Approximate Visibility Grids for Interactive Indirect Illumination

Thomas Bashford-Rogers, Kurt Debattista, Carlo Harvey and Alan Chalmers
IDL, University of Warwick, Coventry, UK
{t.bashford-rogers, carlo.harvey, k.debattista, alan.chalmers}@warwick.ac.uk

***Abstract*—The computation of indirect illumination is fundamental to simulate lighting within a virtual scene correctly and is critical when creating interactive applications, such as games for serious applications. The computation of such illumination is typically prohibitive for interactive or real-time performance if the visibility aspect of the indirect illumination is to be maintained. This paper presents a global illumination system which uses a structure termed the Approximate Visibility Grid (AVG) which enables interactive frame rates for multiple bounce indirect illumination for fully dynamic scenes on the GPU. The AVG is constructed each frame by making efficient use of the rasterisation pipeline. The AVG is then used to compute the visibility aspects of the light transport. We show how the AVG is used to traverse virtual point light sources in the context of instant radiosity, and demonstrate how our novel method enables interactive rendering of virtual scenes that require indirect illumination.**

I. Introduction

High-fidelity graphics are crucial if games and interactive serious applications are to accurately reproduce real environments. One of the major obstacles to overcome in achieving realistic high-fidelity environments within such applications, is the computation of images with multiple bounce indirect illumination within the interactive constraints required by games. As can be seen in Figure 1, global illumination can have a significant impact on the lighting in a scene. Light transport governs the physically-based computation of indirect illumination and is typically simulated by the rendering equation [1]. This simulation is recursive in nature and requires a large number of computational resources to compute. While CPU algorithms can compute global illumination in a more straightforward manner, parallelism is required to achieve interactive frame rates as shown by Wald et al. [2]. Using GPUs is often preferable due to the exponential advance in computing power on GPUs as opposed to CPUs and the benefit of doing all of the graphics computation on the GPU, freeing up CPU resources.

In this paper we present a method for computing indirect illumination at interactive rates on the GPU with fully dynamic lighting, geometry and camera. This supports multiple-bounce indirect illumination with no pre-computation. For each frame the entire illumination including indirect computation is computed from the ground up. We achieve this by introducing a rendering system which relies upon a new acceleration structure: the Approximate Visibility Grid (AVG). The AVG is constructed on the GPU and its role is to approximate the geometry in the scene. It is generated at the start of every frame, and is subsequently used in the calculation of the diffuse (or glossy) indirect component of the illumination.

In our system we demonstrate an application of the AVG to the Instant Radiosity [3] algorithm. The shooting of Instant Radiosity's Virtual Point Lights (VPLs) is accelerated using the AVG. While we demonstrate the potential of using instant radiosity for the computation of direct and indirect lighting, other methods of global illumination, such as distributed ray tracing [4] can also be used in conjunction with AVG.

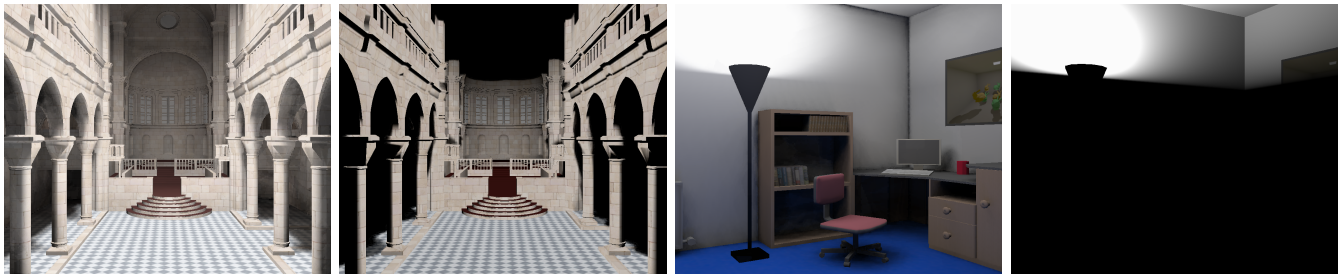
The major contribution of our work is the AVG which is used to accelerate global illumination computation to interactive or real time rates for interactive applications. The AVG is a novel data structure for improving the efficiency of the visibility computation by discretising the geometry and enabling fast traversal on the GPU. This enables an improvement in the realism of virtual scenes for applications such as serious games.

II. Related Work

In recent years, there have been several methods proposed for real-time global illumination techniques on the GPU. Many CPU-based approaches for interactive rendering via interactive ray tracing have been proposed recently, however we will not cover such methods (see Wald et al. [5] for a review) but focus on GPU-based techniques only.

Reflective shadow maps for global illumination were shown by Dachsbacher et al. [6]. This adapted the traditional shadow mapping idea to use certain texels from the shadow map, via a pre-computed sampling pattern, as light sources. This enabled single bounce indirect diffuse lighting at an interactive frame rate at the expense of a lack of visibility calculations for the secondary light sources. This idea was extended by Dachsbacher et al. [7], which used bounding geometry to represent the area covered by light sources. This allowed for both diffuse and glossy surfaces. The sampling pattern used in reflective shadow maps [6] was altered to use importance sampling, which ensured a more optimal distribution of lights, especially in scenes with non-diffuse surfaces. Nichols et al. [8] used a similar method, and hierarchical splatting to draw the light contribution to the screen, a technique they extended in [9].

Coherent surface shadow maps were used in [10] to compute shadow maps for points on surfaces in the scene, which were compressed into an atlas. They use a GPU based implementation of light cuts and hierarchical radiosity to compute



(a) Sibenik with GI (b) Sibenik without GI (c) Office with GI (d) Office without GI

Fig. 1. Scenes with and without Global Illumination (GI).

inter-reflections. These approaches all have the disadvantage inherent from a shadow map based multiple bounce approach; each bounce of VPL traversal step requires a separate shadow map. Imperfect shadow maps by Ritschel et al. [11] is similar to [7]; except they use low resolution shadow maps consisting of a point based representation of triangles randomly selected from the scene to more accurately take into account indirect diffuse illumination. This work requires a pre-processing step on the scene. The AVG can be used in conjunction with imperfect shadow maps as is mentioned in Section III-C.

Hierarchial radiosity approaches have been used by Dong et al. [12] who created a hierarchy of links between surface elements in the scene and used implicit visibility to evaluate this structure. Another approach, as proposed by Dachsbacher et al. [13], is to use a technique similar to hierarchial radiosity, but that does not need visibility queries. They showed this technique could be mapped to the GPU, but requires the scene to be highly tesselated to be effective in dynamic scenes.

Another approach for real-time global illumination is pre-computed radiance transfer introduced by Sloan et al.[14]. This method interpolates incoming illumination stored at vertices as low order spherical harmonics coefficients. However, due to a large pre-processing step, scenes are limited to being static, or allow only rigid movement of objects such as Wang et al. [15]. For deformable objects, one method is to use blockers [16], which are a very simple representation of the scene, often via the use of multiple spherical objects to represent the models in the scene.

Other approaches have been proposed recently which accelerate the Photon Mapping algorithm [17] to an interactive speed. Wang et al. [18] use cuts and clustered photons in order to render scenes at interactive rates. Fabianowski et al. [19] proposed an interactive GPU implementation of the global photon map with photon differentials. Photon mapping was used to compute indirect illumination.

An image space approach was proposed by Ritschel et al. [20] which extended screen space ambient occlusion techniques to cater for indirect lighting. As this method operates in screen space, the light transport simulation only operates in visible areas, and therefore does not account for information from the rest of the scene.

Our algorithm implements the Instant Radiosity method by Keller [3]. Instant Radiosity solves the rendering equation [1] via a quasi random walk which distributes VPLs around the scene. Shadow maps for each of the lights are then efficiently rendered on the GPU, and the contribution for each light is accumulated into a final image. Laine et al. [21] used an iterative update system to only update relevant VPLs, assuming the light source is moving smoothly. It updates a subset of VPLs based on removing lights that have been invalidated by the move, and randomly adding new VPLs. Lights can only bounce off static geometry, and dynamic parts of the scene are restricted to illumination solely from static geometry. Kaplanyan et al. [22] have recently proposed a similar approach to ours in that they use a grid for light propagation and achieve real-time frame rates, however their system limits the occlusion of secondary light sources to directly visible geometry.

III. Algorithm

In this section we give an overview of our algorithm and the distinct steps involved. This algorithm is composed of three stages. The initial stage involves the construction of the AVG data structure. This is a coarse voxelisation of a tesselated scene, where each voxel stores the average reflectance and normal information of the triangles within the voxel. This is used to accelerate the second stage, where VPLs are traced through the scene via the AVG. At each intersection with the AVG, a VPL is placed into a list, and further rays are spawned to calculate multiple bounces on indirect lighting. This is used in the final stage, to calculate the contribution of the VPLs onto the parts of the scene visible from the camera in order to form the final image. The subsequent sections provide a detailed description of this process.

A. Constructing the AVG

For the construction of the AVG, we leverage the computational power of modern GPUs to quickly discretise the scene. The AVG represents the scene by a low resolution 3D grid (stored as a 3D texture on the GPU), where voxels in the grid contain information about the discretised geometry; the normal and colour of the surfaces contained in the voxel. We use a similar process to Dong et al. [23] and Eisemann et al. [24]. However our approach differs as we have to store three component colour and two or three component normal information in each voxel, so we have to use a different

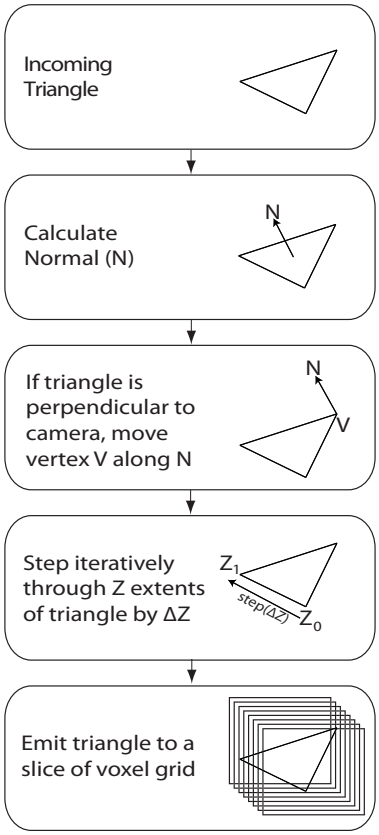


Fig. 2. Highlighting the important steps in the construction of the Approximate Visibility Grid data structure.

method. Figure 2 demonstrates the major steps required for the AVG construction. This process is further explained below.

AVG construction uses an orthographic projection to transform the triangles from world space into texture space. For each triangle, the first stage calculates the plane of the incoming triangle via a cross product. This serves two purposes, to calculate the normal of the triangle for the voxels in the AVG and to test whether the triangle is perpendicular or close to being perpendicular to the camera's plane. This test is required as when they are close to perpendicular to the camera the rasteriser is not able to interpolate across pixels, so nothing is drawn. This is quite a common scenario, which occurs when geometry is aligned with the axis in which the camera points or when the geometry is higher resolution than the grid. We alleviate this by shifting the vertex at the maximum depth value by a small amount (ε) along the triangle normal to ensure it is rasterised:

1: **if** abs(trianglenormal.z) $<$ (1 / gridwidth) **then**
2: trianglevert.maxZ += N * ε;
3: **end if**

Note this does not affect the stored normal. The next step calculates which voxels of the AVG contain each triangle. The rasterisation unit is utilised for this purpose. Firstly, the camera for the orthographic projection is aligned to point down the Z axis, and so each slice of the AVG represents a 3D slice of world space of depth $\triangle$Z along the Z axis. The first Z slice of the current triangle is found:

$$\text{Zslice} = \min(v1.z, v2.z, v3.z) - \text{Camera.z}$$

where $v1, v2$, and $v3$ are vertices belonging to the triangle, and the triangle is drawn to the relevant voxel within the slice. The computation then steps through the rest of the triangle in steps of $\triangle$Z, each time writing the triangle information (normal and albedo) into a slice of the AVG. If the area of a triangle is smaller than a predefined value (in our implementation it is set to be the area of one of the faces of the voxel), then a triangle which is large enough, and is guaranteed to be rasterised is drawn in the place of the smaller triangle. Multiple larger triangles may have to be used when a smaller triangle overlaps more than one voxel. This allows triangles that would otherwise not be rasterised to be accumulated into the AVG. In all the scenes used in this paper, this method completely encloses the scene geometry, thereby preventing light leaking in the tracing pass.

Sometimes more than one triangle will be written to the same voxel. One strategy is to overwrite the contents of the voxel with each new triangle. However, this has drawbacks such as the information from a small triangle overwriting that of a large one, which can result in incorrect distribution of the VPLs. Another approach, which is the one we use, is to sum the contributions of each triangle, and then normalise in the tracing stage. This uses additive blending when writing the triangles to the AVG. Another approach is to weight the contribution of each triangle based on the surface area of the slice of the triangle being written to the voxel.

As stated before, each voxel of the AVG contains a normal which is used to represent the geometry, and the colour representing the material colour. If triangles are textured, the colour stored in the lowest mip-map level of the texture is used to prevent high frequency details from the texture affecting the whole voxel. A mip-map level related to the size of a voxel projected into texture space can also be used for the same purpose, but may give more detail in some scenes.

Our implementation of the AVG employs 3D textures to store the AVG data structure. Two 3D textures are used, one to store the normals and another the colours. The 3D textures we use have a default resolution of $64 \times 64 \times 64$, however they can be any size to more closely suit the geometry of the scene. The computation of the depth slices onto the AVG is performed in the geometry shader. On current hardware the geometry shader performance decreases based on the number of triangles emitted, so our implementation uses multiple passes each writing to a single slice, see Figure 2. If the scene consists of triangles with a small Z depth (i.e. if all are written to one slice of the AVG), the performance is optimal, and decreases based on the Z depth of the triangles in the scene.

B. Tracing

This section covers shooting of VPLs into the scene. Figure 3 illustrates this process. As stated before, it is based on instant radiosity, so rays start at the light sources and propagate light

around the scene until a termination condition is reached. At each intersection point, a VPL is placed, which is used to light the scene in a subsequent pass.

The starting points and the directions of the rays are generated using cosine sampling over the area of the light source. This uses a quasi-random sequence to ensure the same distribution of VPLs between frames, as this eliminates the major source of flickering in dynamic scenes. If there are multiple light sources, then the starting points can be split between the light sources. For instance if there are N_{rays} starting rays and N_{lights} light sources, then N_{rays}/N_{lights} can be assigned to each light source. Other strategies can be used as well, for instance the number of samples could be weighted based on the power of the light source.

The AVG is used to facilitate the VPL traversal step. The traversal stage takes the form of a GPU program consisting of a geometry shader which accepts rays starting at the lights as an input, and emits VPLs. Once a ray is sent to the geometry shader, the traversal of the scene begins. As there is currently no support for recursion on the GPU, we implement this in an iterative manner. This loop can be terminated by a user controlled value or a term based on the average diffuse reflectance of the scene or through Russian Roulette. Rays are traced through the AVG using ray marching.

At each step, the normal information is sampled from the AVG, and if there is information present, then we assume a surface has been hit, and stop marching the current ray. The colour value is also sampled from the AVG at the intersection point. We use this to calculate the colour of the VPL. The VPL information (position, direction and colour) is then emitted from the geometry shader. The position of the VPL is pushed a random distance into the voxel, as if VPLs are deposited on the boundaries of voxels, the conservative nature of the voxelisation process will show.

The next step is to select an outgoing direction of the ray from the intersection point. This uses cosine sampling of the hemisphere, and the results of this are transformed into world space. This is calculated in spherical coordinates as:

$$(\theta, \phi) = (\sqrt{\cos(\xi_1)}, 2\pi\xi_2)$$

where the quasi-random numbers $\xi_1 \subseteq [0, 1)$ and $\xi_2 \subseteq [0, 1)$ are designated by a texture lookup in a pre-computed texture. This texture contains quasi-random numbers generated from a low discrepancy series on the CPU. This frees the GPU from calculating the quasi-random numbers each time they are required in order to avoid unnecessary extra computation in the shader. The outgoing ray is then weighted by the probability that the direction was selected. This process continues until the required number of bounces has been reached or the path terminates.

C. Shading and Interleaved Sampling

The shading stage computes the illumination per pixel. This is carried out using deferred shading and interleaved sampling [25]. First, a G-buffer [26], [27] is computed, which stores a full resolution copy of the world space positions, normals, and colours of the scene. We use the G-Buffer to minimise the shading costs per pixel, and as a further enhancement, the G-buffer is split into M subsets (in our case 16), using a single pass swizzle operation, similar to Segovia et al. [28]. This is to improve texture access time when computing subsequent stages, as coherent texture accesses have been shown to improve performance.

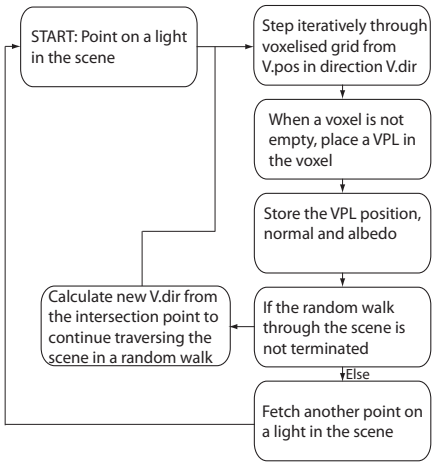


Fig. 3. Path continuation and termination heuristics for tracing VPLs used in Instant Radiosity through the Approximate Visibility Grid.

When shading each pixel, the lighting between each VPL and the pixel in the G-Buffer is calculated. To achieve this, we have to solve the rendering equation:

$$L(x \to x') = L_e(x \to x') + \int_A f(x'' \to x \to x') L(x'' \to x) G(x \leftrightarrow x'') dA$$

where L is the radiance transmitted between two points, x, x', and x'' are points in the scene, f is the Bi-directional Reflectance Distribution Function (BRDF), and G is the geometry term defined as:

$$G(x \leftrightarrow x') = V(x \leftrightarrow x') \frac{(\vec{N_x} \cdot \vec{w})(\vec{N_{x'}} \cdot -\vec{w})}{|x - x'|^2}$$

and N_x is the surface normal at the point x, $\vec{w}$ is the normalised direction vector from $\vec{x}$ to $\vec{x'}$, and V is the visibility between two points. The visibility term is evaluated via shadow maps for visibility between points in the scene, similar to Instant Radiosity [3]. As this process has to happen at interactive rates, we use a low resolution shadow map for each VPL, all of which are stored in a large texture. We use instancing to decrease the number of draw calls required to create the shadow maps. We can speed the process further by using lower level of detail geometry for the shadow maps, or by using Imperfect Shadow Maps [11].

Once these shadow maps are computed, indirect lighting is computed per pixel. This works by assigning a subset of size (N_{VPLs}/M) VPLs to use when shading the pixel, where N_{VPLs} is the total number of VPLs and M is the total

(a) Split G-buffer (b) Shadow texture (c) Shading performed on subsets (d) Final result

Fig. 4. An illustration of the shading pipeline for the Sibenik scene. The G-Buffer is first split into subsections, then shadow maps for the VPLs are computed and stored in a large texture. The pixels in each subset are then shaded from the relevant subset of VPLs, and finally the image is reconstructed.

number of subsets. The shader loops over the subset of VPLs associated with the pixel, samples the shadow map associated with each VPL from the shadow map texture, and uses this to compute whether the pixel is visible from the VPL. If the pixel is visible, then the light is attenuated based on the BRDF f and the geometry term G, the latter of which is calculated using world space positions and normals from the G-buffer and the VPL information.

The full image containing indirect lighting is then re-assembled, using a single pass swizzle operation. The image is then filtered to remove the structured noise implicit from utilising interleaved sampling. This uses a 4×4 box filter and also calculates discontinuities on the fly. This is computed via a dot product of the normals of the texels being sampled, and the length in world space between the samples. When a texel is found to be on a discontinuity, its contribution is ignored and the filter is re-weighted to take this into account. Finally, this is multiplied by the material colour of the original surface to give the output image. This process is illustrated in Figure 4, and the contribution of the indirect lighting to the final image is shown in Figure 5.

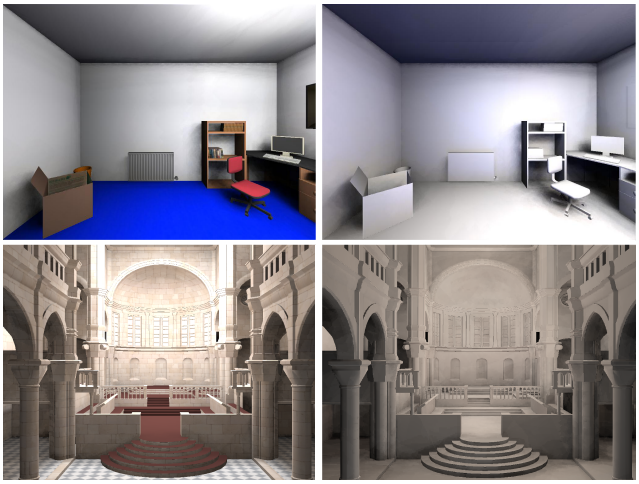


Fig. 5. This shows the final image (left) vs. indirect lighting only (right) for the Office and Sibenik scenes.

IV. Results

We demonstrate the performance of this approach with a set of results taken from the scenes described below, shown in Figure 8. The frame rate per second for each scene is shown adjacent to the name of the scene. All of the results were generated whilst using this system specification: Nvidia 8800GTX GPU, 2GB RAM and a Q6850 Extreme 3GHz CPU. 1024 VPLs were utilised for each of our test scenes. The AVG consisted of two 3D textures both of dimension: $64 \times 64 \times 64$. We used a Lambertian shader for rendering materials in each scene, however this can be extended to use other BRDFs. In our approach, we use a maximum of three bounces. For all the scenes demonstrated, the entire computation was carried out each frame. We use paraboloid shadow maps for scenes containing smaller triangles, and traditional shadow maps for scenes containing larger triangles to avoid the warping present in paraboloid shadow maps in this type of scene. This can lead to artifacts due the lack of visibility information at the edges of the shadows due to field of view specified, however this is not too noticeable due to the reduction in radiance from the cosine from the light source in the geometry term.

The first scene consists of a traditional Cornell Box 8(a) with an animated elephant in the center of the scene highlighting support for dynamic geometry. For Sibenik 8(b) we use solely static geometry, with the light source emitting downwards. We use a lower level of detail representation of the geometry for Sibenik for voxelisation and shadow computation (7000 triangles). This optimisation gave around a 4 times speed up over rendering shadow maps when compared with the original model with over 80000 triangles. For the Office 8(c) we have static geometry again, however we also have a light source (lamp) that is dynamic within the environment. Image 8(d), the U-Room, highlights the indirect illumination, as the right-hand portion of this image is entirely lit indirectly. Image 8(e) is another view of the Sibenik cathedral, with the light source located in the dome, however in this case it is emitting upwards. Scene 8(f) shows the Happy Buddha inside a Cornell Box with the light source pointing upwards so most of the scene is indirectly lit.

Table I shows the timings of the different stages of the pipeline. All timings were taken with the same initial settings.

(a) Instant Radiosity using the AVG to trace VPLs

(b) Instant Radiosity CPU VPL tracing Reference

Fig. 6. Comparison of the results of VPL tracing running on the GPU via the AVG 6(a), and a reference image with the VPLs traced on the CPU 6(b).

(a) 256 VPLs

(b) 1024 VPLs

Fig. 7. Comparison of a view of Sibenik rendered with 256 VPLs 7(a), and 1024 VPLs 7(b).

As can be seen from the table, the times for the voxelisation stage (the construction of the AVG) are in the order of a couple of milliseconds for moderately complicated scenes. The traversal stage is also quite short, with the majority of the time taken by traversal of incoherent rays through the AVG, leading to a random access pattern in the 3D texture which stored the AVG. The vast majority of the time taken to render the scene is taken with the calculation of the shadow maps for each VPL. We also present timings for the scenes rendered without visibility for the VPLs, which are real-time for all scenes. As expected the sequence of steps to compute the shading for each pixel (G-Buffer creation, G-Buffer splitting, VPL contributions and re-assembling the image with discontinuities) is also quite expensive, but roughly constant throughout all scenes.

Figure 6 shows a comparison between the results of VPLs being traced via the AVG on the GPU (6(a)), and on the CPU (6(b)). Figure 6(a) was calculated at 9 FPS, while Figure 6(b) requires a rebuild of the acceleration structure on the CPU every time there is an alteration in the scene geometry. The CPU version has a slightly better distribution of VPLs due to actual scene geometry being used, rather than the discretised version in the AVG. However, as can be seen from the figure, the lighting in the images is similar.

We also tested the algorithm with 256 VPLs, which is a figure that has been commonly used by other instant radiosity methods [21]. This ran at an average frame rate of 14 FPS, although the quality of the visibility calculations was reduced, as is shown in Figure 7. We found that changing the dimensions of the AVG (from 64^3 to 128^3) made no difference to the frame rate or image quality for the scenes in this paper.

Resolution	Office FPS	Sibenik FPS
640×480	7.9	10.2
800×600	7.5	9.9
1024×768	7.2	8.9
1280×1024	6.9	8.1
1920×1080	6.7	7.2
1920×1200	5.5	6.9

TABLE II
FRAMES PER SECOND TIMINGS FOR THE DIFFERENT TEST RESOLUTIONS FOR THE OFFICE AND SIBENIK SCENES.

V. CONCLUSION AND FUTURE WORK

In this paper we have introduced a new technique for accelerating the computation of multiple-bounce indirect il-

Scene	Triangles	Vox	Tra	Sha	Ill	FPS (NV)	FPS (V)
U Room	156	0.4	1.8	19.9	19.1	62.1	22.3
Elephant	2113	1.4	1.9	28.6	19.7	61.8	21.1
Office	28218	2.2	2.3	112.3	21.4	23.7	7.2
Sibenik	80062	3.2	2.2	80.6	19.7	45.5	9.1

TABLE I

TIMINGS FOR THE SCENES. **VOX** ARE THE TIMES FOR THE VOXELISATION STAGE, **TRA** THE TIMES FOR TRACING OF THE VPLS, **SHA** STANDS FOR THE SHADOW MAP COMPUTATION TIME AND **ILL** IS THE TIMES FOR THE OTHER STAGES (G-BUFFER CREATION, ILLUMINATION AND FILTERING). ALL TIMES ARE IN MS. ALSO INCLUDED IN THE TABLE ARE THE NUMBER OF TRIANGLES IN THE SCENE, AND THE FRAMES PER SECOND WITHOUT VISIBILITY FROM THE VPLS **FPS (NV)** AND WITH VISIBILITY FROM VPLS **FPS (V)** AT A 1024×768 RESOLUTION.

lumination, in its entirety per frame, for fully dynamic scenes enabling interactive applications to use high-fidelity graphics. This makes it possible, as this technology improves, to begin creating serious applications with a high graphics fidelity. We demonstrated how the AVG can be used in conjunction with multiple bounce instant radiosity to accelerate the VPL shooting stage. This acceleration applies to static geometry as well as dynamic, as temporal variations do not affect performance, and calculations pertaining to the illumination are recomputed upon each successive frame.

VI. ACKNOWLEDGEMENTS

We would like to thank Marko Dabrovic for the Sibenik model, Stanford Computer Graphics Laboratory for the Happy Buddha model, and Vedad Hulusić for the office scene. The work was partially supported by the project EP/I006192/1.

REFERENCES

[1] J. T. Kajiya, "The rendering equation," in *Computer Graphics*, 1986, pp. 143–150.

[2] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek, "Interactive Global Illumination using Fast Ray Tracing," in *13th EUROGRAPHICS Workshop on Rendering*, Pisa, Italy, 2002.

[3] A. Keller, "Instant radiosity," in *Computer Graphics*, 1997, pp. 49–56.

[4] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1984, pp. 137–145.

[5] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek, "Interactive global illumination using fast ray tracing," in *Proceedings of the 13th Eurographics workshop on Rendering*. Eurographics Association Aire-la-Ville, Switzerland, Switzerland, 2002, pp. 15–24.

[6] C. Dachsbacher and M. Stamminger, "Reflective shadow maps," in *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*. New York, NY, USA: ACM, 2005, pp. 203–231.

[7] ——, "Splatting indirect illumination," in *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*. New York, NY, USA: ACM, 2006, pp. 93–100.

[8] G. Nichols and C. Wyman, "Multiresolution splatting for indirect illumination," in *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*. New York, NY, USA: ACM, 2009, pp. 83–90.

[9] G. Nichols, J. Shopf, and C. Wyman, "Hierarchical image-space radiosity for interactive global illumination," in *Proceedings of the 19th Eurographics Workshop on Rendering*, 2009.

[10] T. Ritschel, T. Grosch, J. Kautz, and H.-P. Seidel, "Interactive global illumination based on coherent surface shadow maps," in *GI '08: Proceedings of graphics interface 2008*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2008, pp. 185–192.

[11] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz, "Imperfect shadow maps for efficient computation of indirect illumination," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 1–8, 2008.

[12] Z. Dong, J. Kautz, C. Theobalt, and H.-P. Seidel, "Interactive global illumination using implicit visibility," in *Pacific Conference on Computer Graphics and Applications*, 2007, pp. 77–86.

[13] C. Dachsbacher, M. Stamminger, G. Drettakis, and F. Durand, "Implicit visibility and antiradiance for interactive global illumination," *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, vol. 26, no. 3, August 2007.

[14] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," in *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2002, pp. 527–536.

[15] R. Wang, J. Zhu, and G. Humphreys, "Precomputed radiance transfer forreal-time indirect lighting using a spectral mesh basis," in *Eurographics Symposium on Rendering*, 2007, pp. 23–34.

[16] P.-P. J. Sloan, N. K. Govindaraju, D. Nowrouzezahrai, and J. Snyder, "Image-based proxy accumulation for real-time soft global illumination." in *Pacific Conference on Computer Graphics and Applications*, M. Alexa, S. J. Gortler, and T. Ju, Eds. IEEE Computer Society, 2007, pp. 97–105.

[17] J. Wann, "Realistic Image Synthesis Using Photon Mapping," *AK Peters Natick, Massachusetts*, 2001.

[18] R. Wang, K. Zhou, M. Pan, and H. Bao, "An efficient GPU-based approach for interactive global illumination," in *ACM SIGGRAPH 2009 papers*. ACM, 2009, p. 91.

[19] B. Fabianowski and J. Dingliana, "Interactive global photon mapping," *Computer Graphics Forum*, vol. 28, no. 4, pp. 1151–1159, 2009.

[20] T. Ritschel, Tobiasand Grosch and H.-P. Seidel, "Approximating dynamic global illumination in image space," in *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Association for Computing Machinery (ACM). Boston, MA, USA: ACM, February 2009, pp. 75–82.

[21] S. Laine, H. Saransaari, J. Kontkanen, J. Lehtinen, and T. Aila, "Incremental instant radiosity for real-time indirect illumination," in *Proceedings of Eurographics Symposium on Rendering 2007*. Eurographics Association, 2007, pp. 277–286.

[22] A. Kaplanyan and C. Dachsbacher, "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*. New York, NY, USA: ACM, 2010, pp. 99–107.

[23] Z. Dong, W. Chen, H. Bao, H. Zhang, and Q. Peng, "Real-time voxelization for complex polygonal models," in *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 43–50.

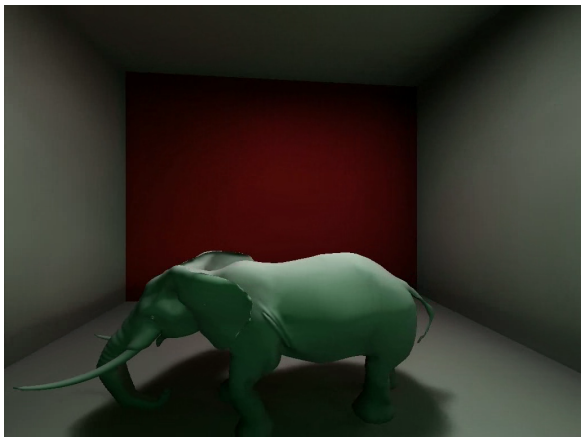
[24] E. Eisemann and X. Décoret, "Fast scene voxelization and applications," in *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. ACM, 2006, p. 78.

[25] A. Keller, E. Keller, and W. Heidrich, "Interleaved sampling," in *Rendering Techniques 2001 (Proc. 12th Eurographics Workshop on Rendering*. Springer, 2001, pp. 269–276.

[26] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The triangle processor and normal vector shader: a vlsi system for high performance graphics," in *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1988, pp. 21–30.

[27] T. Saito and T. Takahashi, "Comprehensible rendering of 3-d shapes," *SIGGRAPH Comput. Graph.*, vol. 24, no. 4, pp. 197–206, 1990.

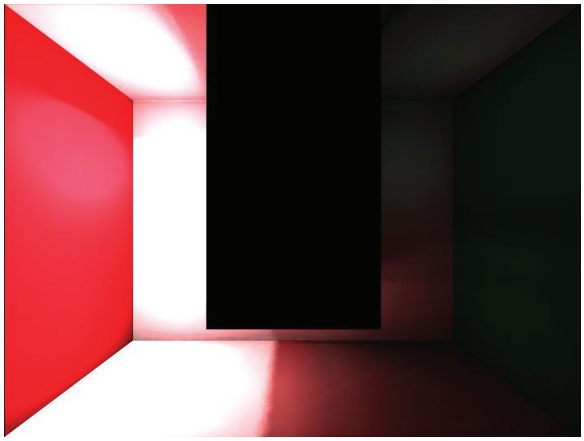
[28] B. Segovia, J.-C. Iehl, and B. Proche, "Non-interleaved Deferred Shading of Interleaved Sample Patterns," in *Eurographics/SIGGRAPH Workshop on Graphics Hardware '06*, Sep. 2006, pp. 53–60.



(a) Elephant, 21fps

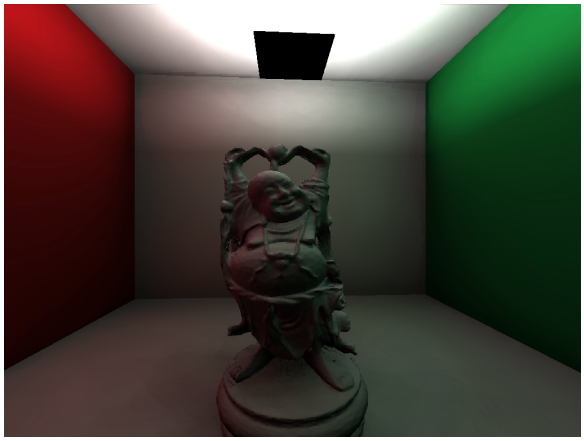
(b) Sibenik, 9fps

(c) Office, 7fps



(d) U Room, 22fps

(e) Sibenik, 9fps



(f) Happy Buddha, 13fps

Fig. 8. The scenes used for our results: 8(a) Elephant animated in the Cornell Box showing green colour bleeding onto the Elephant model 8(b) The Sibenik cathedral model with the light source located in the dome 8(c) The Office scene showing multiple bounce effects (note the blue colour on the ceiling requiring two bounces) 8(d) The U shaped room again illustrating multiple bounce effects such as the green colour bleeding in the indirectly lit area 8(e) Sibenik with the light source located in the dome, but emitting upwards so that a minimum of two indirect bounces are required to light this portion of the scene 8(f) Happy Buddha in Cornell Box, with light source pointing upwards showing colour bleeding from multiple indirect bounces. All images were rendered at 1024×768.