Adoption of Sparse 3D Textures for Voxel Cone Tracing in Real Time Global Illumination

Igor Aherne, Richard Davison, Gary Ushaw and Graham Morgan
School of Computing, Newcastle University, Newcastle upon Tyne, U.K.

Keywords: Global Illumination, Voxels, Real-time Rendering, Lighting.

Abstract: The enhancement of 3D scenes using indirect illumination brings increased realism. As indirect illumination is computationally expensive, significant research effort has been made in lowering resource requirements while maintaining fidelity. State-of-the-art approaches, such as voxel cone tracing, exploit the parallel nature of the GPU to achieve real-time solutions. However, such approaches require bespoke GPU code which is not tightly aligned to the graphics pipeline in hardware. This results in a reduced ability to leverage the latest dedicated GPU hardware graphics techniques. In this paper we present a solution that utilises GPU supported sparse 3D texture maps. In doing so we provide an engineered solution that is more integrated with the latest GPU hardware than existing approaches to indirect illumination. We demonstrate that our approach not only provides a more optimal solution, but will benefit from the planned future enhancements of sparse 3D texture support expected in GPU development.

1 INTRODUCTION

The realism of a rendered scene is greatly enhanced by indirect illumination - i.e. the reflection of light from one surface in the scene to another. Achieving believable looking indirect illumination in real-time remains a challenging prospect, as each rendered fragment of a surface must integrate light that has fallen on it from any direction, and that light may have already been reflected from other surfaces within the environment. In turn, the light reflected from those secondary surfaces must also have been integrated from the light arriving from multiple directions to the secondary surface. Clearly the computational complexity of the problem rises exponentially with the number of light bounces considered. Further to this, light that has been reflected from surfaces which are outside the viewing frustum can still affect rendered fragments. Consequently a technique for indirect illumination must combine knowledge of the fragments to be rendered to the screen with a model of the 3D environment for determining visibility between arbitrary points in the scene.

A recently adopted approach to real-time indirect lighting is the use of voxel cone tracing (Crassin et al., 2011). The approach has received a lot of attention and commercial usage in high-end video game engines (Mittring, 2012) as it computes two light-bounces with minimal reliance on the specifics of the original 3D mesh in the calculations (thus achieving desirable frame-rates almost independent of the complexity of the scene). The approach entails representing the scene as a hierarchical voxel structure and storing it, on the GPU, in a dynamic sparse octree. The octree is constructed for the scene on initialisation, and is then modified as the dynamic parts of the scene move. The voxel cone tracing technique is then used to estimate visibility of scene elements for calculation of indirect illumination.

Sparse 3D texture mapping offers an alternative approach for storing the hierarchical voxel representation of the scene. Sparse 3D textures provide a structure which is more suitable to the random accesses required by the dynamic hierarchical voxel representation than that provided by a sparse octree structure. Hardware interpolation is also available for highly efficient trilinear and bilinear interpolation of lighting values stored in the voxelised texture at minimal software cost. Further to this, a 3D texture may allow for a larger scene to be represented, as it is not as constrained by GPU memory requirements as the sparse octree. However, as the octree can be stored on the GPU, access is very fast. Current graphics hardware and SDKs are increasingly incorporating efficient 3D texture support into their design, so the speed of access is improving with each generation of graphics

Aherne, I., Davison, R., Ushaw, G. and Morgan, G.
Adoption of Sparse 3D Textures for Voxel Cone Tracing in Real Time Global Illumination.
DOI: 10.5220/0008874502010209
In *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2020) - Volume 1: GRAPP*, pages 201-209
ISBN: 978-989-758-402-2; ISSN: 2184-4321

pipeline.

In this paper we present an implementation of voxel cone tracing for global illumination which utilises a sparse 3D texture to store the hierarchical voxel representation of the scene. We also incorporate some further performance enhancements used in our engineered solution, including the use of compute shaders to create the mip-chain. We present performance figures and rendered imagery which are favourably comparable to the reference solution. We further argue that, as graphics hardware focuses more on utilising 3D textures, the advantages of random access to such a structure over a sparse octree, coupled with the potential for representation of larger scenes, and further utility of hardware-implemented interpolation, will see this approach increasingly adopted.

2 BACKGROUND AND RELATED WORK

2.1 Real Time Global illumination

While techniques for real-time rendering are improving rapidly, they inevitably lag behind the realism and complexity of graphical fidelity that is achievable through off-line computation. Interactive media such as video gaming allow the user to control the camera in the 3D scene so calculation of the light that reaches the camera must occur dynamically, often with the aim of achieving 60fps on relatively low power hardware. Off-line techniques for convincing indirect illumination, such as ray-tracing (Kajiya, 1986) and photon-mapping (Jensen, 2001), are well established, and widely used in the film industry etc.

The use of reflective shadow maps (RSM) was an early solution to modelling indirect illumination of surfaces (Dachsbacher and Stamminger, 2005). Each light source is allocated a shadow map, which determines the parts of the scene that are illuminated by the light (i.e. can be seen by the light). A depth value is stored relative to the light source, and is then used to determine whether a rendered point in the final frustum was part of the shadow map allocated to each light source. The computation cost and the memory required for shadow map textures increases with each additional light source, and only single bounce illumination is provided.

Increased efficiency in lighting calculation can be achieved through the application of algorithms in screen-space (Nichols et al., 2009). After the pixels that are displayed in the frustum are determined, a post-processing stage performs further lighting calculations based only on the information contained within those rendered pixels. The obvious disadvantage here is that any light reflected from surfaces outside the frustum is discarded. Such algorithms trade this inaccuracy off against the computational savings afforded by the post-processing step only considering those pixels that will definitely be rendered (Ritschel et al., 2012). Screen space techniques have been successfully adopted on relatively low power games consoles for simplified indirect illumination in the form of screen-space ambient occlusion (Goulding et al., 2012).

Layered global illumination (Hachisuka, 2005) simplifies the visible geometry to be considered for lighting calculation by introducing a series of depth-peeled layers, extending from a number of fixed viewpoints, and rendered orthographically. The fixed points form a spherical distribution to afford better coverage of global illumination. As the environment is rendered orthographically, consecutive layers along a particular direction can be used to represent the transition of light between one layer and the next. Since each layer is a two-dimensional image, rasterisation is readily employed, taking advantage of the parallel nature of GPU computation. A computational expense of the layered global illumination approach is the requirement to project all the visible pixels onto each layer every time it is created. Furthermore the size of the texture required to store the layered projections scales with the size of the environment, as the whole entity has to be rasterised at an appropriate resolution for each layer and from various viewing directions. The technique results in a single bounce of light; further light bounces requiring the entire procedure to be repeated.

Scalable solutions to real-time illumination from multiple light sources are described in (Dachsbacher et al., 2014). Scalability is a key factor here with results trading off visual fidelity against computation time, ranging from almost real-time to tens of minutes per rendered frame. Cascaded light propagation volumes (Kaplanyan and Dachsbacher, 2010) have also been utilised to provide a scalable solution to indirect illumination. The approach uses lattices and spherical harmonics to represent the spatial and angular distribution of light in the rendered scene. The initial work showed viable performance for single bounce illumination. The efficiency for multiple bounces has been addressed with solutions including selective light updates (Sundén and Ropinski, 2015), and partial precomputation (Jendersie et al., 2016).

2.2 Voxel Cone Tracing

A technique for calculating indirect illumination must rely on knowledge of the spacial relationship between surfaces in the proximity of the viewing frustum. Voxelisation has been employed successfully to represent this three-dimensional spatial information. The voxel cone tracing technique described in (Crassin et al., 2011) is the basis for the work presented in this paper and provides our reference solution.

The geometry of the scene is rasterized into voxels, being stored in a data structure containing colours and normals in the form of a sparse octree (Laine and Karras, 2011). The scene's illumination is then injected into the resulting voxels and is stored in a second structure which contains the lighting information, particularly the incoming radiance. If the implementation stores the reflected light value instead (for simpler computations, but losing specular highlights), it is referred to as the *diffuse*. The lighting is then filtered upwards within the structure (equivalent to mipmapping of a texture). Coherence within the nearby environment is maintained via the 3D structures. Any voxel in such a dataset can access the information about any other voxels, including those outside the viewing frustum or obstructed by other voxels.

The secondary lighting is then computed. For any voxel in the data structure, the light arrives from the hemisphere of directions around the surface normal. The approach in (Crassin et al., 2011) estimates this volume with a number of cones. Each cone can be thought of as a number of cubes stacked one after another with increasing size. As each cube carries information (eg colour), the parameter at any point in the cone volume can be approximated by linear interpolation between the two nearest voxels. To accumulate the incoming light for each of the voxels, the algorithm marches along each of the cones, sampling the corresponding voxels in each one. As the distance travelled along the direction increases, the lower-resolution data structure is sampled. The secondary light is computed for all the voxels within the structure, resulting in a single-bounce global illumination.

When all the light is filtered and the first bounce is calculated, any final fragment at the end of the deferred rendering pipeline can be located within the data structure and assigned its own corresponding, already existing, voxel. Rather than directly assuming the contents of that voxel, the fragment carries out another round of voxel cone tracing, gathering information from the previously computed output, acquiring indirect illumination in a further bounce.

The voxel data is stored in a sparse octree structure, which resides on the GPU in a linear buffer, and is updated when required. A highly compact version of an octree is utilised, which uses one pointer per node, pointing to a cluster of all children nodes. If the octree is small enough then it benefits significantly from GPU caching.

Some work has recently introduced a cascaded volume to represent the voxelised environment (McLaren and Yang, 2015). The work focuses on computational overhead for viability on the Playstation4 console, using a set of 3D texture volume cascades.

2.3 Sparse 3D Textures

With OpenGL 4.3 the concept of *sparse textures* was introduced. The desire for increased resolution and scale of readily available texture data has grown to the point where the amount of available graphics memory is the limiting factor. The default solution had previously been to page the texture memory (significantly decreasing performance) (Weiler et al., 2000). The introduction of sparse textures separated the address space of the graphics processor from the physical graphics memory, provisioning a partial virtualisation of the texture memory mapping and non-contiguous access to texture data.

Sparse texture access is also available for three dimensional textures. Efficiently creating sparse textures while preserving the fidelity of the source image is an ongoing research topic (Peyré, 2009). Approaches typically entail the identification of affine regions within the texture (Lazebnik et al., 2005) and contour identification for image segregation (Gao et al., 2013). As sparse texturing support is further incorporated into the rendering hardware pipeline it will become increasingly efficient and accessible to the graphics programmer.

Interpolation between samples of sparse texture data occurs in the hardware (as with bilinear and trilinear sampling of texture data for the fragment shader stage) (Shreiner et al., 2013). This provisions a highly efficient method for interpolating between sampled values in the sparse texture, and removes the necessity for the creation of bespoke software to achieve the required interpolation from data saved in a less hardware efficient manner.

3 IMPLEMENTATION

In this section the implementation details which are specific to the adoption of 3D sparse textures for the voxelisation of the scene are described. The broader

global illumination approach of voxel cone tracing is well-established (Crassin et al., 2011). The section starts with a justification for the use of 3D textures in this context, before detailing the implementation, and describing some optimisations.

3.1 Applicability of Sparse 3D Textures

A sparse 3D texture was selected for the hierarchical voxelisation of the scene, rather than the previously used octree. Sparse texture maps reduce the memory footprint of images with areas of unused space, so appear to be well suited to a hierarchical voxel structure. The contents of a sparse texture are loaded in chunks, only if they contain non-null data. If used to represent a voxelised space, chunks will only be uploaded if they contain a non-uniform space.

While there is potential for a reduced memory footprint, there is a caveat. Each chunk must be at least 64kb in size (this constraint is likely to change as hardware evolves). In the voxel environment this means at least 26x26x26 blocks of volume sent each time (for an 8-bit RGBA texture). Even if there is a single stored entry and the rest of the voxels are null, the entire chunk has to be sent to the GPU. Consequently the viability of the approach will vary depending on the scene's geometry.

An additional benefit of using 3D textures is the availability of efficient hardware trilinear interpolation. This technique calculates the value within a cube volume, based on eight control values. To obtain the final result, two values are obtained via the two bilinear interpolations for the top and bottom plane, followed by computing the value in between them, with another linear interpolation. In the case of a dynamically changing octree laid-out linearly in the GPU memory, this would be hard to achieve, and would require bespoke code rather than have hardware support. Furthermore, estimating the values between the voxel volumes is achieved via quadrilinear interpolation (a further linear interpolation based on the pair of trilinear values fetched from each voxel).

As graphics hardware evolves, support for 3D textures is likely to increase, providing faster access to the texture data, more efficient hardware interpolation, scattered texture writing and larger high-speed memory space. Conversely, the sparse octree approach is likely to continue to require bespoke code tailored to the constraints of general GPU computing.

3.2 Voxelisation in Sparse 3D Textures

Scattered texture writing is a relatively recent addition to graphics SDKs (eg OpenGL 4.3 onwards). The technique enables a fragment shader to write to an arbitrary texel in a texture. Ordinarily, the use of frame-buffers entails binding each fragment to a target texel, with the correspondence of fragment to texel pre-determined before the fragment shader is run. Programmable texel access allows us to write voxel details dynamically to the 3D texture map.

Without scattered texture writes the program would have to select a single layer of the texture volume into which a fragment will be rasterized. In our implementation layers are ordered along the z-axis, hence if a triangle extends along the z-axis, many duplicates may be generated during the geometry shader stage. A unique layer would then be selected for each generated triangle. However there is a limit to how many vertices can be generated from any incoming primitive during the geometry shader stage. In addition, the more information that is associated to a vertex and has to be carried into the fragment stage, the less the number of new vertices that can be created. The approach was worthwhile only with relatively small triangle pieces that could extend to twenty layers in the worst case scenario (meaning an additional 60 vertices to be generated). Due to the varying limitations of graphics cards, this technique would be heavily hardware dependent. Therefore, we chose to employ scattered texture writes, giving the ability of writing into an arbitrary number of fragments within the texture volume.

A triangle should be rasterized to the single plane where its projection onto that plane results in the greatest surface area. This requirement is met when a triangle is projected onto the plane that is most perpendicular to the triangle's normal. In order to avoid dynamic branching, all three projection matrices were constructed in advance, and a simple interpolation technique was used to select the appropriate matrix during the geometry shader stage.

It is essential to store fragment data to be used in further lighting calculations, however 4-channel textures can only store four floating point numbers per texel. Thus, it was necessary to work with multiple images, which made it important to group some information and re-use it where possible. Simply dedicating a new texture for any data-type when desired would consume a lot of memory, so numbers were packed into a single float value as much as possible. Every voxel had to contain a normal, base material colour (raw colour of the unlit surface) and the transparency. During the voxelisation stage, each voxel block had to determine the shading coming from the light sources in the scene. The block's centre was compared to the existing value in the shadow map, determining whether it was in the shadow. This

was combined with the squared fall-off attenuation to make voxels dimmer the further they were from the light source, as well as its colour. This information was stored to a third 3D-texture, referred to as *Diffuse*.

3.3 Light Propagation

The voxel cone tracing (VCT) technique is now addressed with particular reference to how it is adapted to use 3D textures. When several voxel cones are combined, they can approximate a hemisphere, representing all the light incoming from nearby objects. This approach relies on increasingly averaging the sampled content from the textures, the further a ray gets from the starting position of a cone. Thus, it was necessary to mipmap the 3D textures, combining the colours of nearby voxels into higher-level versions of the texture. Originally the default functionality provided by OpenGL (*generateMipmaps()*) was used. However this turned out to be unproductive, as the mipmaps were built on the client side, via CPU and only then uploaded to the GPU. Each call to this function stalled the program for three hundred milliseconds, so a custom solution had to be devised.

The solution entailed the use of OpenGL compute shaders to spawn a kernel per each yet-to-be-made mip level of a texture. The compute shaders work in parallel to build the entire mip-chain. The process required sampling eight texels from a higher level, to be interpolated and stored in the desired part of the current mip level. The hardware provides this functionality through linear filtering, so it was only necessary to perform a single sample, in the middle of the eight texels. The hardware performs interpolation and the resulting value can be directly stored at the needed position in the mipmap (the octree approach of the reference solution can not take advantage of this hardware interpolation).

As multiple layers of textures are generated in this manner, it is possible to use VCT to achieve a first bounce of global illumination. Because of the way the cones are represented, their volume is estimated by a series of successive cube-shaped texture samples. Each cube is sampled from a higher mipmap level where the texel size corresponds to the size of the current sample within that cone. It is possible to estimate the size of such a cube (and therefore the corresponding mipmap level) based on the distance travelled from the cone's origin and the tangent function. Figure 1 shows a voxelised cone across a range of mip-map levels in the space.

Because the sample-size may not exactly correspond to the dimensions of a mipmap's texel, trilinear filtering was required. This removed occurrences of hard transits along the mip-chain, interpolating between the two closest 3d-texture mip-levels. As the hemisphere is represented by 7 cones of 45 degrees each, a limit is needed on the number of steps each cone can carry out before termination. Implementing the texture volumes with 128 cubed resolution, a maximum of seven steps sufficed - each new cone samples from a higher mipmap, of which there were seven in total.

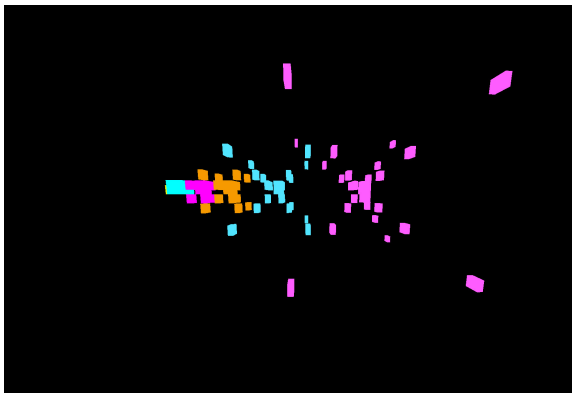

Figure 1: A 45 degree cone within the voxel volume expanding rightwards. Each colour change represents transition to a higher level mipmap. An X shape is centred in each layer.

RGB and Alpha channels get averaged as the mip-chain is computed. When Alpha represents transparency it is possible to compute the accumulated opacity sampled by the cone. Using this information, it can be determined when a calculation of the incoming light from a particular direction can be stopped, meaning that enough opaque objects were encountered on the way and the search should be terminated. When few wide cones are used to estimate the hemisphere, the transparency might take longer to be gathered, resulting in extra colours that may come from the obstructed objects. Such a phenomenon is referred to as *light leaking*. Due to the increasing intervals, this problem might still occur with narrow cones, where a geometry voxel is simply missed by the samples, resulting in other surfaces located behind being sampled. This can be mitigated to an extent by avoiding using narrow geometry, and combining the illumination with the post-processing Screen-Space Ambient Occlusion (SSAO) effect. As hardware computing power increases, shorter intervals can be employed, rectifying the core of the issue.

Additionally, two thin planes located close to each other might occupy two neighbouring voxels. During the mipmapping process, their data will be averaged into a lower-resolution variant of the textures, leading to a further cause of light leaking. To reduce this,

anisotropic filtering should be used, where the contents of the grid vary based on the viewing direction. This however, requires more sampling of the textures which would lead to increased processing times.

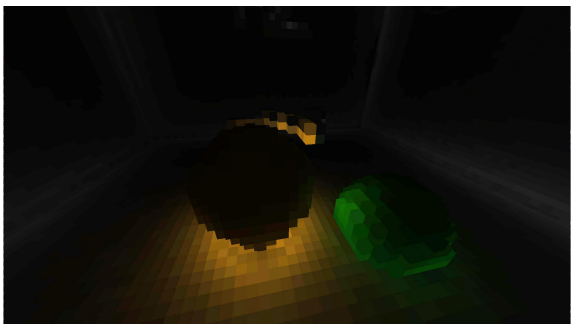

Figure 2: Single bounce illumination volume. This is used as the input to the second light bounce calculation. The final image is shown in Figure 5.

Assembly of the final image requires restoration of each fragment's world position and normal, using the deferred pipeline results. That is followed by performing cone tracing for each fragment, similar to how it was done for the 3D textures. Because the diffuse and first-bounce volumes are available, they are used to gather single and double-bounce pixel-lighting. This operation has to be performed uniquely for each fragment, otherwise the image will contain the voxel grid contents on the surfaces of objects.

3.3.1 Multiple Bounce Complexity

The voxel cone technique entails a linear increase in computational cost as a higher number of light bounces are introduced. This is in contrast to other ray-casting techniques which see an exponential rise (as each ray generates multiple further ray casts from each bounce). To achieve a double bounce, the diffuse information is mipmapped and used for the first-bounce, as shown in Figure 2. This first bounce lighting information is then again mipmapped and used to calculate the second bounce in the same manner, and so on.

Memory cost also increases linearly, as the additional requirement is simply a new mipmapped texture, constituting another stage of light transfer. However since we are dealing with a three-dimensional volume of texels, any increase in resolution (or space size) must obey the "power of two" rule (each texture size must be a power of two in all dimensions). This implies that each further growth in resolution will consume eight times the previous data-structure memory. Additionally,memory must be allocated for the mipmapping chain, that is an extra thirty-three percent of the memory for the chained textures.

3.4 Emissive Surfaces

Emissive entities were also added to our solution (Figure 3). Each object, as well as a base RGB colour, and Alpha for transparency, is able to glow, discarding any effect from the nearby light sources (such as lambert, attenuation or shadowing) due to being luminous on its own. The amount of glow and adherence to the external shadowing was controlled by a *brightness* parameter. The diffuse lighting values of an emissive surface could be modulated linearly, from 100% to 0% for brightness values below 0.5. For higher values, the entity is made unresponsive to illumination effects and only emits its pure, base colour onto the surrounding objects. Higher values simply meaning the object emits a stronger colour.

Some adaptations were made to implement the self-illumination technique. During the construction of the diffuse texture any shadowing has to be reduced, depending on the brightness parameter of the triangle primitive being voxelised. The final value in the diffuse output will then contain the diffuse colour, as well as the base-colour of the voxelised surfaces if the triangle is emissive.

Figure 3: Emissive orange and blue spheres illuminate the white-surface character producing soft shadows.

Clearly, it is important to retain the brightness in a texture that will be mipmapped, as it is an important coefficient during all the cone-tracing stages. The final step of transmitting the glow onto the nearby objects is to carry out the standard first-bounce gathering, which picks up emissiveness by default. The emissive illumination affects the surroundings due to the brightness coefficient sampled by the cones, unlike when sampling standard, non-glowing neighbouring voxels. As the viewpoint moves towards the emissive surface, the resulting illumination produces soft, life-like shadows with umbra and penumbra, their quality depends on the resolution of the data structure.

3.4.1 Voxel Cone Orientation

During the voxel cone tracing, an array of cone directions had to be re-oriented via multiplication of the Tangent-Binormal-Normal (TBN) matrix for a given fragment shader invocation. As a result, the entire *bouquet* of such vectors was modified to face in the direction of the voxel normal. Because the matrix had to be constructed at runtime from the normal, it was not possible to supply it as a pre-computed uniform value. Similarly, the array of cone directions had to be multiplied by the matrix, thereby consuming some processing power. The situation was improved by ensuring the array is supplied with vectors of pre-normalized directions, although this enhancement was minor. A bouquet of seven cones was used, each one being forty-five degrees wide, yielding a relatively-homogeneous hemisphere representation (Figure 4).

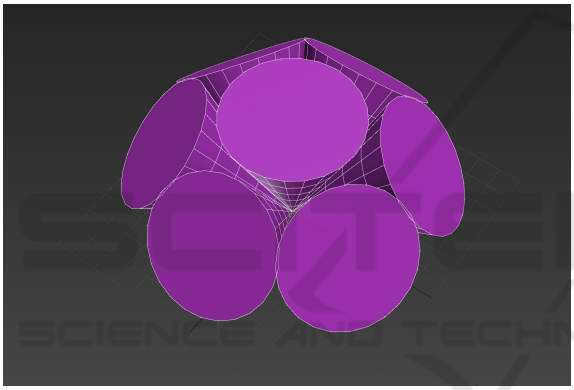

Figure 4: The bouquet of seven 45° voxel cones used in the tracing algorithm.

It was also necessary to spin the entire bouquet around the voxel normal after it was aligned in that direction. Without this improvement, cones dispatched from the voxels of walls and the floor could light up in mild, yet visible, ellipsoidal patterns that grew larger as the spherical emissive objects approached those surfaces. Spinning the bouquet eliminated the presence of inevitable gaps between cone volumes and yielded more plausible visual results (as also reported in the reference solution (Crassin et al., 2011).

4 RESULTS AND EVALUATION

Implementation of voxel cone tracing using sparse 3D textures to simulate double-bounce indirect lighting resulted in a 1920x1080 resolution image locked at 60 frames per second on a GTX980 graphics card. The data structure contained 128^3 texels. Two mipmapped

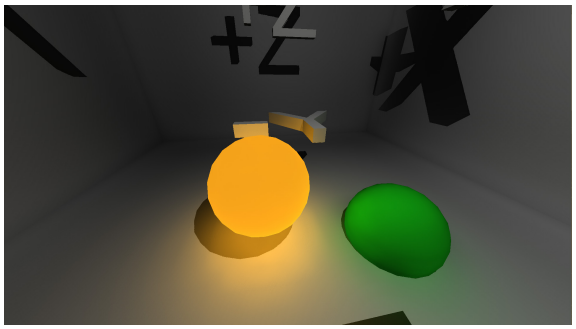

Figure 5: Final image with diffuse and double bounce indirect illumination.

textures were used along with the base-colour and normals texture (which did not require a mip-chain). The interpolation scheme employed in the generation of reflective shadow maps (Dachsbacher and Stamminger, 2005) was used to reduce the number of computed pixels and achieve high-quality frame rates, upscaling one third of the original resolution to the full screen and yielding an additional 8 frames improvement. An image from the resulting render is shown in Figure 5.

The relative computational expense of each step in the process is presented in Figure 6. The final per-fragment cone-tracing step remains the most computationally expensive. The computational efficiency afforded by the direct hardware access to the 3D sparse texture data, and the interpolation step between sampled voxel values, is evident in the relatively low computation costs of the voxelisation and light propagation steps. These results compare favourably with those presented in the reference work (Crassin et al., 2011) where a lower frame-rate is achieved for a 1024x768 image using the sparse octree approach.

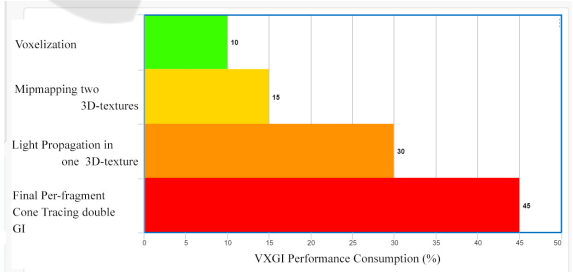

Figure 6: Relative computational expense of the four stages of voxel cone tracing technique for two-bounce indirect illumination using sparse 3D textures.

Table 1 shows the computational timing for achieving double bounce indirect illumination using sparse 3D textures on our test environment at three different screen resolutions (1920x1080, 1600x900 and 1024x768). In each case timings are averaged across 1000 iterations of rendering the scene. Com-

Table 1: Computation timings (in milliseconds) for stages of double bounce voxel cone tracing using sparse textures at different screen resolutions. All timings averaged over 1000 iterations.

Screen resolution	1920x1080		1600x900		1024x768	
	ms	%age	ms	%age	ms	%age
Voxelisation	0.74163	4.02	0.6672	4.24	0.656725	5.23
Generate mipmap 1	1.383379	7.5	1.312604	8.35	1.281829	10.2
First bounce (voxel space)	5.590338	30.31	5.596479	35.59	5.577985	44.4
Generate mipmap 2	1.194863	6.48	1.184982	7.54	1.168038	9.3
Second bounce (screen space)	9.531563	51.68	6.963091	44.28	3.878667	30.87
Total	18.441773		15.724356		12.563244	

paring the required computation time for the first bounce (in voxel space) and the second bounce (in screen space) shows that, as expected, the most significant cost of increasing the screen resolution is on calculation of the second bounce, as the calculation takes place in screen space.

The two mipmapping steps, and the voxelisation of the scene, are not the most computationally expensive aspects of the process. Further optimisations in these steps would therefore have less of an effect. The mipmapping stages in particular could be further optimised through the use of automated mipmap generation extensions such as OpenGL's SGIS.

Some further savings in computation time can be considered, although they have a trade-off in potential degradation of image fidelity. Partial computation of the update of the mipmap textures (a fraction of texture volume per frame) can speed up performance significantly when coupled with the interpolations discussed previously. This optimisation however must be approached with caution, as deferred recomputation can result in voxels lagging behind their respective fragments, an effect which will become more pronounced with increasingly dynamic scene elements. The adoption of an axis-aligned bounding box (AABB) for voxelisation, would result in an update only occurring for the texels captured by the box. Similarly, lighting computations can be carried out partially, for a portion of the illuminated 3D textures.

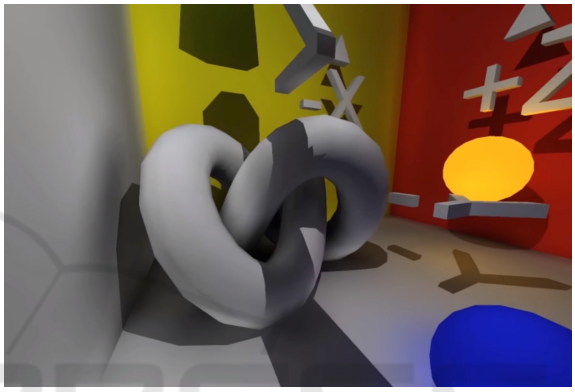

Figure 7: Image from final render with double bounce indirect lighting combined with shadow-mapping.

5 CONCLUSIONS

Voxel Cone Tracing has been successfully employed for indirect illumination in real-time rendering of scenes, as the technique allows for the calculation of multiple light bounces with only a linear additional cost in computation. To date, the voxelised scene has been stored in the form of a sparse octree, taking advantage of GPU caching for high speed access to the scene's spatial information. An alternative structure for storing the 3D scene information to be used in the calculation of indirect illumination is the 3D texture, as it provides the required hierarchical data structure (through mip-mapping) and can be stored in a hierarchically sparse manner (through sparse texturing).

In this paper we have presented an implementation of voxel cone tracing for global illumination utilising sparse 3D textures for the storage of the 3D spatial information of the rendered scene. The performance and fidelity of our implementation has been shown to be of comparable quality to the reference solution. Utilising 3D textures allows us to take advantage of hardware interpolation when sampling the mip-mapped textures which represent the voxelised scene. This provides very efficient bilinear and trilinear interpolation, with little software overhead, so that the estimation of lighting parameters between the voxelised points is highly efficient.

As GPU and graphics card technology evolves, the integration of 3D texture access into the hardware is likely to increase. This will make the adoption of 3D textures in the context of voxel representation of a scene for lighting an increasingly attractive proposition, as both access times and high-speed memory

devoted to 3D textures are likely to significantly improve with the hardware.

REFERENCES

Crassin, C., Neyret, F., Sainz, M., Green, S., and Eisemann, E. (2011). Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pages 1921–1930. Wiley Online Library.

Dachsbacher, C., Křivánek, J., Hašan, M., Arbree, A., Walter, B., and Novák, J. (2014). Scalable realistic rendering with many-light methods. In *Computer Graphics Forum*, volume 33, pages 88–104. Wiley Online Library.

Dachsbacher, C. and Stamminger, M. (2005). Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231. ACM.

Gao, Y., Bouix, S., Shenton, M., and Tannenbaum, A. (2013). Sparse texture active contour. *IEEE Transactions on Image Processing*, 22(10):3866–3878.

Goulding, D., Smith, R., Clark, L., Ushaw, G., and Morgan, G. (2012). Real-time ambient occlusion on the playstation3. In *GRAPP/IVAPP*, pages 295–298.

Hachisuka, T. (2005). High-quality global illumination rendering using rasterization. *GPU gems*, 2:615–633.

Jendersie, J., Kuri, D., and Grosch, T. (2016). Precomputed illuminance composition for real-time global illumination. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 129–137. ACM.

Jensen, H. W. (2001). *Realistic image synthesis using photon mapping*, volume 364. Ak Peters Natick.

Kajiya, J. T. (1986). The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM.

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

Laine, S. and Karras, T. (2011). Efficient sparse voxel octrees. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1048–1059.

Lazebnik, S., Schmid, C., and Ponce, J. (2005). A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1265–1278.

McLaren, J. and Yang, T. (2015). The tomorrow children: lighting and mining with voxels. In *ACM SIGGRAPH 2015 Talks*, page 67. ACM.

Mittring, M. (2012). The technology behind the unreal engine 4 elemental demo. *part of "Advances in Real-Time Rendering in 3D Graphics and Games," SIGGRAPH*.

Nichols, G., Shopf, J., and Wyman, C. (2009). Hierarchical image-space radiosity for interactive global illumination. In *Computer Graphics Forum*, volume 28, pages 1141–1149. Wiley Online Library.

Peyré, G. (2009). Sparse modeling of textures. *Journal of Mathematical Imaging and Vision*, 34(1):17–31.

Ritschel, T., Dachsbacher, C., Grosch, T., and Kautz, J. (2012). The state of the art in interactive global illumination. In *Computer Graphics Forum*, volume 31, pages 160–188. Wiley Online Library.

Shreiner, D., Sellers, G., Kessenich, J. M., and Licea-Kane, B. (2013). *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. Addison-Wesley.

Sundén, E. and Ropinski, T. (2015). Efficient volume illumination with multiple light sources through selective light updates. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 231–238. IEEE.

Weiler, M., Westermann, R., Hansen, C., Zimmermann, K., and Ertl, T. (2000). Level-of-detail volume rendering via 3d textures. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 7–13. ACM.