Interactive Indirect Illumination Using Voxel Cone Tracing

Andreas Heider Technische Universität München
heidera@in.tum.de

Abstract

Plausible illumination is crucial for rendering realistic images, but real time global illumination is still an unsolved problem. Voxel cone tracing is an extension of ray tracing and can be used for fully dynamic two-bounce indirect illumination, with both diffuse and specular components. Through the use of a hierarchical voxel scene representation and the GPU, voxel cone tracing is able to run at interactive frame rates.

1 Introduction

We see the world by detecting a huge number of photons with photoreceptors in our eyes. Each photon has travelled along a path through the world, originating from a light source, then possibly bouncing around multiple times before finally ending up in one of our eyes.

To generate realistically illuminated images, the effects of this light transfer process have to be simulated. Finding a good approximation, balancing runtime and image quality, is one of the key challenges in computer graphics.

In a typical rendering pipeline, the lighting is evaluated during the shading of a pixel or fragment: When a pixel of an image is rendered, its color is determined by approximating how much light enters the eye from the direction covered by that pixel.

1.1 The rendering equation

The theoretical foundations of the light transfer are mathematically described by the rendering equation [9]:

$$L_o(x,\omega) = L_e(x,\omega) + L_r(x,\omega) \tag{1}$$

It states that the outgoing light L_o from a point x in direction ω equals the amount of light emitted L_e towards this direction plus the reflected light L_r.

The reflected light is modeled by an integral over the product of the bidirectional reflectance distribution function (BRDF) of the material at the point x, which defines how incoming light is reflected, multiplied with the incoming light L_i from all directions ω_i.

$$L_r(x,\omega) = \int_{\Omega^+} L_i(x,\omega_i) f_r(x,\omega_i \to \omega) \langle N(x), \omega_i \rangle^+ d\omega_i \tag{2}$$

The rendering equation is also illustrated in figure 1.

Efficiently evaluating the rendering equation is hard, because the amount of incoming light L_i at one point recursively depends on the outgoing light L_o of many other points.

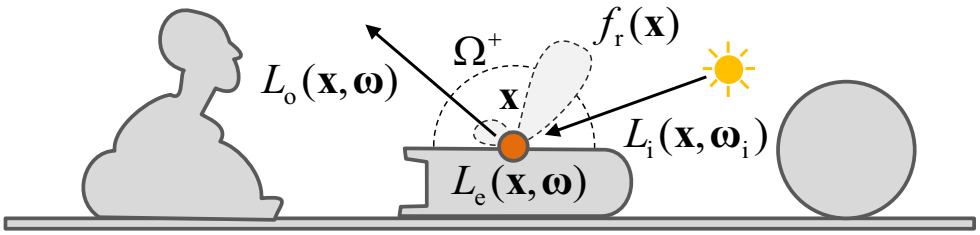


Figure 1: *The components of the rendering equation [9]*

1.2 Previous approaches

To be able to compute the lighting in real time computer graphics, running with at least 30 frames per second, the rendering equation has to be approximated to make it cheaper to evaluate.

One extreme is to completely eliminate the complex recursive dependency and only consider direct light sources for the reflection term. Combined with a simple BRDF this results in the widely used phong lighting model.

1.2.1 Lightmaps

But totally disregarding indirect illumination leads to unrealistic images. The next step for interactive computer graphics was the use of lightmaps. They were so successful through the combination of speed and quality that they are still broadly used.

This technique is based on one main insight: When only diffuse reflection is considered, meaning that materials reflect incoming light evenly to all directions, the resulting illumination is independent of the view direction. Therefore it is possible to do expensive light simulation for the indirect illumination once for the entire scene and save it to a lightmap. During rendering the indirect illumination can simply be looked up in the lightmap, which is extremely fast.

The drawback of this method is that the lighting is static and only diffuse indirect illumination is possible. Moving light sources or geometry requires re-computation of the indirect lighting, which is expensive but possible in real time nowadays [7].

1.2.2 Ray tracing

A different approach to rendering that is mostly used for non-interactive applications is ray tracing. The idea behind it is to stochastically sample the rendering equation, as illustrated in figure 2 [9]. To evaluate the shading at a point, rays are sent out following the path of light backwards from the observer towards the light sources.

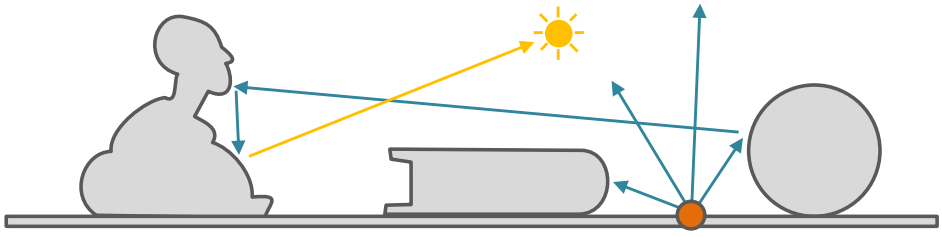


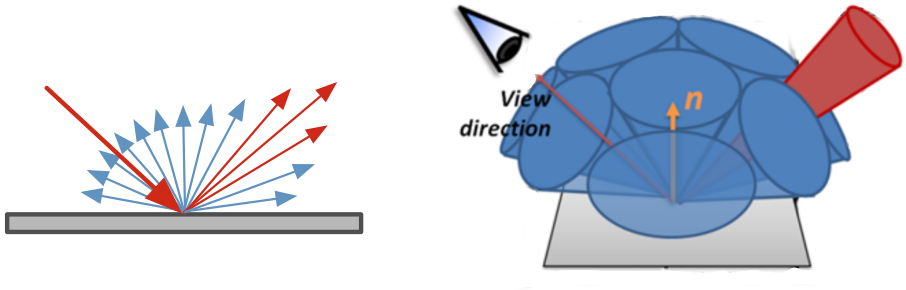
Figure 2: *Sampling the rendering equation with ray tracing [9]*

Directly integrating the rendering equation in this way has the advantage that many lighting effects that require special attention when using rasterization are automatically achieved. However, ray tracing is prone to highly visible noise unless a large number of rays have been sent out per pixel.

Over many years of research, a plethora of optimizations has been developed to accelerate ray tracing, but even the most advanced ray tracers such as Brigade just barely manage real time frame rates when simulating global illumination.

2 Voxel cone tracing

Voxel cone tracing improves upon ray tracing to make it fast enough to replace lightmap lookups for indirect illumination in real time computer graphics. Crassin presents a whole framework for this purpose in [4].



(a) *Ray Tracing: many rays* **(b)** *Cone Tracing: few large cones [2]*

Figure 3: *Ray tracing compared to cone tracing*

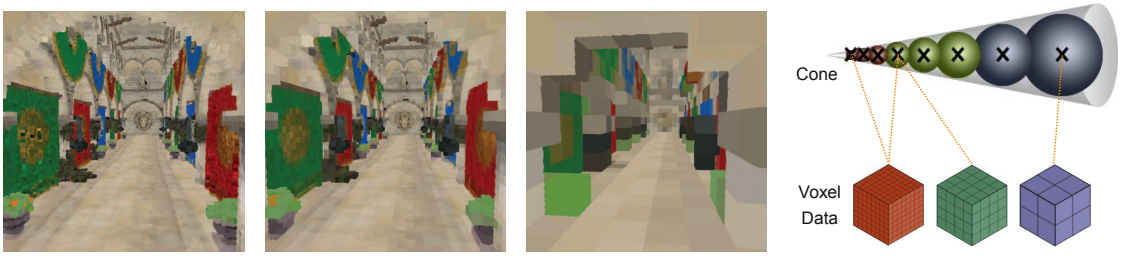
Figure 3 illustrates the key idea behind cone tracing: Large groups of rays can be approximated by a single cone, so less traces are required.

However, tracing a cone on detailed polygon data is still too expensive. Finding the intersection points between a cone and polygon geometry is more complex than ray-polygon intersection and a single cone might intersect with a large number of polygons.

2.1 Fast cone tracing on voxel data

Cone tracing can be made fast by using voxels to represent the scene instead of polygon data. A voxel scene can be filtered down to a less detailed version in a very straightforward way, as shown in figure 4a. Furthermore, multiple levels of detail can then be stored efficiently in an octree.

This hierarchical voxel structure can later be used for approximate cone tracing by tracing a simple ray through the voxel scene, as shown in figure 4b. To achieve the effect of an extending cone, an increasingly coarser version of the scene is used for the trace samples as the cone gets bigger. Intermediate values are generated by interpolating between the different resolutions.



(a) *Different levels of detail of a voxel scene* **(b)** *Cone tracing on voxel data [8]*

Figure 4: *Using a hierarchical voxel scene for fast cone tracing*

2.2 Using cone tracing for indirect illumination

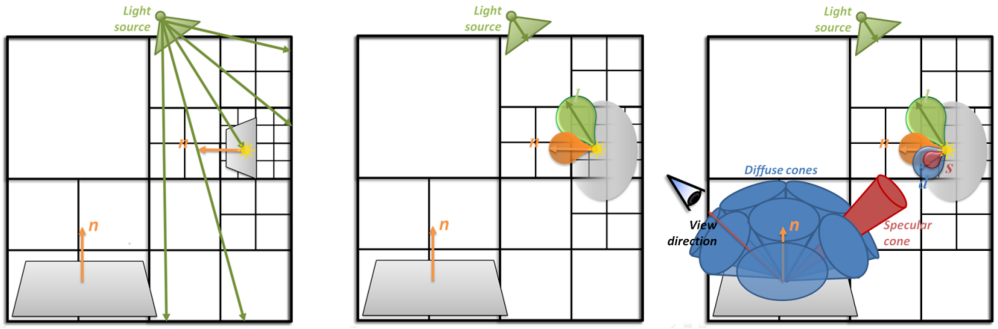
Fast cone tracing can then be applied to efficiently compute indirect illumination. Crassin still uses traditional methods for direct illumination, but additionally adds two-bounce indirect illumination by tracing cones from the point to be shaded.

Figure 5 shows the three passes of the indirect illumination algorithm:

First, light is distributed from the light sources and written into the octree, at the highest level of detail of the scene. After this step, each leaf voxel in the octree contains the direction and intensity of the incoming light.

To be able to access lighting information in lower detail voxels, it is then filtered down in a second pass. Due to the regular structure of the octree the filtered value of a coarser node is simply the average of its descendants.

In the final rendering phase the actual cone tracing is performed. Cones are traced according to the BRDF of the material, accumulating incoming indirect light.



(a) *Pass 1: Light distribution* **(b)** *Pass 2: Filtering* **(c)** *Pass 3: Cone tracing*

Figure 5: *Algorithm overview [4]*

2.3 Voxelizing polygon scenes

Modern game engines are almost exclusively based on polygon geometry, so the scene has to be converted to a voxel representation before voxel cone tracing can be used. Within the voxel cone tracing framework this is fully done on the GPU, using an approach based on the GigaVoxels engine [2].

During the voxelization the entire scene is rasterized along all three major axes with the desired resolution. But instead of writing shaded pixels into a framebuffer, the fragment shader is used to write the corresponding voxels into memory. More details about the voxelization process can also be found in [3].

Furthermore, the voxel scene has to be kept in sync with the polygon scene when geometry changes. To improve the performance, the scene is split into a static part, which is only voxelized once, and a dynamic part that is updated every frame.

3 Conclusion

Voxel cone tracing is not a new idea. The underlying techniques have long been known, for example cone tracing has already been used by Amanatides to speed up ray tracing of diffuse materials almost 30 years ago [1]. The same is true for the application of voxels, which have been used in computer graphics just as long.

Recently there has been a revival of voxels in computer graphics, with a multitude of publications using them for applications such as global illumination [10], ambient occlusion [5] and shadows [11].

While voxel cone tracing is certainly a part of this movement, it stands out by providing a practical solution to a pressing problem. Through a combination of using voxels to speed up cone tracing and moving almost the entire tracing pipeline to the GPU, voxel cone tracing achieves true interactive indirect illumination at real time framerates. The current implementations are limited to just two-bounce indirect illumination, which is comparable to other real time approaches such as light propagation volumes [6].

Additionally, voxel cone tracing can simulate specular indirect illumination, which was not possible with classic methods like radiosity.

Thanks to interpolation and the hierarchical voxel structure, the visual artefacts that occur due to the various approximations used by voxel cone tracing just lead to blurring instead of the high frequency noise seen with e.g. monte carlo path tracers. This kind of error is much less visible than flickering errors created by other methods. It also makes this method especially suited for the rendering of soft shadows, in contrast to traditional methods gets faster the softer the shadow is.

The main drawback of voxel cone tracing is the need for a whole second scene representation in addition to the original scene. Keeping both scene representations in sync can become very expensive, especially when large parts of the scene are dynamic.

There are still a number of open issues that require further research, such as light leaking with thin geometry. But voxel cone tracing is already getting accepted by the industry, with the Unreal Engine 4 being the first major engine to incorporate an improved version of it [8].

References

[1] John Amanatides. *Ray Tracing with Cones*. 1984.

[2] Cyril Crassin. *GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes*. PhD thesis, Grenoble University, 2011.

[3] Cyril Crassin and Simon Green. *Octree-Based Sparse Voxelization Using The GPU Hardware Rasterizer*. CRC Press, 2012.

[4] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. *Interactive Indirect Illumination Using Voxel Cone Tracing*. 2011.

[5] Eduardo Ceretta Dalla Favera and Waldemar Celes Filho. *Ambient Occlusion Using Cone Tracing with Scene Voxelization.* 2012.

[6] Anton Kaplanyan and Carsten Dachsbacher. *Cascaded light propagation volumes for real-time indirect illumination*. I3D '10. ACM, 2010.

[7] Sam Martin and Per Einarsson. *A real-time radiosity architecture for video games*. SIGGRAPH, 2010.

[8] Martin Mittring. *The Technology Behind the "Unreal Engine 4 Elemental demo"*. 2012.

[9] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. *Comput. Graph. Forum*, 31(1):160–188, February 2012.

[10] Sinje Thiedemann, Niklas Henrich, Thorsten Grosch, and Stefan Müller. *Voxel-based global illumination*. I3D '11. ACM, 2011.

[11] Chris Wyman. *Voxelized shadow volumes*. HPG '11. ACM, 2011.