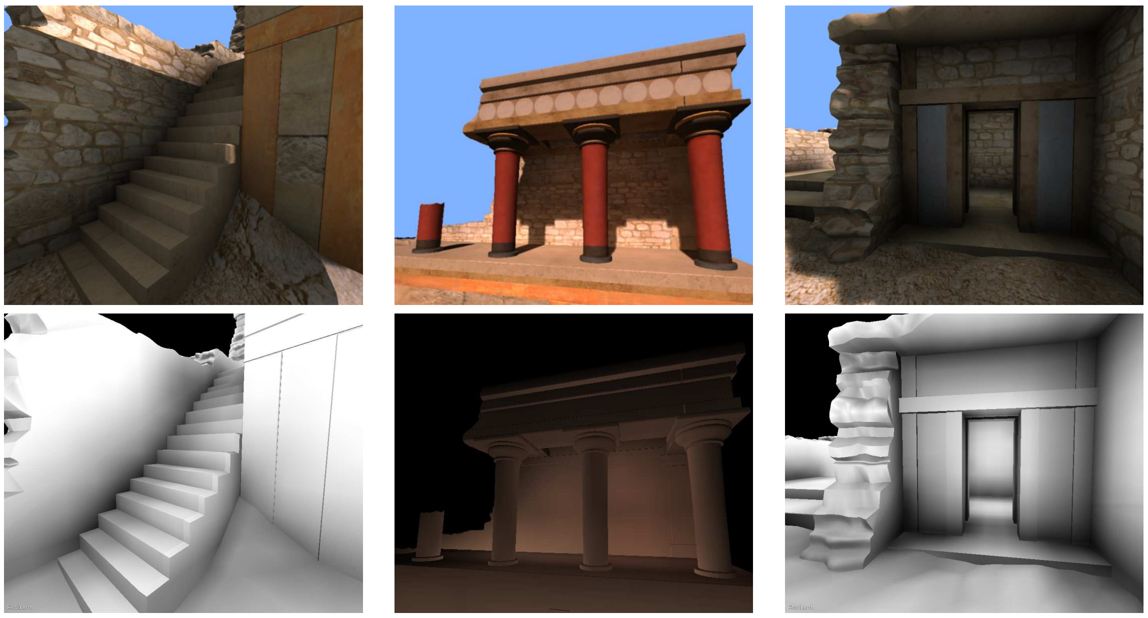

Figure 9: Several shots from of the knossos scene, showcasing ambient occlusion (left and right) and global illumination (middle). Three bounces were used for the indirect diffuse illumination.

to be expected since branching introduces irregular workload to the shader units, and the hardware scheduler of the GPUs is extremely inefficient in such cases (Ragan-Kelley, 2010).

Another advantage of our algorithm over instant radiosity methods, is that since no triangle scan-conversion is required, it can be implemented efficiently on any parallel architecture, like future many-core cpus, without the need of dedicated fixed-function hardware. That holds true for both the volume creation and the ray-marching phase of our method.

Instead of the used ray-marching scheme, the radiance propagation method from (Kaplanyan and Dachsbacher, 2010) could be used, in conjunction with the imperfect volume and our radiance reconstruction strategy. Our tests showed problems with the energy conservation of this method when multiple propagation steps are used. On the other hand the usage of volume ray-marching permited us to unify the calculation of AO and GI under the same method. While the advantages of ray-marching over the radiance propagation scheme could be argued, the *imperfect volume* we propose here contains more information about the invisible parts of the scene compared to the volume proposed by Kaplanyan (which is only build from framebuffer and shadowmap reprojection), with a moderate performance hit.

Since even perfect voxelization is a rough discretization of the scene geometry, global illumination effects from small scale geometric details cannot be reproduced accurately by our method. Higher voxel resolutions can always be used, but with a performance hit since more points per triangle should be generated in that case. On the other hand, our method is more efficient than screen space methods at computing large scale occlusion or GI and also includes information not available in the framebuffer.

Also, due to graphics hardware and memory bandwidth limitations, like most of the other real-time methods, we only used second order spherical harmonics, which lack 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.

Overall, we believe that our method improves on major shortcomings and trade-offs of previous methods, while the remaining limitations are justified in order to achieve the desired level of performance.

5 Conclusion and Future Work

We have presented a new method for the computation of ambient occlusion and diffuse indirect light transport in large and fully dynamic scenes in real-time. Our method operates on a rough approximation of the entire scene, an *imperfect volume*, 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.

We believe that imperfect volumes can also be useful for many types of algorithms that operate on volume data, like real-time smoke and water simulation, where high performance is essential and small inaccuracies in the voxelized geometry will be masked out in the results.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their helpful comments. The sponza palace model has been kindly provided by Crytek. The knossos model is available from the AUEB computer graphics group website.

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, Toronto, Ont., Canada, Canada. 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). Also available as Technical Report #1623, IRISA, http://graphics.cs.ucf.edu/RCache/index.php.

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).

Ragan-Kelley, J. (2010). Keeping many cores busy: Scheduling the graphics pipeline. In *SIGGRAPH '10: ACM SIGGRAPH 2010 Courses*, New York, NY, USA. ACM.

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, New York, NY, USA. ACM.

Turk, G. (1990). Generating random points in triangles. pages 24–28.

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.