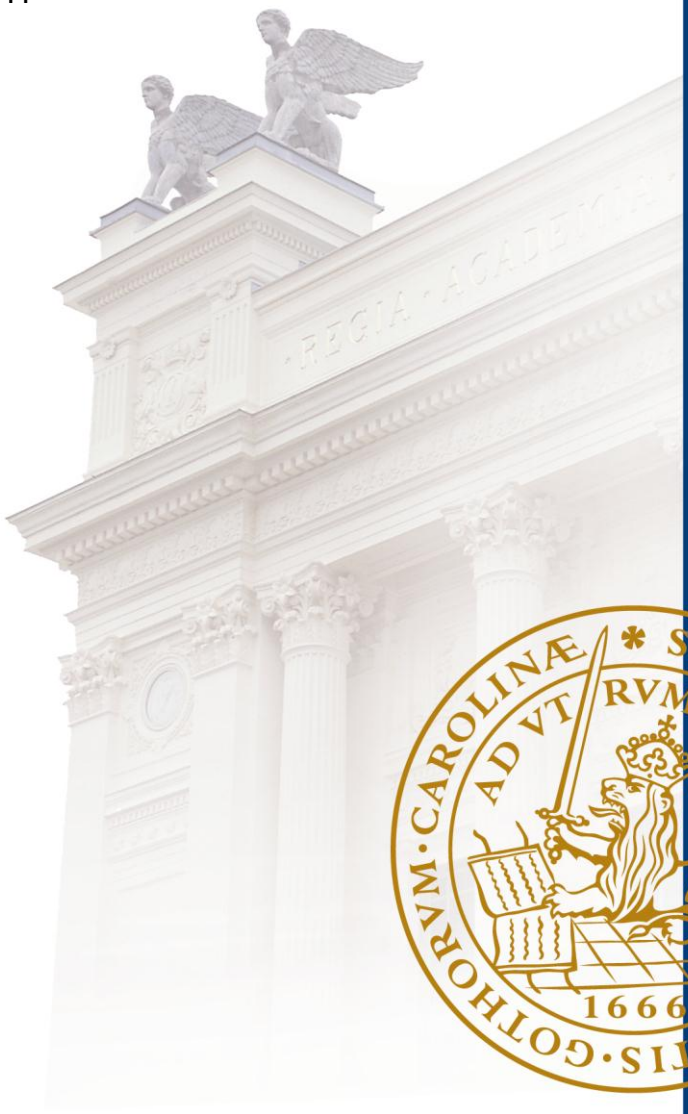
Master's Thesis

Octree light propagation volumes

John David Olovsson

Department of Computer Science
Faculty of Engineering LTH
Lund University, 2012



ISSN 1650-2884
LU-CS-EX: 2012-16

LUND UNIVERSITY

MASTER THESIS PROJECT

Octree light propagation volumes

Author:
John David Olovsson[*]
Department of computer science
Faculty of engineering, LTH
Lund University

Supervisor:
Michael Doggett[†]
Faculty of engineering, LTH
Lund University

Examiner:
Tomas Akenine-Möller[‡]
Faculty of engineering, LTH
Lund University

September 9, 2012

[*] dt07jo1@student.lth.se
[†] mike@cs.lth.se
[‡] tam@cs.lth.se

Abstract

This master thesis project presents a new method for representing light propagation volumes using an octree data structure, and for allowing light from regular point light sources to be injected into it. The resulting technique uses full octrees with the help of a separate data structure for representing the octree structure. Light from point light sources is injected into the octree using an already proven technique.

Some key parts of the original technique prove to be well suited for the new data structure and can be greatly sped up by it. Other parts are mostly unaffected and some new steps need to be introduced. The implementation of the technique renders the Sponza Palace Atrium scene at 9.4 frames per second, with good visual quality. This is with all steps involved in the technique including indirect occlusion. While the implementation did not achieve real-time frame rates, several possible improvements are presented. These may help to achieve the desired frame rates. Additionally there are some visual artefacts that can be experienced with the new technique, but they are generally not noticeable. Light injection from multiple point light sources with the method used in this project infers a large overhead compared to a single directional source. This additionally counters the viability of the technique in practical real-time applications.

Acknowledgements

This thesis would not have been possible without, first and foremost, Michael Doggett for being the supervisor of the project and for providing help and guidance throughout the entire project. Additionally Thomas Kjeldsen, at the Alexandra institute, provided invaluable assistance in explaining some of the maths used for point light injection for subsurface light propagation volumes. The thesis project also owes thanks to the examiner, Tomas Akenine-Möller. Finally, fellow student Cem Eliyürekli has provided much assistance with quality control and feedback on the project report.

Contents

1 Introduction

Realistic real-time illumination has been a popular field of research in computer graphics and games for a long time. One large subset of real-time rendering techniques are dedicated to providing realistic *local illumination*. There are also techniques that try to model various aspects of *global illumination*. Some of these focus on for example *diffuse indirect lighting* or *specular reflections*.

Unfortunately the nature of global illumination makes it computationally difficult to achieve in real-time. As such it is common for techniques targeted at real-time applications to use rather crude approximations or to rely on preprocessing. One problem with techniques relying on preprocessing is that they are usually static. The introduction of dynamic changes of the environment would require the preprocessing to be redone in order to retain correct illumination.

Light propagation volumes is a technique which approximates global illumination, specifically diffuse indirect lighting, without any preprocessing stage, in real-time[10]. In practice this means that the technique approximates the effect that light has on the scene after it has been reflected from diffuse surface. The most prevalent visual result of this is *light bleeding*. The technique relies on storing a representation of the light in each part of the scene in a uniform grid. It is then iteratively propagated throughout the scene.

While the light propagation volumes technique does run in real-time the uniform grid representation does limit the scalability. In order to achieve a more detailed and accurate result, the grid resolution would have to be increased. Unfortunately this also means that it would require more iterations to propagate the light throughout the scene. Such a high resolution representation would also be wasteful in any part of the scene which has no or relatively uniform geometry.

Many of the available implementations of the light propagation volumes technique use a single directional light source. This efficiently models for example sun light, but it not suited for local light sources. Such are better represented using point light sources.

This master thesis project investigates whether a representation using *full octrees*[1] instead of a uniform grid will allow for more efficient and scalable technique. The motivation for doing so is that an octree could allow for light to propagate further using fewer iterations due to the varying element sizes. The project also explores the possibility of using multiple fully dynamic point light sources instead of a single directional light source.

The project will essentially be an extension on top of the original light propagation volumes technique. It will incorporate omnidirectional point lights as the primary light sources and a dense octree implementation as the data structure. An implementation of light propagation volumes featuring omnidirectional point lights will be used as a baseline for comparison with the octree implementation. The visual quality and performance of the octree implementation can then be compared to this baseline version in order to evaluate whether or not it is an improvement.

The purpose of this thesis is to propose a method for using octrees as the underlying data structure in the light propagation volumes technique. It also examines the resulting differences of using such a structure compared to the

[1] A full octree is an octree where all possible nodes are allocated. This is explained further in section 4.2.

original technique. Finally, it will verify whether or not the technique is suitable to be used with dynamic point light sources in real-time applications.

It is not a focus in this project to optimize all parts of the resulting implementation but rather just the octree specific code. Everything else has already been shown to be feasible to implement in real-time by the original paper[10].

Outline A survey of previous work and related techniques is presented in section 2 in order to place this project into perspective. Section 3 presents a fairly detailed insight in the related concepts of spherical harmonics and light propagation volumes. Afterwards, in section 4, a description of the resulting method is presented including the theory on how it works. In section 5 the actual implementation and its capabilities are described. The results and an analysis of them is later presented in section 6. Then follows a discussion of the project with focus on the results and possible future research in section 7. Finally a conclusion including a quick summary of the entire project is presented in section 8.

2 Previous work

There has been much research in the field of global illumination in computer graphics and much of it is very recent. This generally includes not only purely real-time techniques but also static or semi-static solutions based on precomputation steps as well as entirely offline solutions. This thesis is however focused on real-time applications and as a result, offline solutions will only be mentioned briefly while real-time techniques will be considered, presented and compared in further detail in this section.

Light propagation volumes Light propagation volumes is the underlying technique that this project extends upon. The technique was first developed by Kaplanyan for Crytek[10]. It makes it possible to render scenes with realistic *diffuse* indirect lighting in real-time on current generation personal computers and gaming consoles. This process is performed in four distinct steps. The scene is first rendered from each light source into *reflective shadow maps*. The texels in these reflective shadow maps are then used to represent virtual point lights. After the reflective shadow maps have been created the contributions of the virtual point lights are *injected* into a uniform three dimensional grid. The grid encodes light color, intensity and direction as *two-band spherical harmonics*. After the injection stage these initial values in the grid are then *propagated* throughout the rest of the grid using an *iterative* process. In each iteration each grid element receives a contribution from each of the grid elements immediately surrounding it. The accumulated result of all the propagation iterations is then sampled during *rendering* to illuminate the final image.

A more detailed understanding of this process is essential to fully understand the contributions of this project. Therefore the light propagation volumes technique will be presented in further detail in section 3.2.

Cascaded light propagation volumes The cascaded light propagation volumes technique is an extension of the original light propagation volumes technique. It was developed and published by Kaplanyan who also wrote the paper for the original technique as well as Dachsbacher[11]. This technique contains all of the steps from the original technique. It replaces the single uniform grid with multiple such grids with different density, positioned relative to the camera. These grids all have the same resolution, but because of the different densities they have different extents in the scene. A high density grid is used close to the camera providing high detail close to the viewer while gradually lower density grids are used further away from the camera.

The data structure used in cascading light propagation volumes is in some ways very similar to that of the octree light propagation volumes which is presented in section 4.2. Both use multiple overlapping grids. The difference is that cascaded light propagation volumes uses a small preset of grids which all have the same resolution. Octree light propagation volumes uses more volumes but with different resolutions. In cascaded light propagation volumes the different grids also cover a differently sized area in world space while octree light propagation volume grids all span the same volume, no matter their resolution. Finally, cascaded light propagation volumes do not need any separate grid to maintain the structure of the different grids.

Subsurface light propagation volumes Subsurface light propagation volumes is another technique derived from the original light propagation volumes technique. Unlike the other techniques this one is intended to approximate the subsurface scattering of light inside solid objects rather then the indirect lighting in regular scenes. The technique was developed by Børlum et al.[19] Apart from adapting light propagation volumes to represent subsurface scattered light, the paper also introduces a new method for injecting the contribution of *point light sources* into a light propagation volume. This new injection method depends on various attributes of the render-to-texture cameras used to create the reflective shadow maps during the first step of the technique. This is described in more detail in section 3.3.

Reflective shadow maps Reflective shadow maps is a technique that extends regular shadow mapping. Each texel in the resulting shadow maps store additional information. This allows the texel to be considered as an independent indirect light source. While a traditional shadow map would usually only store the depth value a reflective shadow map also stores the surface normal, the reflected light flux and the world position. These reflective shadow maps are then evaluated on the fly during rendering to contribute to the indirect lighting of the scene. The technique was developed by Dachsbacher and Stamminger[7]. Reflective shadow maps are also used in the light propagation volumes technique and in the techniques derived from it. When used with light propagation volumes however, instead of being evaluated on the fly during rendering, all the reflective shadow maps are injected into the light propagation volume after which they are no longer used.

Spherical harmonics lighting Spherical harmonics is a mathematical construct which defines an orthonormal basis over the sphere. The basis functions are complex in nature but can be transformed into a real valued basis. These functions are commonly indexed in two dimensions, one of which is usually referred to as the *band index*. Low values of the band index correspond to the low frequency basis functions. Among the early uses of low order spherical harmonics in real-time computer graphics was by Sloan et al[21]. Spherical harmonics has since then been established as a powerful tool in real-time computer graphics[9, 20]. In the light propagation volumes technique the grid stores light flux in the form of low order spherical harmonics. Specifically, only the first two bands of spherical harmonics are used. The theory required to cover such use of spherical harmonics is presented in section 3.1.

Interactive indirect illumination using voxel cone tracing The interactive indirect illumination using voxel cone tracing is another alternative for real-time indirect illumination. It was developed independently from light propagation volumes but still relies on the idea of injecting reflected light into a grid. Unlike the light propagation volumes technique it does not propagate the light in the grid but rather performs a gathering procedure during rendering using cone tracing. The technique achieves real-time performance in complex scenes. Unlike traditional light propagation volumes it is also designed to support glossy indirect lighting as well, not only diffuse indirect lighting. Similarly to the technique presented in this thesis it also implements its structure using an octree.

It does however use a *sparse* pointer based octree rather then a *full* octree. It was originally developed by Crassin et al[8].

Image space photon mapping As one of several techniques aimed towards accelerating typically offline techniques for use in real-time applications indirect illumination by image space photon mapping does just that for the photon mapping technique. Image space photon mapping achieves real-time frame rates on modern hardware in complex dynamic scenes. The technique splits up traditional photon mapping into different steps. Some are executed on the GPU and some on the CPU. Both a *bounce map* and a *photon volume* is introduced in the technique. The bounce map is similar to a reflective shadow map. It is rendered on the GPU from the space of the light source and stores the photon depth, color, direction along with some additional information after the first bounce. The photons are then simulated on the CPU until they are absorbed, storing the incoming photons in the photon volume before each bounce. The photon volumes are then rasterized, scattering the photon contributions in screen space. The technique was developed by McGuire and Luebke[15]. Image space photon mapping yields good looking results in real-time but has some camera dependent artefacts and does rely much on CPU processing which may be required for other tasks.

Incremental instant radiosity Traditional instant radiosity is typically not fast enough for interactive real-time applications[12]. The derived technique of incremental instant radiosity provides an algorithm for reusing virtual point lights generated during one frame in future frames. It also incrementally updates them to ensure that the per-frame computation time is low. This is enough to achieve real-time frame rates. The technique does not require any pre-computation and supports dynamic light sources with the limitation that they must move reasonably smoothly. The geometry causing indirect illumination must additionally remain static and dynamic objects can not cause any indirect lighting, but may receive indirect light from static parts of the scene. The technique was developed by Laine et al[14]. The technique can provide visually pleasing results in real-time but places several severe limitations on both the scene geometry itself and the lighting conditions which may not be ideal in some applications.

Screen space directional occlusion Screen space ambient occlusion is a technique to approximate ambient occlusion using geometry information available in screen space. Screen space directional occlusion is an extension of screen space ambient occlusion which is based on the same principle but allows it to also approximate one bounce of indirect illumination and directional shadows. This is done primarily by retaining more information in the per-pixel evaluation of neighbouring pixels and specifically directional information which is usually discarded when using traditional screen space ambient occlusion. The technique was developed by Ritschel et al[18]. The technique is relatively fast and does not add significant computational complexity on top of the simpler ambient occlusion technique. It does however share the limitation that only on-screen information can be included. As such it is relatively view-sensitive.

Static or semi-static global illumination techniques There are many variations of techniques for global illumination that rely either on an offline precomputation step or on additional artist provided resources. The most common and straight forward of these is probably *ambient light* which is just a single uniform and undirected light color and intensity which affects everything in the scene equally[1]. While such a solution is essentially free with regard to computational complexity and memory use it does not accurately model the indirect lighting of a typical scene. A common yet simple improvement is *environment mapping* which is a set of techniques relying on using a static pre-computed map, usually a texture, to store the light coming into the scene from each direction[2]. Such techniques have the inherent limitation that their contribution is view-independent. As such it is only appropriate for lighting contributions from very distant sources.

Other, slightly more advanced techniques for precomputed global illumination include various forms of precomputed lighting, irradiance and occlusion techniques. These techniques rely on precomputed lighting information stored in maps or volumes and sampled during rendering of the scene. The precomputed values may for example have been generated using an entirely offline technique for global illumination.[3]

Offline global illumination techniques It is common for the offline global illumination techniques to be able to reach a much higher level of accuracy than real-time techniques. While these techniques generally do not achieve real-time performance on modern hardware their results can be saved and reused in static environments. The traditional techniques worth mentioning are *radiosity* and *ray tracing*. Radiosity is designed to simulate diffuse interreflections between objects while ray tracing specializes in glossy reflections, refraction and shadows.[3]

Conclusion After surveying the state of techniques enabling global illumination for real-time applications there are several viable techniques. Some of the techniques are closely related to each other or even extend each other. The relevant such relationships are illustrated in figure 1. Each of them have their own unique limitations. For this project the light propagation volumes technique was chosen as a starting point. This is due to its promising real-time characteristics and for not being restricted to either static scenes or screen space information. The cascaded light propagation volumes technique already extends light propagation volumes and does to a certain extent address some of the issues that this project sets out to investigate. Cascaded light propagation volumes has been designed to exclusively prioritize spatial proximity to the viewer when it comes to the quality and accuracy of the technique. This project will however explore some other possibilities, specifically proximity to scene geometry and reflected indirect light instead.

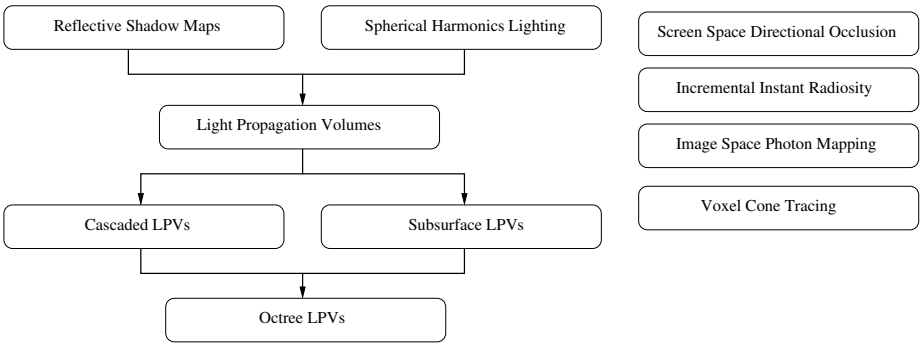


Figure 1: Illustration of the relation between the relevant presented techniques for real-time indirect illumination. Specific focus has been devoted to the relations between those techniques that lead up to this project. The techniques in the hierarchy to the left are closely related or extend each other while those listed on the right have no obvious relation to other techniques in the figure.

3 Real-time global illumination

While there are many possible paths towards approximating global illumination in real-time, as indicated by section 2, this thesis will focus entirely on extending the light propagation volumes technique. This section will explain how light propagation volumes work to the extent required to understand the extensions that were developed as part of this project. These extensions are in turn covered in section 4. Light propagation volumes rely heavily on low order spherical harmonics for representing light. As such, an introduction of spherical harmonics will be given prior to the description of light propagation volumes themselves.

3.1 Spherical harmonics

It does not fall within the scope of this report to provide the complete in-depth theory of spherical harmonics. This section will instead present a short introduction with focus on the properties of spherical harmonics that are used in the light propagation volumes technique and consequently in the technique developed in this master thesis project. Additional background on spherical harmonics can be found in papers devoted to them[4, 20, 9] and in papers which use them as part of a technique[21, 17].

Spherical harmonics are a set of mathematical functions. Together they form an orthonormal basis. The basis spans a function space, specifically the space of functions defined over the unit sphere. There is an infinite amount of these basis functions[20, 9].

The spherical harmonics basis functions are divided into so called frequency bands where each band corresponds to a frequency higher than that of the band before it. The first band consists of a single constant function, the second band consists of three linear functions, the third band consists of five quadratic functions and so on[20].

Spherical harmonics are based on the *associated Legendre polynomials* which in turn are solutions to the associated Legendre differential equation. These associated Legendre polynomials are themselves far outside the scope of this report and will simply be denoted $P_l^m(x)$ where the band index l is a positive integer and $m = -l, \ldots, l$. The spherical harmonics basis functions are then defined as[21]:

$$Y_l^m(\theta, \varphi) = K_l^m e^{im\varphi} P_l^{|m|}(\cos\theta) \tag{1}$$

Here, K_l^m is just a normalization constant:

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}}$$

The arguments θ and φ are the spherical coordinates that correspond to a single point s on the unit sphere. Such a point can also be written as a triplet of Cartesian coordinates (x, y, z) with $\sqrt{x^2+y^2+z^2} = 1$. They are related to each other according to:

$$s = (x, y, z) = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta) \tag{2}$$

Equation 1 will generally result in complex values while techniques in computer graphics usually need to represent only real values. As a result the equation

has been derived into a real valued form as well. Each complex basis function Y_l^m corresponds to a real valued basis function y_l^m:

$$\begin{aligned} y_l^m(\theta,\varphi) &= \begin{cases} \sqrt{2}\,\mathrm{Re}\{Y_l^m(\theta,\varphi)\} & \mathrm{m} > 0 \\ \sqrt{2}\,\mathrm{Im}\{Y_l^m(\theta,\varphi)\} & \mathrm{m} < 0 \\ Y_l^0(\theta,\varphi) & \mathrm{m} = 0 \end{cases} \\ &= \begin{cases} \sqrt{2}K_l^m \cos(m\varphi) P_l^m(\cos\theta) & \mathrm{m} > 0 \\ \sqrt{2}K_l^m \sin(-m\varphi) P_l^{-m}(\cos\theta) & \mathrm{m} < 0 \\ K_l^0 P_l^0(\cos\theta) & \mathrm{m} = 0 \end{cases} \end{aligned}$$

Due to the involvement of the associated Legendre polynomials P_l^m in these basis functions, the polynomial order of each basis function is equal to the band index l. This is why the first band $l = 0$ is just a constant value, the second band $l = 1$ consists of linear functions and so on. Each band also consists of $2l + 1$ basis functions which means that there are a total of n^2 basis functions on the first n bands $l = 0, \ldots, n - 1$.

The spherical harmonics basis functions can be used to represent spherical functions. Those are functions that are defined on the unit sphere and can be evaluated for $f(s)$ where s is a point on the unit sphere according to equation 2. The more bands that are used to represent a function the more accurate that representation will be. It is not possible in practice to use an exact representation for a general function due to the infinite amount of basis functions. However, since the bands correspond to increasing frequencies it is possible to represent low frequency information reasonably accurately. This uses only the basis functions in the first n bands, where the value of n depends on the application. A representation like this is well suited for diffuse lighting and is exactly what is used in the light propagation volumes technique and in this thesis. In light propagation volumes a value of $n = 2$ is used. It thus uses a total of four basis functions[21].

The four basis functions for $n = 2$ expressed using Cartesian coordinates are[20]:

$$\begin{aligned} y_0^0(s) &= \frac{1}{2\sqrt{\pi}} \\ y_1^{-1}(s) &= -\frac{\sqrt{3}}{2\sqrt{\pi}} y \\ y_1^0(s) &= \frac{\sqrt{3}}{2\sqrt{\pi}} z \\ y_1^1(s) &= -\frac{\sqrt{3}}{2\sqrt{\pi}} x \end{aligned} \tag{3}$$

For each function $f(s)$ the spherical harmonics representation that best approximates it can be determined. For each basis function with $l < n$, a coefficient c_l^m is calculated. It indicates how well that basis function corresponds to that basis function. This means that the spherical harmonics representation for a function $f(s)$ using n bands is a set of n^2 such coefficients, one for each basis function. These coefficients are calculated as the integral of the product between the approximated function $f(s)$ and the currently evaluated basis function $y_l^m(s)$

over the unit sphere S:

$$c_l^m = \int_{s \in S} f(s) y_l^m(s) ds \tag{4}$$

This operation is a *projection*. In order to reconstruct the approximated function $\tilde{f}(s)$ or to evaluate it at a certain point s all of the basis functions are scaled by their corresponding coefficients and then summed:

$$\tilde{f}(s) = \sum_{l=0}^{n-1} \sum_{m=-l}^{l} c_l^m y_l^m(s) \tag{5}$$

So called *zonal harmonics* are spherical harmonics projections of functions which are rotationally symmetric around an axis. In each band among the spherical harmonics basis functions there is such a basis function that has rotational symmetry around the Z axis. These are the ones with the index $m = 0$ in each band. One property of zonal harmonics is that they only have one non-zero coefficient in each band. Zonal harmonics are also particularly simple to *rotate*[20].

An example of a function which will prove to be very useful is the *clamped cosine lobe* function. Because of the properties hinted above, it follows logically to choose a clamped cosine lobe *in the direction of the Z axis*. The function could be described as:

$$f(\theta, \varphi) = \max(\cos\theta, 0)$$

Since $f(\theta, \varphi)$ in this case is clearly invariant of the φ argument and only relies on θ, it is rotationally symmetric around the Z axis. It thus lends itself to a convenient zonal harmonics representation. Projecting this function into a spherical harmonics representation with $n = 2$ according to equation 4 gives the four coefficients $\left(c_0^0, c_1^{-1}, c_1^0, c_1^1\right) = \left(\frac{\sqrt{\pi}}{2}, 0, \sqrt{\frac{\pi}{3}}, 0\right)$. Here the factor $c_0^0 = \frac{\sqrt{\pi}}{2}$ is the non-zero coefficient of the first band while $c_1^0 = \sqrt{\frac{\pi}{3}}$ is the only non-zero coefficient of the second band.

The above representation is convenient and useful as long as it is sufficient to only represent clamped cosine lobes that are directed along the Z axis. This is however not enough for the light propagation volumes technique. As hinted earlier it is particularly convenient to rotate zonal harmonics functions such as the clamped cosine lobe above. Using rotation it is possible to extend the above result into a general spherical harmonics representation of a clamped cosine lobe in any arbitrary direction.

The method for rotating a general spherical harmonic is typically to use a transformation matrix of size $n^2 \times n^2$ where n is the amount of bands included in the representation. For the two first bands used in this case, that would result in a 4×4 matrix. As previously mentioned, rotation is more convenient for zonal harmonics. In order to rotate a zonal harmonic representation of a function, where the non-zero coefficient of band l is z_l, into the direction d, the spherical harmonic coefficients of the rotated zonal harmonic are given by:

$$f_l^m = \sqrt{\frac{4\pi}{2l+1}} z_l y_l^m(d) \tag{6}$$

The clamped cosine lobe pointing along the Z axis with spherical harmonic coefficients $\left(\frac{\sqrt{\pi}}{2}, 0, \sqrt{\frac{\pi}{3}}, 0\right)$ would then have $z_0 = \frac{\sqrt{\pi}}{2}$ and $z_1 = \sqrt{\frac{\pi}{3}}$. Inserting these values into equation 6 and setting $d = (x, y, z)$ gives the spherical

harmonics representation for a clamped cosine lobe pointing in the direction d:

$$\begin{aligned} f_0^0 &= \frac{\sqrt{\pi}}{2} \\ f_1^{-1} &= \sqrt{\frac{\pi}{3}}y \\ f_1^0 &= \sqrt{\frac{\pi}{3}}z \\ f_1^1 &= \sqrt{\frac{\pi}{3}}x \end{aligned} \tag{7}$$

A very important property of spherical harmonics is that they can be used to efficiently approximate the integral of a product between two functions over the sphere. This is done using just the dot product of the spherical harmonics coefficient vectors for those functions. Here the two functions are $f(s)$ and $g(s)$ and the coefficients of their spherical harmonics approximations are c_l^m and d_l^m, respectively.

$$\int_S f(s)g(s)ds \approx \int_S \tilde{f}(s)\tilde{g}(s)ds = \sum_{l=0}^{n}\sum_{m=-l}^{l} c_l^m \cdot d_l^m \tag{8}$$

A convenient operation for adding two spherical harmonic representations can also be defined. This is as simple as performing component-wise addition on all the coefficients.

It is especially convenient in computer graphics that the two lowest bands of a spherical harmonics representation consists of exactly four coefficients since that lends itself well to storage in regular four component vectors and RGBA color texels. This also allows for integrals on the form presented in equation 8 to be efficiently evaluated using the traditional dot product for these vectors.

3.2 Light propagation volumes

Light propagation volumes is, as already mentioned in section 2, a technique for approximating indirect diffuse lighting in a scene. What makes this technique interesting is that it is fast enough to run at real-time frame rates on current hardware. This section will give an introduction to the technique with details on how it works. The focus will be on the areas which are later modified and used in the implementation using octrees, which is covered in section 4. Additional detail on the technique can be found in the original papers of the light propagation volumes technique by [10], as well as the paper on cascaded light propagation volumes by [11].

The technique is divided into four distinct steps. Every step is executed, in order, each frame. This enables support for fully dynamic scenes and lighting. These steps will be described in detail later but a short summary of the tasks performed in each step is as follows:

Virtual point light creation - The first step consists of gathering information about geometry and reflected light. This is done from the viewpoint of the light sources. Each fragment that is visible from the viewpoint of a light source is considered as a *virtual point light* responsible for emitting the indirect light in the scene.

Light injection - The contribution from the virtual point lights in the previous step is injected into a three dimensional discrete grid. This grid covers the entire scene. The light is injected into the grid position nearest to the virtual point light.

Propagation - The light that was injected into the grid is then iteratively propagated through the grid. In each iteration the light in each grid element is blurred to all adjacent elements.

Rendering - After the desired amount of iterations in the propagation step have been performed, the resulting light distribution is sampled. For each rendered fragment the contribution of indirect lighting is evaluated using the light distribution in the grid.

The technique essentially provides the indirect lighting corresponding to the light's first diffuse surface bounce. The cascaded light propagation volumes technique additionally extends this technique in several ways. Some of these ways have already been mentioned in section 2. Apart from those it also introduces *fuzzy indirect occlusion* as well as support for *multiple indirect light bounces*[11]. Both of these additions rely on the use of a new, separate, three dimensional grid. This grid contains a discrete representation of the solid geometry in the scene instead of the lighting. This can then be sampled and taken into consideration during the propagation. That way the light can take scene geometry into account.

3.2.1 Virtual point light creation

Virtual point light sources are, unlike regular light sources, not part of the scene itself. However, they are still taken into consideration during rendering of the scene. In the light propagation volumes technique they are used to represent the light that is being reflected from diffuse surfaces. Every fragment that is directly hit by a regular light source is considered a virtual point light which emits light on its own. In a sense, each surface that is illuminated by a light source will then turn into an area light source. It is covered in many small virtual point lights.

The virtual point lights in the technique are represented as texels in a reflective shadow map. These store the distance from the original light source, the surface normal and the reflected light color and intensity. Knowledge of the location and the view of the original light source can then be used to position the virtual point light in world space. This allows the contribution to the rest of the scene to be evaluated.

Details of shadow mapping and reflective shadow mapping is outside the scope of this report, but is covered in [5, 7]. In short, shadow mapping renders the scene to a buffer using the render-to-texture capabilities of the graphics system. This rendering is done from the viewpoint of the light source. This is done prior to rendering the scene onto the screen from the viewer's camera. The scene is however not rendered in the traditional sense. Instead it writes the depth value, or the distance, from the light source into the buffer. This information can then be sampled during the rendering to the screen to determine whether a certain fragment is being directly lit by a light source or not. Reflective shadow mapping

writes additional information to this render-to-texture buffer. It especially stores the normal of the surface and the color of the reflected light.

As mentioned before, in the light propagation volumes technique each texel in such a reflective shadow map represents a small virtual point light, emitting its own light that should contribute to the scene. These reflective shadow maps are generated for each light source in the scene. It is done by utilizing multiple render passes. The reflective shadow maps are then used by subsequent steps of the technique. The next step is to *inject* the contribution of each of these virtual point lights into the three dimensional grid.

3.2.2 Light injection

At this point, there is a set of reflective shadow maps, originating from each light source in the scene. The information in the texels of these maps, along with information[2] about the original light source, is then used to pinpoint the world space position of each virtual point light. The light color, intensity and direction of the lights is also available in the reflective shadow maps.

The next step is to add the contributions of each virtual point light into a separate three dimensional grid. This grid is what is known as the light propagation volume. Its size is usually uniform in each dimension, $n \times n \times n$. Each element in this grid is a set of three vectors. Each vector has four components, representing the four coefficients of a *two-band spherical harmonics representation*. One vector stores the coefficients corresponding to the red color channel, one for the green and one for the blue.

Each element in the grid corresponds to a cuboid volume in the scene. This volume depends on the position of the element within the grid and on the extents of the grid itself. Given that information, each world space position can be mapped to an unambiguous grid element. This is assuming it is within the extents spanned by the grid. This is used in the light propagation volumes technique to map each and every virtual point light into their corresponding grid element. The details of this mapping are available in [10, 11]. The outgoing light distribution from each virtual point light is then projected onto a spherical harmonics representation for each color channel. These can then simply be injected into the appropriate grid element using component-wise addition of the spherical harmonics coefficient vectors.

The projection of a virtual point light into spherical harmonics coordinates is a two step process. The directional distribution of reflected light from a diffuse surface is here represented by a clamped cosine lobe around the normal of the surface. Given this normal, the spherical harmonics coefficients c for such a function is given by equation 7. This will however not take the light intensity or color into account, but just the direction of the reflected light. In order to properly preserve intensity and color, each resulting vector is scaled by the light flux Φ. This flux depends on the reflected light intensity I and the albedo A of the corresponding color channel. The flux also needs to be scaled by the so called *texel weight* w. This gives $\Phi = IAw$. The final vector injected into the grid is then given by $v = \Phi c$.

The texel weight w is an estimate of the ratio between the area of one texel of the reflective shadow map s_{rsm} and the area of the cut of a single grid element

[2] This information includes the position, orientation, field of view etc. of the light source.

s_{grid}. This gives

$$w = \frac{s_{\text{rsm}}}{s_{\text{grid}}}$$

In the case that the reflective shadow map is generated using an orthographic projection this can be calculated using a simpler approach. Assuming the extents of the reflective shadow map is approximately the same as the extent of the grid in that plane, then w is approximately the same as the quota between their respective texel counts. If the reflective shadow map is $t_{\text{rsm}} = m \times m$ texels and a cut of the grid is $t_{\text{grid}} = n \times n$, then the following applies:

$$w = \frac{t_{\text{grid}}}{t_{\text{rsm}}} \tag{9}$$

3.2.3 Propagation

The injection step ensures that there is lighting data in the grid elements that contain directly lit geometry. Every grid element that does not contain any directly lit geometry will contain zeroes. The propagation step lets the light propagate or spread into these zeroed grid elements. This is done using an iterative process. In each iteration every grid element is, in a sense, blurred with its neighbouring elements in the axial directions. This blurring function actually involves both evaluating the spherical harmonics function of a grid element and then re-projecting it into the adjacent grid element as a new virtual point light.

During each iteration, every grid element will be propagated to the six neighbouring grid elements in the axial directions using the blurring algorithm mentioned above. The propagation from a grid element $A = (a_0^0, a_1^{-1}, a_1^0, a_1^1)$ to a neighbouring grid element $B = (b_0^0, b_1^{-1}, b_1^0, b_1^1)$ actually consists of five spherical harmonics evaluations and projections. The spherical harmonic of A is evaluated in the direction of each of the outwards faces of B[3]. They are evaluated according to equation 5 where the basis functions are defined according to equation 3.

The basis functions are evaluated in the directions from the center of A towards each face of B as shown in figure 2, left. Each such evaluation gives a single valued result. This result is then scaled by the *solid angle* occupied by that face, as seen from the center of A, which is shown in figure 2, center. This is in turn used to scale the coefficients of a clamped cosine lobe function. The clamped cosine lobe is directed towards the current face, seen from the center of B, as shown in figure 2, right. The summed coefficients of every such scaled cosine lobe is then stored in B.

Another way of looking at it is that the coefficients of each grid element A will be the sum of $6 \cdot 5 = 30$ scaled clamped cosine lobe projections. Each of the six adjacent grid elements has five projections.

Snapshots of the grid contents after each iteration can be denoted S_i where $i = 0, \ldots, k$. The variable i denotes the i:th iteration, in the range $1, \ldots, k$. S_0 is the initial light distribution after injection. The upper limit k is the total amount of iterations performed during the propagation step.

The iterations are performed in such a way that the contents of S_{i-1} are used to generate the contents of S_i according to the propagation function explained

[3]The face facing back towards A is excluded.

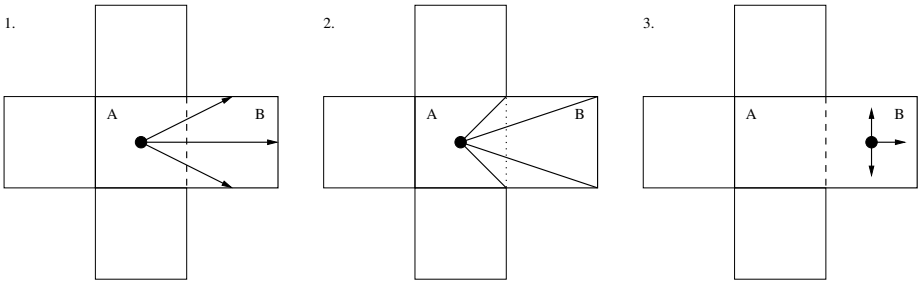


Figure 2: The three concepts used during the propagation between two grid elements A and B. In reality this is in three dimensions, but the figure is flattened to two dimensions for simplicity. The figure on the left shows the directions in which the spherical harmonics function of A is evaluated. The figure in the center shows the extents for which the solid angle is computed. The figure on the right shows the directions used when re-projecting the light.

earlier. Once all k iterations have been performed and all S_i are populated then the final light distribution, or the *accumulated light*, is given by:

$$S_A = \sum_{i=0}^{k} S_i$$

An example of some fictional such snapshots $S_0, \ldots, S_4, S_A$ are illustrated in figure 3.

3.2.4 Rendering

The final stage of the light propagation volumes technique uses the accumulated light buffer S_A to query the indirect illumination in the scene. This can be done during the render pass in which the direct lighting is applied. For each rendered fragment F, the position of that fragment is mapped to its corresponding grid element. The spherical harmonics coefficients in that grid element are then sampled and evaluated in the direction of the fragment's negative surface normal $-n$. This is illustrated in figure 4. In order to evaluate the spherical harmonics a clamped cosine lobe function in the direction of the negative normal $-n$ is projected into spherical harmonics coordinates. The resulting coefficients are then evaluated using equation 8 for each color channel. This yields a set of red, green and blue color intensities. These intensities are then scaled by the fragment's surface albedo and an optional scaling coefficient. Finally the resulting color is applied to the fragment by adding it with the direct lighting contributions.

3.3 Point light injection

The light injection method described in section 3.2.2 allows for the simplified calculation of the texel weight w from equation 9. This does however only work in the case of orthographic projections, where the imagined rays of light are parallel. Such a projection is well suited for modeling light sources far away, such as the sun. It is however not well suited for modeling local light sources

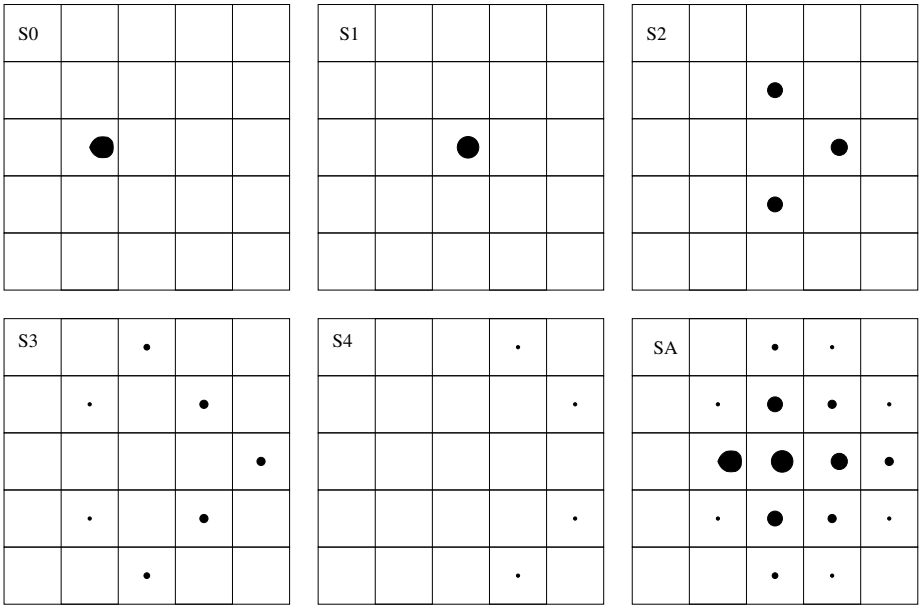


Figure 3: Snapshots of the light distribution during four iterations and the resulting final light distribution summed together in S_A. Initially, from the injection step, there is only a single grid element containing lighting. This is then propagated throughout the rest of the grid.

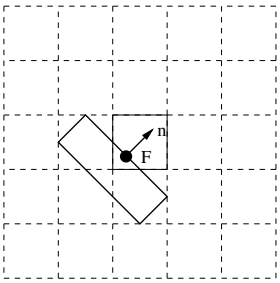


Figure 4: A fragment F with normal n of a rectangular object being rendered. The grid element to be sampled for this fragment is highlighted with a solid outline.

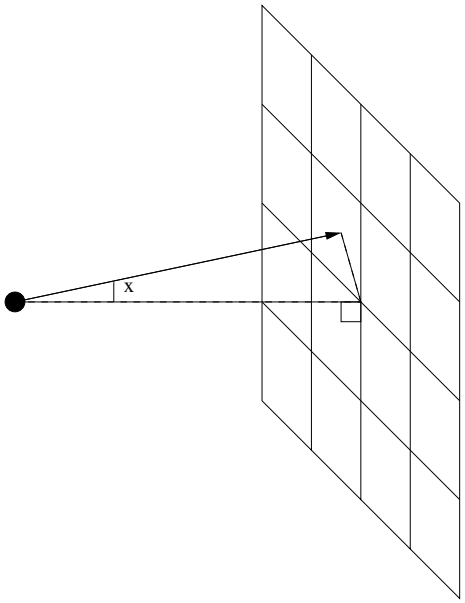


Figure 5: The view angle θ between the image plane normal and the view vector to a texel.

since the light rays from such a local light source will diverge. A perspective projection is then better suited. Such a projection does however make the texel weight dependent on distance and on the angle of the texel from the center. A texel that represents a surface further away will cover a larger area in world space. The same goes for a texel that is captured from an angle by not being in the center of the view.

The paper on subsurface light propagation volumes [19] presents a new method for injecting the contributions of point light sources into the light propagation volume. That method defines the flux Φ through a texel differently than in the original technique. It uses the solid angle Ω subtended by the texel from the light source. Specifically, the flux is defined as $\Phi = IA\Omega$. The solid angle does in turn depend on the field of view of the camera used to create the reflective shadow map as well as its resolution. It also uses the view angle θ, which is the angle between the normal of the imaging plane and the view vector to the texel. This angle is illustrated in figure 5.

Assuming that the aspect ratio of the camera that created the reflective shadow map is one, the solid angle Ω can be written as:

$$\Omega = \frac{4\cos^3\theta\tan^2\frac{\mathrm{FOV}}{2}}{N_x N_y}$$

In this expression N_x and N_y are the horizontal and vertical resolutions of the reflective shadow map, respectively.

4 Method

This section starts by introducing some of the methodology used to assess and evaluate the developed technique. Then follows, in section 4.1, a description of how related techniques were integrated. This includes the integration of the light propagation volumes techniques and the handling of omnidirectional light sources. Finally, in section 4.2, the octree data structure and the resulting technique are presented in detail.

This project is focused on achieving a practically applicable technique for indirect illumination. As such, a working implementation of the technique will be developed and maintained throughout the project. The details of that implementation are presented in section 5.

In order to establish a baseline for the comparison and evaluation of results an implementation is created, incorporating some of the underlying techniques used in this project. This particularly involves a system implementing the original light propagation volumes technique, using a scene that can effectively showcase the effects of the technique.

The baseline implementation will then be gradually extended with new functionality to support dynamic omnidirectional point lights and octree based light propagation volumes. This extended implementation is the system described in section 5.2.

The results from the project will be presented in two forms. The first consists of timing and performance data. The other is that of correctness, mainly through visual evaluation. If the resulting technique runs in real-time and specifically improves upon some aspects of the original technique while not introducing any severe visual artefacts then the project is successful.

The quality and any artefacts can conveniently only be visually evaluated. As such it is a highly individual metric. The basis for comparison in this case is the original light propagation volumes technique and the cascaded light propagation volumes technique. Part of the evaluation will be done against the baseline light propagation volumes implementation which is a part of this project. Another part will be the implementation of cascaded light propagation volumes by NVIDIA from [6].

In particular, the visual assessment of the resulting technique will be focused on artefacts such as temporal flickering and discretization of the indirect lighting.

Scene considerations Details of the scene and of how it is set up is outside the scope of this section and is rather presented in section 5.2. The choice of scene and what it should contain is however key in order to allow convenient evaluation of the resulting lighting. The primary scene chosen for this project is the Sponza Palace Atrium scene from [16]. It is a common choice for global illumination demonstrations and it allows the global illumination effects to show themselves in a realistic setting.

As a secondary scene for testing purposes a simple cube room with coloured walls was created. Such a scene allows for a very predictable resulting lighting and can be used to conveniently confirm the rendered result with the expected result.

The lighting is contributed by multiple point light sources inside the scene itself. This is a contrast to the typical distant directional light, usually from the sun. Such nearby point light sources are able to really test the dynamic nature

and flexibility of the technique. It does however also infer additional challenges as indicated by section 3.3.

4.1 Integration of existing techniques

Light propagation volumes Based on the available real-time techniques surveyed in section 2, light propagation volumes is the most promising technique for this project. The details of the technique itself are presented in section 3.2.

The light propagation volumes are in this application chosen with an extent sufficient to uniformly cover the entire rendered scene. This ensures that the indirect illumination will affect the entire scene and all objects in it. It also means that with an unchanged resolution the accuracy of the technique will be lower then if only a part of the scene had been covered. Fortunately the scenes in this case are relatively compact, making good use of the storage in the light propagation volumes.

Due to the power and efficiency of GPGPU computation in modern hardware, parts of the light propagation volumes technique were implemented using CUDA. Such computations were chosen to perform the light injection and light propagation in the place of shaders. Among other things this requires a different approach to light injection then the point based rendering used in the original papers [10, 11]. It also relies upon efficient sharing of memory between the GPGPU processing and the shaders.

Omnidirectional light sources An omnidirectional light source is a light source which emits light in all directions around itself. Shadow mapping, and by extension reflective shadow mapping, can be done for such light sources in a variety of ways. The most intuitive such way is to generate multiple separate shadow maps for each light source by performing multiple render passes. Assuming these shadow maps are generated using a field of view of 90° it requires six shadow maps for each light source and an equal amount of render passes. While straightforward, this is a quite performance heavy operation. It is however still the method chosen for this project due to its simplicity.

The generation of the reflective shadow maps is then done by using perspective projection. A possible way of injecting virtual point lights from such a reflective shadow map into the light propagation volumes has already been presented in section 3.3. That injection method is adopted for this technique as well.

Propagation schemes While the original paper on light propagation volumes describes a certain algorithm for light propagation for use in the technique, there are slight variations in different papers and implementations. The propagation scheme detailed in section 3.2 is the one presented in the *cascaded* light propagation volumes paper[11]. This is the propagation scheme that is used in this project. The reason for this, as opposed to using the propagation scheme associated with traditional light propagation volume, is that it has better theoretical coverage in its paper. This is also the propagation scheme used in the implementation of cascaded light propagation volumes by NVIDIA[6].

The main difference between the two archetypes of propagation schemes, the original one and the one from the cascaded light propagation volumes paper,

is that the original one does not evaluate the stored spherical harmonics function in the direction of each face as was demonstrated in figure 2. Instead it only evaluates it in the direction towards the adjacent grid element that it is propagating to.

The propagation scheme usually only makes itself known by the resulting shading of surfaces in the scene. This does not allow for convenient verification of the results in a general scene. In order to overcome this limitation the chosen propagation scheme, along with several alternative variations of it, were simulated in a non-real-time environment.

In this project, such an environment is implemented in a propagation simulator. Details of this simulator are provided in section 5.1. This is used to verify that the implemented propagation scheme can produce a realistic light distribution, not only along the geometry of a particular scene, but in general.

While the original light propagation volumes technique does not consider any indirect occlusion, cascaded light propagation volumes do. As briefly explained in section 3.2, this relies on maintaining a separate grid, representing the geometry of the scene. Such a representation was chosen to be included in this project as well since it provides a more realistic resulting lighting.

4.2 Octree light propagation volumes

This section covers both the representation of the octree data structure itself and how it is used to facilitate light propagation. There are essentially two important parts of the octree data structure. First and foremost are the *octree levels* themselves. These store the actual data, but have no information about the current structure of the octree. Then there is the *index volume*. It complements the octree levels by keeping track of how to index the octree levels and what structure the octree currently has.

Octree representation An *octree* is a tree based spartial partitioning data structure. It is a tree with a dedicated root node and where each node except the leaf nodes have exactly eight children. The root node represents, or *covers*, the entire space encompassed by the octree. Each of its children then covers one eighth of that space and so on. This continues recursively either until a certain depth is reached, the volume covered by a leaf node falls below a certain threshold or the contents of that volume fulfil some condition.

There are several ways to represent and manage an octree. One possible way to categorize them would be into pointer-based octrees and pointer-less octrees. In a pointer-based octree each node typically contains eight pointers, one for each possible child node. Such octrees are usually conservative with regard to memory usage but are prone to inefficient caching behaviour. Pointer-less octrees typically store nodes linearly in memory and relies upon some deterministic indexing or search pattern to access nodes in a tree. Each of these categories of octrees in turn have additional sub-categories.

One particular type of a pointer-less octree representation is a *full octree*[13]. A full octree, is required to allocate memory for all nodes that could possibly exist down to a pre-defined level. By ensuring that this memory is laid out linearly, it is possible to compute the index or address for any node in the tree. Such a representation is highly inefficient with regard to memory usage if the

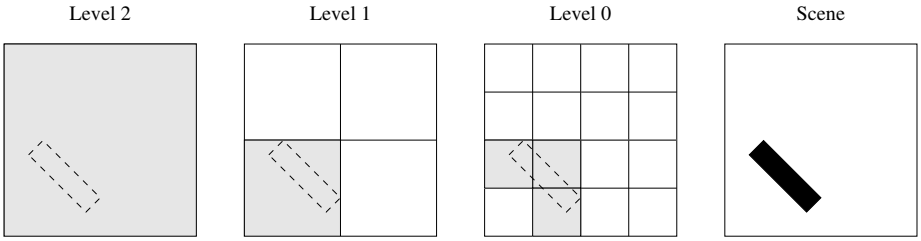


Figure 6: Two dimensional illustration of how levels of a full octree represented by multiple textures could correspond to certain areas of a scene. The highlighted grid elements are the ones that contain parts of the rectangle from the scene.

represented data is sparse. It is however suitable in cases where the larger nodes represent a more coarse representation of the data.

A full octree can be thought of as something similar to a mipmapping structure of a three dimensional texture. It can also be represented in memory, both on the GPU and the CPU as a set of such textures of different resolutions. This allows especially the GPU to optimize memory access to it as if it was a regular texture. An example of this in two dimensions is shown in figure 6.

Because of the ease of use and the very straightforward representation on the GPU a full octree representation has been chosen to replace the light propagation volumes from the original technique. This imposes an additional cost in GPU memory but has the benefit that each individual level of the octree is just a uniform grid which is relatively easy to work with. There will essentially be several overlapping light propagation volumes of different resolutions covering the same scene.

In order to simplify the description of these octrees a basic notation is proposed. An octree is assigned a size, or resolution n, which should be a power of two $n = 2^1, 2^2, \ldots$. Given an octree that corresponds to a $n \times n \times n$ light propagation volume, the highest resolution level of that octree also has the size $n \times n \times n$. This level is the *first* level, or level $i = 0$. There are a total of $l = \log_2 n + 1$ levels in such an octree, $i = 0, \ldots, l - 1$. Since every level of the octree is a three dimensional grid it can be indexed using three indices, $\hat{x}$, $\hat{y}$ and $\hat{z}$. Along with a level index i in which these indices are used they uniquely denote a single element in the octree. The indices are defined within the range $\hat{x}, \hat{y}, \hat{z} \in [0, \ldots, \frac{n}{2^i}]$. The notation $O_i(\hat{x}, \hat{y}, \hat{z})$ denotes the grid element on octree level i at $(\hat{x}, \hat{y}, \hat{z})$.

In addition to the data storage levels themselves the technique also needs a way to know which level of the octree to sample for each world position. In the original technique it was straightforward to map a world position to its corresponding grid element according to the description in section 3.2.2. In the full octree representation one single position in world space maps to multiple grid elements in the octree, one in each level of the tree. It will only make sense to use one of these for the computations.

As a complement to the full octree representation a separate *index volume* can be introduced. The index volume would store integer indices instead of spherical harmonics. This volume is denoted I. It could be represented as just

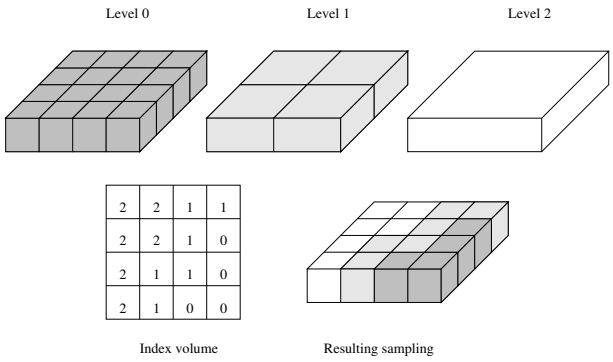


Figure 7: A two dimensional slice of a flattened index volume, three levels of a full octree and the resulting sampling.

another three dimensional structure of the same size n as O_0. Unfortunately, such a representation requires some quite inefficient operations to be added to the propagation step. In order to avoid this, a hierarchy of such structures can be used instead. These are laid out in the same way as the data storage structures themselves and can be denoted I_i where $i = 0, \ldots, l-1$, just as for the data itself. It is also indexed using the same coordinates (x, y, z) as the data. Each element in the index volume is an integer index $I_i(x, y, z) = 0, \ldots, l-1$, originally initialized to $l-1$. This index is the number of the octree level to sample for a particular volume in world space. It will be populated with values during several of the steps performed throughout the technique. Details of this will follow later. During the sampling, the world position is then mapped to an element in the *index* volume. The value in the index volume is then used to do a second mapping into the correct octree level. The complication with this representation is that there are l index volume elements that could be picked for each world space position, one from each level i. To solve that the element containing the minimum value is used.

An index volume I with the levels I_i, $i = 0, \ldots, l-1$ where I_0 has size $n \times n \times n$ is considered to have size n. It can be indexed using the coordinates (x, y, z) where $x, y, z \in [0, \ldots, n-1]$ according to:

$$I(x, y, z) = \min_i I_i \left(\frac{x}{2^i}, \frac{y}{2^i}, \frac{z}{2^i} \right)$$

The resulting sampling in the octree corresponds to $O_{I(x,y,z)}(\frac{x}{2^{I(x,y,z)}}, \frac{y}{2^{I(x,y,z)}}, \frac{z}{2^{I(x,y,z)}})$. This is illustrated, using a flat visualization of the index volume, in figure 7. Such a sampling is additionally given the simplified notation:

$$S(x, y, z) = O_{I(x,y,z)}(\frac{x}{2^{I(x,y,z)}}, \frac{y}{2^{I(x,y,z)}}, \frac{z}{2^{I(x,y,z)}}) \tag{10}$$

Octree light propagation volumes With the proposed octree representation there would be one such octree containing the lighting distribution. Each element in each level of the octree would contain a set of three spherical harmonics vectors, one for each color channel, just as in the original technique. It

would also be possible to extend the technique by adding an equivalent octree structure to be used for occlusion that works in the same way as in the cascaded light propagation volumes technique.

There are several possible ways to use the proposed octree structure to replace a light propagation volume. In this project it was chosen that the propagation of the injected light should be performed individually on each level of the octree. This allows the propagation scheme to be used almost without any modifications in the octree representation. On first sight this may seem wasteful compared to just propagating on the highest resolution level, as in the original technique. However, the elements of the other levels in the octree cover a larger volume in world space. As a result the light on those levels will propagate further in fewer iterations. This allows the more detailed levels of the octree to be used to light the parts of the scene close to where the light was first injected. This will often be the areas where that light still has high enough intensity to be clearly visible. On the other hand, parts of the scene further away from this reflected light are unlikely to be hit by much indirect lighting. Because of that, those parts can be safely lit using a more coarse approximation of the indirect lighting. This is the main idea explored in this project.

The same four steps that were performed in the original technique would need to be extended to work correctly for the new octree representation. Luckily the first step, the creation of the reflective shadow maps, is already completely independent of the actual light propagation volume. As such it does not have to be changed at all. The other three steps do however need to be extended accordingly.

Light injection and downsampling To perform propagation individually in each level of the octree it is required that each such level is first injected with light. All the levels of the octree are just variously accurate or coarse-grained representations of the light distribution. As such there will never be a need for a more detailed representation than the highest resolution level O_0. As a first step the light is thus injected into O_0 in the exact same way as in the original technique.

Injecting the light into O_0 leaves the other levels of the octree empty. All the necessary data can however be derived from the data in O_0 since all other levels are to be less detailed. Each element $O_i(x, y, z)$, $i = 1, \ldots, l-1$ will cover exactly the same volume in world space as a specific set of eight elements from O_{i-1}. These elements are $O_{i-1}([2x, 2x+1], [2y, 2y+1], [2z, 2z+1])$. In order to maintain a uniform range of light intensities in all octree levels, each element in O_i are populated by simply averaging the corresponding eight elements from O_{i-1}:

$$O_i(x, y, z) = \frac{1}{8} \sum_{\hat{x}=2x}^{2x+1} \sum_{\hat{y}=2y}^{2y+1} \sum_{\hat{z}=2z}^{2z+1} O_{i-1}(\hat{x}, \hat{y}, \hat{z}) \tag{11}$$

The process of populating the layers $i \neq 0$ using the above averaging formula is called the *downsampling step*. It is considered as a separate step since the it is done after, and completely without any interaction with the injection step.

The index volume is updated during both the injection and downsampling step. During the injection step, for each element $O_0(x, y, z)$ that receives a non-zero contribution it follows that the element in question is close to scene ge-

ometry which is diffusely reflecting light. It is also clear that for these elements there is no chance of there being an even more accurate representation of the light in that area since O_0 is the highest resolution level. Because of that the index volume can immediately be assigned zeroes for all those elements in I_0, $I_0(x, y, z) = 0$. This means that after the injection step the index volume will give 0 for all elements where $O_0(x, y, z) \neq 0$ and $l - 1$ for all other elements. Other values are filled in later.

When it comes to maintaining the index volume during the downsampling step, each element $O_i(x, y, z)$ has a corresponding element in the index volume $I_i(x, y, z)$. For each such element $O_i(x, y, z)$ that is assigned a non-zero value during the downsampling, the index volume is updated to $I_i(x, y, z) = i$.

Propagation After the injection and downsampling steps the octree contains an initial distribution of the diffuse reflected light in the scene. The index volume generally gives lower values close to geometry and higher values in empty areas. The next task is to allow this initial light distribution to propagate throughout the scene. As previously mentioned, this is done individually on each level of the octree in the same way as the single level was propagated in the original technique. This means that each of O_i, $i = 0, \ldots, l-1$ is considered as a standalone light propagation volume of size $\frac{n}{2^i}$. The propagation then proceeds according to section 3.2. Specifically, if the propagation function giving the contribution to $O_i(x, y, z)$ is denoted $P(x, y, z)$, a scaling factor p called the propagation factor can be introduced to allow tweaking of the propagated light intensities according to:

$$O_i(x, y, z) = p \cdot P(x, y, z)$$

The amount of iterations performed during the propagation step can be denoted as k. In the original technique a single light propagation volume of size $n \times n \times n$ was used. The worst case in such a light propagation volume requires at least $k = n$ in order for light injected in one end of the volume to reach the other end. Luckily the light often dies out before that, so in practice a lower value can be used. With the octree representation however, right after the downsampling, O_{l-1} will already be propagated since it is just a single $1 \times 1 \times 1$ volume. O_{l-2} will only need a single iteration in order for light to propagate through that entire level. In the same way, O_i will need a total of $k = 2^i - 1$ iterations for light to propagate throughout the entire level. Of course, the more iterations are performed, the more complete O_0, which is the most accurate, will be. However, no matter what value of k is used, there will always be at least one level, often more, of the octree where the propagation is completed. These are $i = l - 1 - \lfloor \log_2 k \rfloor, ..., l - 1$. The higher detailed levels $i = 0, \ldots, l - 2 - \lfloor \log_2 k \rfloor$ can only be guaranteed to have a partial coverage of propagated light.

When it comes to the levels where the propagation is guaranteed to provide a complete coverage there is no good reason to sample from any level but the most detailed of them. This is $i = l - 1 - \lfloor \log_2 k \rfloor$. The index volume should reflect this and help to ensure it. As such, after k iterations, $0 \leq I(x, y, z) \leq l - 1 - \lfloor \log_2 k \rfloor$ should hold for any point (x, y, z). For the octree levels that do not guarantee complete coverage they have one thing in common, they are of higher resolution than the ones that do. As such, wherever they do provide coverage they should be used rather then the lower resolution ones.

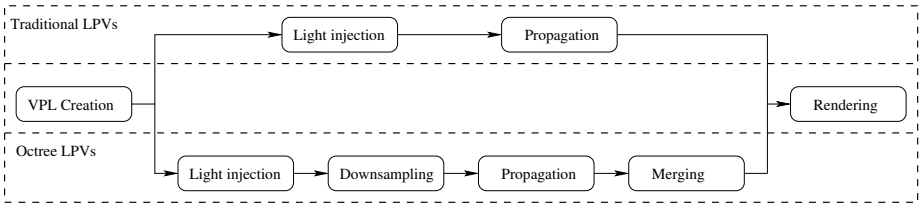


Figure 8: An overview of the steps included in the octree based and the traditional light propagation volumes technique. The steps are divided into those that are the same for both techniques and those that need to be implemented specifically for one particular technique.

In order to satisfy the above requirements during the propagation the index volume is continuously updated. Whenever the propagation scheme adds a non-zero contribution to an element $O_i(x, y, z)$, the index volume is updated so that $I_i(x, y, z) = i$. This simple procedure will ensure that the index volume fulfils the requirements above after each completed iteration.

Merging and rendering After the propagation the scene can be rendered as in the original technique with the exception that the sampling scheme from equation 10 must be used. This is however not practical since it would require the rendering pipeline to have access to and to be able to read both all the levels of the octree O_i and all the levels of the index volume I_i. To avoid this an additional *merge step* is introduced. It stores all the sampled values using $S(x, y, z)$ into a single traditional $n \times n \times n$ light propagation volume M where:

$$M(x, y, z) = S(x, y, z)$$

Once the merge step has been completed the result is just a single light propagation volume that works in the exact same way as in the traditional technique. This also makes it possible to use the exact same rendering step as in the traditional technique.

The steps involved in this technique compared to the original technique are illustrated in figure 8.

5 System description

Two separate systems were implemented as part of this project; one offline simulator and one real-time implementation. The simulator implements a number of variations of the propagation scheme for the octree light propagation volumes technique. It also visualizes the light distribution throughout the simulation. The real-time implementation loads and renders a scene using the octree light propagation volumes technique and allows various options to be enabled, disabled and tweaked during runtime. This offers immediate visual feedback of the changes.

5.1 Light propagation volume simulator

The light propagation volume simulator is a standalone application written in the Java programming language. It implements all of the steps involved in the octree light propagation volumes technique, but without the real-time requirement. The steps are implemented as separate actions which can be applied individually in any combination and in any order. The result after each performed step is visualized in the form of a two dimensional grid, representing a slice through the underlying three dimensional structure. Propagation iterations are applied one by one as well, making it possible to visualize how the light propagates in discrete steps. There is also an action for performing all the steps of the algorithm in their natural order, effectively running an *automated simulation*.

Apart from the size of the octree used in the simulation there are three main aspects of the simulation that can be customized during runtime. These are the light injection, propagation and indirect occluders. The light injection configuration consists of an editable set of *injectors*. An injector is an entity that describes which elements of the octree's first level that should be assigned which values during light injection. They can be of several types such as *wall injectors*, *single element injectors* or *file injectors*. For some of these it is possible to specify a multiplier of the injected light and the direction of the clamped cosine lobe used to create the injected spherical harmonics.

The propagation is configured using one of the implemented *propagators* along with a custom propagation scaling factor. In the case where an automated simulation is used it is also possible to customize the amount of iterations used during the propagation. There are several propagators implemented. They are based on various papers and implementations, some of which are more accurate and realistic than others. The main propagator which has been used the most throughout the project is the propagator based on NVIDIA's implementation of cascaded light propagation volumes from [6].

The user interface of the simulator displays three separate grids. One is used as the source for propagation, one is used as the target and one is the sum of all propagations so far. The source and target grids alternate between being the source and the target after each iteration. The elements in the grids are two dimensional visualizations of spherical harmonic functions for a single color channel. The visualizations are created by sampling the spherical harmonic function, according to equation 5, at evenly spaced positions along the $z = 0$ plane of a unit sphere. These samples are then scaled and rendered as a vertex of a two dimensional polygon shape. The vertex is positioned a distance from

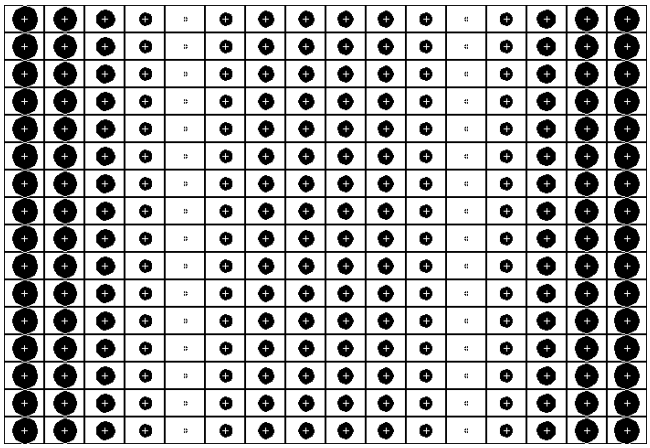


Figure 9: The grid used for visualizing light propagation volumes in the simulator. The light distribution comes from a pair of wall injectors, one on either side of the grid. It also showcases one of the artefacts of the technique which will be covered in more detail in section 6.1.

the center of the element equal to the scaled sampled value in the direction that was used to evaluate the function in the first place. One example of a rendered grid is shown in figure 9.

The scaling mentioned above can be adjusted by the user during runtime. The user can also select which level of the octree that should be displayed at any time during the simulation, or if the merged result should be displayed. The control panel of the user interface for the simulator is displayed in figure 10.

The simulator also has the capability of comparing the resulting octree values $O_i(x, y, z)$ with what they would have been if the propagation had been performed on O_0 and *then* downsampled. This can be thought of as a simple error metric. The average and maximum error across all the elements in the octree are made available in a plot such as the one shown in figure 11. Both the absolute and relative errors are calculated using the infinity norm of the spherical harmonics vectors.

There is one important difference between the simulator compared to the technique described in section 4. Instead of a hierarchy based index volume with multiple levels I_i, the simulator only uses a single three dimensional array to represent the resulting values of $I(x, y, z)$ directly. This is managed in such a way that it makes no difference to the end result but it simplifies the implementation of the visualization.

5.2 Real-time implementation

The real-time implementation is another standalone application that, unlike the simulator from section 5.1 attempts to utilize the available graphics hardware efficiently to perform the steps involved in the technique. It also produces an actual rendered scene in real-time. It is developed using the C++ programming language and uses the external libraries OpenSceneGraph and osgCompute for rendering and for GPGPU computations, respectively. It uses the OpenGL shad-

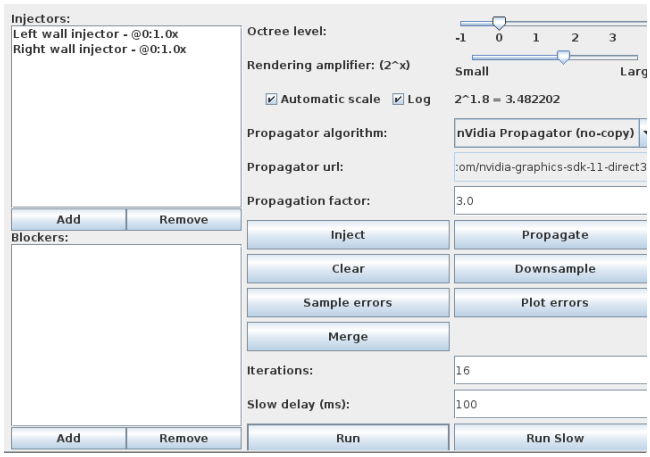


Figure 10: The control panel of the simulator.

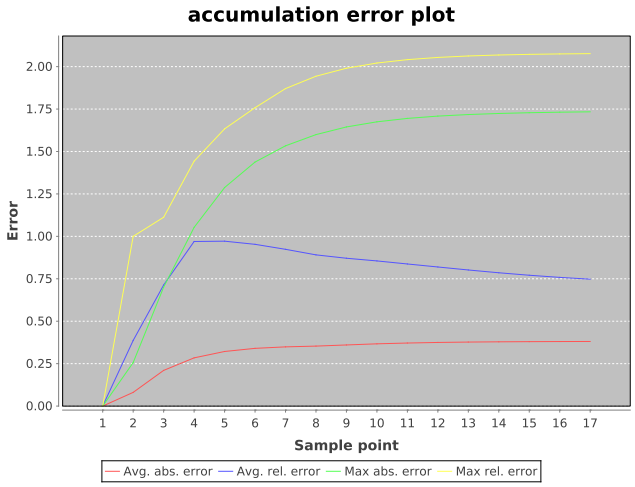


Figure 11: The error plot of a complete simulation with 16 iterations.

ing language to implement the final rendering step and for the creation of the reflective shadow maps. All other steps of the technique are implemented using CUDA.

Scene graph structure The OpenSceneGraph library represents scenes using a scene graph structure. This structure includes not only the geometry itself but also light sources and cameras. It also contains per-node configuration of rendering state such as shader programs and their uniforms. With the addition of the osgCompute plugin, the scene graph also contains GPGPU computation nodes. These can be set to execute at certain times in relation to parent and child nodes.

The scene graph setup used in the real-time implementation of the octree light propagation volumes technique is not trivial. It is built according to the structure in figure 12. The root of the scene graph is the *root camera*. This is the camera that ultimately renders the scene to the screen. It renders a single node called the *root node*. This root node then splits up into multiple branches. Some of these branches are just there to provide the HUD or some optional debug visualizations. Details of those branches will be left out in this description since they do not affect the actual technique. There are however two branches that are important. The first one is the *pre-render branch*, which is responsible for performing all the steps of the technique from the creation of the reflective shadow maps to the merging step. Note that it does not perform the actual rendering. That is done in a separate branch, the *rendering branch*. Both these branches eventually end up in the *main scene node* which contains the loaded Sponza model.

The pre-render branch will be processed using a separate traversal pass before the rendering to the screen. It contains a chain of `osgCompute::Computation` nodes, each of which corresponds to a distinct step in the technique. There is also an additional such node responsible for clearing and resetting all of the data structures to their initial state. These computation nodes are chained in reverse order. They use the `osgCompute::Computation::PRERENDER_AFTERCHILDREN` option. This will cause them to be invoked in the correct order during traversal. After the chain of computation nodes there is a common group node called the *pre-render root node*.

The pre-render root node has one child node for each light source in the scene. These are called the *light source root nodes*. The light source root nodes each have six cameras attached to them called the *pre-render cameras*. The pre-render cameras are the cameras which are configured to render to texture, creating the reflective shadow maps. They each have the `osg::Camera::PRE_RENDER` render order set. All of these pre-render cameras, from all light sources, then share one child node called the *pre-render scene node*. The pre-render scene node provides some shared configuration such as shader programs and uniforms that are needed for creating reflective shadow maps. The pre-render scene node has only one child node, the main scene node, which is shared with the render branch.

The rendering branch starts with a utility node called the *real scene node*. This node holds all of the configuration needed by the final rendering pass and its shader. It has three child nodes, one of which is the main scene node, the one that is shared with the pre-render branch. The other is a group node called

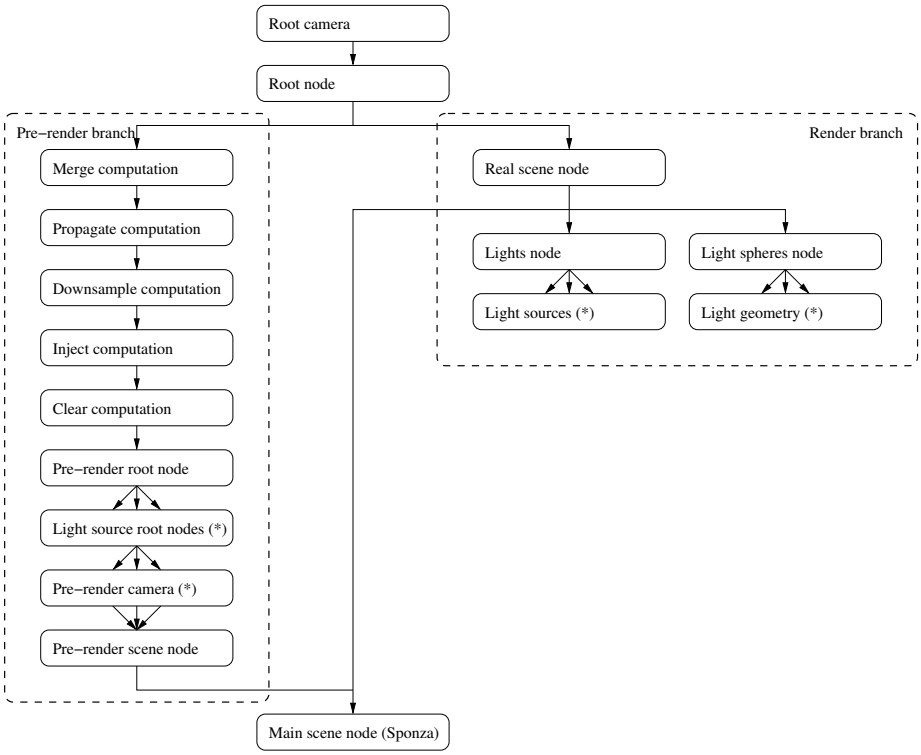


Figure 12: Diagram showing the structure of the scene graph used in the real-time implementation. Non-essential branches have been omitted. Nodes marked with a star (*) and diverging incoming arrows may exist in multiple copies for each parent node.

the *lights node*. The lights node in turn has all of the light sources in the scene as its children. The last child node of the real scene node is another group node called the *light spheres node* which contains the visual representations of the light sources, their geometry.

Pre-render setup The scene graph structure of the pre-render branch has already been described but without any details on how different parts fit together and interact with each other. The data structures in the implementation use a utility *volume* structure. It is actually just a combination of a three dimensional texture and its corresponding GPU compute memory. The texels have the format of RGBA values where each component is a 32-bit floating point value. This allows for convenient and accurate storage of four component spherical harmonic vectors. These textures have the same size along each dimension. A *hierarchy volume structure* is a hierarchy of volume structures, represented as a vector or array of volumes. The first volume has the full size $n \times n \times n$ and each successive volume is one eighth the size of the previous one with the size halved in each dimension. The *RGB volume* and *RGB hierarchy volume* are simple a set of three volumes or hierarchy volumes, respectively. One for each color channel. Essentially, an RGB volume is just three regular volumes, one for each color channel. The data structures based on these structures or used with them are:

Geometry volume - A geometry volume is one hierarchy volume structure, the *source buffer*, coupled with a single regular volume structure, the *target buffer*. The reflective shadow maps are processed one by one. Each is injected as geometry into the target buffer after which these new contributions are merged into level zero of the source buffer. Once level zero of the source buffer contains the contributions of all light sources it is downsampled so that the other levels are also populated.

Index volume - The index volume is a hierarchy of byte arrays. The array at level zero has size $n_0 \times n_0 \times n_0$ where n_0 is the size of the light propagation volume structure. Each successive level has $n_i = \frac{n_{i-1}}{2}$.

Light propagation volume - The light propagation volume itself is a collection of three RGB hierarchy volumes. These are the *source buffer*, *target buffer* and the *accumulation buffer*. The source buffer and target buffer swap between each iteration during the propagation. New light from the reflective shadow maps is injected into the target buffer. The accumulation buffer contains the sum of the contents of the target buffer between each iteration.

Reflective shadow map - The reflective shadow map is a set of three cube map textures[4]. These contain the encoded depth values, reflected color and surface normals, respectively.

All of the memory for these data structures is entirely contained within the GPU memory and processed using OpenGL shaders and CUDA kernels. Since it can remain there throughout the entire technique, the potential performance issues of transferring it back and forth between memories are minimized.

[4]These are actually implemented as three dimensional textures with each slice representing one side of the cube map.

The reflective shadow maps are rendered using render-to-texture cameras. Each light source has six such cameras associated with it, one in each direction of each axis, `X`, `Y`, `Z`, `-X`, `-Y` and `-Z`. They each use a perspective projection with a 90° field of view. Each such camera has three FBOs attached to it, one for each of the three components in the resulting reflective shadow map. Each of the six cameras writes to a different slice of the attached FBO three dimensional texture, or equivalently, to a different side of the attached FBO cube map texture. They all use a specific shader program during rendering that is responsible for encoding the light distance, reflected light color and surface normal into the three FBOs.

The clear computation node contains thee computation modules. The first two are responsible for setting all the memory used by the light propagation volume and the geometry volume to zero. Note that only the first level O_i of the light propagation volume's and the geometry volume's target buffers needs to be cleared. The other levels are completely overwritten during downsampling and the accumulation buffer is overwritten during the first iteration of the propagation step. The last one resets the index volume so that each element points towards the last level of the octree structure O_{l-1}.

The inject computation node contains two injection computation modules for each light source in the scene. The first one injects the initial light distribution itself and the second one injects the geometry. Each geometry injection module additionally merges the source and target buffers of the geometry volume and then clears the target buffer again, making it ready for injection from the next light source.

The downsample computation node contains two downsampling modules, one for the light propagation volume and one for the geometry volume. Following the downsample node is the propagation computation node which contains a single propagation module. It is responsible for performing all of the iterations of the propagation.

Finally, the merge computation node contains a single merge module. It merges the levels of the accumulation buffer. This is done two levels at a time. O_i is merged into O_{i-1} for $i = l - 1, \ldots, 1$. After all the merges the values for sampling using $S(x, y, z)$ will all be stored in $O_0(x, y, z)$. As such, only O_0 needs to be shared with the OpenGL shader program during rendering.

The scene is finally rendered to the screen by traversal of the render branch. During this process one shader program is used to combine the contributions of direct lighting, shadows, indirect lighting and bump mapping. The direct lighting is performed using traditional phong lighting, taking the bump maps of the surfaces into account. This is then scaled by a shadow factor determined by using percentage closest filtering from the depth component of the reflective shadow map with a 2×2 kernel. The indirect lighting is determined by sampling the merged accumulation buffer after the merge step. It is sampled by mapping the world position of the fragment into the best matching grid element. The spherical harmonics function of that grid element is then evaluated for the opposite direction of the surface normal.

In order to achieve a visually pleasing result the reflective shadow maps have a resolution of 1024×1024. This high resolution is however only used for the shadow mapping. The injection into the light propagation volume effectively only use 256×256. It is sampled only once in each 4×4 region. This allows for high quality shadows while maintaining reasonable performance during light

injection.

The light propagation volume itself has a size of $32 \times 32 \times 32$. The other levels of the octree then have the sizes $16 \times 16 \times 16$, $8 \times 8 \times 8$, $4 \times 4 \times 4$ and $2 \times 2 \times 2$. Note that the last level, $1 \times 1 \times 1$ does not exist in this implementation. This is because there were issues allocating a three dimensional texture with only a single texel in it. The rest of the technique does however work as previously described.

Finally, the injection computation module in this implementation is not the central part of the technique and as such it has only been implemented in a very simple way by using component-wise atomic write operations to global GPU memory. This is unlike the original technique which suggests implementing it using point based rendering instead.

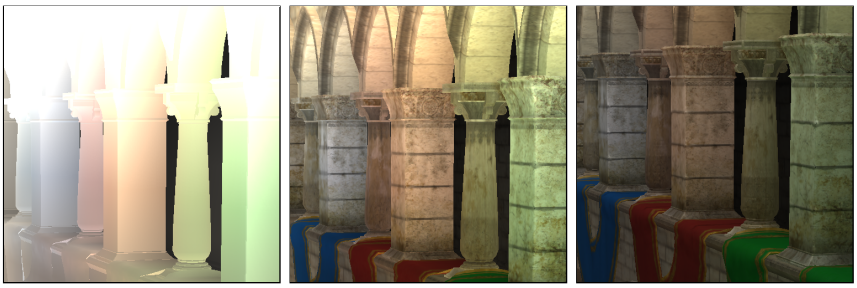


Figure 13: Samples of rendered images using the real-time implementation. The leftmost image shows only indirect illumination without any textures. The center image shows indirect illumination but with textures. The rightmost image shows the final rendering with both direct light, indirect light and textures. Note that the effect of indirect lighting has been slightly exaggerated in these images. Full size versions of these images are available in appendix A.

6 Results

The project has yielded a wide range of results. These will be presented in this section with focus on the quality of the resulting rendering, the correctness of the light distribution, and on the performance characteristics of the real-time implementation.

The final version of the real-time implementation has been manually tuned both based on visuals, performance and simulated results. This version uses a total of $k = 4$ iterations. As already mentioned, the octree has a size of $n = 32$ and the reflective shadow maps have resolution 1024×1024 while the effective resolution with regard to the light propagation volumes technique is 256×256. The propagation factor is set to $p = 3$.

6.1 Quality and artefacts

In terms of visual quality and visual artefacts, the octree light propagation volumes technique is mostly equivalent to the original light propagation volumes technique. It shares many of the same strengths and weaknesses. Rendered imaged from the real-time implementation are shown in figure 13. The full size versions of these images are available in appendix A. There are however some differences that are either directly visible in the rendered result in some situations or that can be seen using the simulator.

Spatial intensity jumps When two adjacent elements sampled using $S(x, y, z)$ and $S(\hat{x}, \hat{y}, \hat{z})$ are sampled from different levels i from the octree such that $I(x, y, z) \neq I(\hat{x}, \hat{y}, \hat{z})$ there are sometimes discrete *jumps* in the light intensities. This especially means that it is sometimes possible for light that has almost died out completely to suddenly regain some intensity. This can cause visual artefacts in situations where this happens while the involved intensities are still big enough to be visible. This artefact is illustrated using the simulator in figure 14. The effect in that figure is exaggerated and in practice it is usually not noticeable.

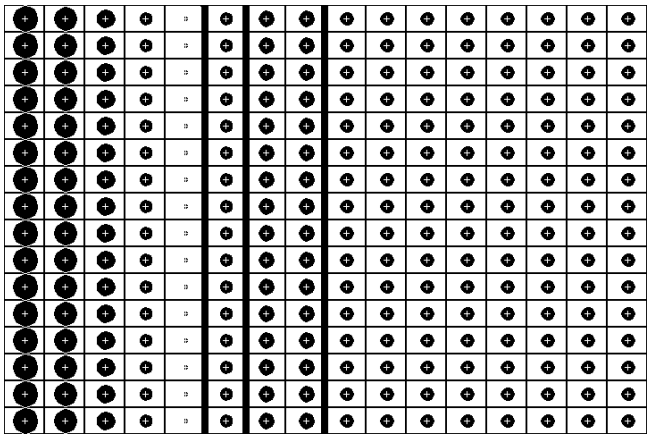


Figure 14: An illustration of the intensity jumps caused by octree level switching. Borders between levels are indicated by bolder lines. The effect is exaggerated in this figure by using a logarithmic scaling function during rendering a and specifically chosen propagation scaling factor.

Temporal intensity jumps Because of how the index volume is maintained during propagation the level sampled for $I(x, y, z)$ can depend on where the closest directly lit geometry in the scene is. In particular, when geometry is moving in certain ways or the light sources are moving around in the scene it is possible that $I(x, y, z)$ gives different results in one frame then it did the previous frame. While the transition between two adjacent grid elements within one octree level are usually quite smooth, in part thanks to linear filtering between the texels, that is not necessarily true for elements on different levels of the octree. When these conditions occur it is possible to notice that certain surface areas suddenly appear slightly brighter then the previous frame. This artefact is unfortunately not easily made visible in a figure.

Convergence using few iterations After each iteration during the propagation the overall intensities in the result are lower than in the previous iteration. This means that after a number of iterations, the contribution of performing additional iterations is negligible. At this point it can be said that the light distribution has *converged*. Since it takes very few iterations for light to propagate throughout the lower resolution levels of the octree, light may have propagated throughout the entire scene without having converged yet. This is however mostly a theoretical problem. In the real-time implementation there are no clear improvements when rendering with more than approximately four iterations.

Light propagation distance In the original technique, with the propagation scheme that is used in this project, the light intensities rapidly decrease after each iteration. Because of that there is a practical upper limit of how far the light can propagate in the light propagation volume before it has died out. The exact distances depend on the scaling factors that are used during the propagation. It is however rare for light injected in one part of the scene to affect the lighting

distribution far away even if there is nothing in between to stop the light. Using the octree representation there can however be a slightly higher light intensity stored in the lower resolution levels of the octree. This is because it is just the average of the light intensities in the entire scene. Luckily, these levels of the octree essentially work like a directional ambient light for the entire scene or large parts of it. The light intensities themselves also tend to be small enough not to look unnatural in practice.

Comparison to the original technique The visual quality in practice is approximately equal to that of the original light propagation volumes technique. As described above there are some specific artefacts that have been introduced by the use of octrees, but those are only distinguishable in very specific situations. The comparison has been made against both the implementation of light propagation volumes created during this project and the implementation of cascaded light propagation volumes created by NVIDIA[6].

6.2 Correctness

Error measurements There are a variety of ways to measure errors of the type of data contained in a light propagation volume. In this project the error is measured as the average pairwise error between two spherical harmonics vectors a_e and b_e. The infinity norm $||v||_\infty = \max_i v_i$, or max norm, of the vectors is used to calculate the errors. Both the absolute error $||a_e - b_e||_\infty$ and the relative error $\frac{||a_e - b_e||_\infty}{||a_e||_\infty}$ is considered.

Each element in the octree can be denoted by four indices, the octree level i and the coordinates within that level (x, y, z). For each such element $i \neq 0$ it is possible to calculate an averaged value from the corresponding elements from level $i - 1$ according to the downsampling in equation 11. The downsampled value can be denoted $\hat{O}_i(x, y, z)$. There are then two error metrics, the average absolute error e_{abs} and the average relative error e_{rel} of all elements in the octree $i \neq 0$. They are defined according to:

$$
\begin{aligned}
e_{\text{abs}} &= \frac{7}{8^l - 8} \cdot \sum_{i=1}^{l-1} \sum_{p \in O_i} ||O_i(p) - \hat{O}_i(p)||_\infty \\
e_{\text{rel}} &= \frac{7}{8^l - 8} \cdot \sum_{i=1}^{l-1} \sum_{p \in O_i} \frac{||O_i(p) - \hat{O}_i(p)||_\infty}{||O_i(p)||_\infty}
\end{aligned}
$$

In the above expressions, $\frac{7}{8^l - 8}$ is the inverse of the combined number of elements in $O_1, \ldots, O_{l-1}$, derived from the geometric sum, and $p = (x, y, z)$.

It would certainly be possible to use other error metrics as well. For example, by comparing $S(x, y, z)$ with the contents of a light propagation volume from the original technique.

The two metrics e_{abs} and e_{rel} are the primary error measurements calculated by the simulator. It calculates the values for both of them after each iteration during the propagation. Figure 15 shows the errors for a simulation closely corresponding to the situation in the real-time implementation. This shows that the average relative error generally increases steadily up until approximately 20 iterations. After that it is essentially constant. This is simply because there

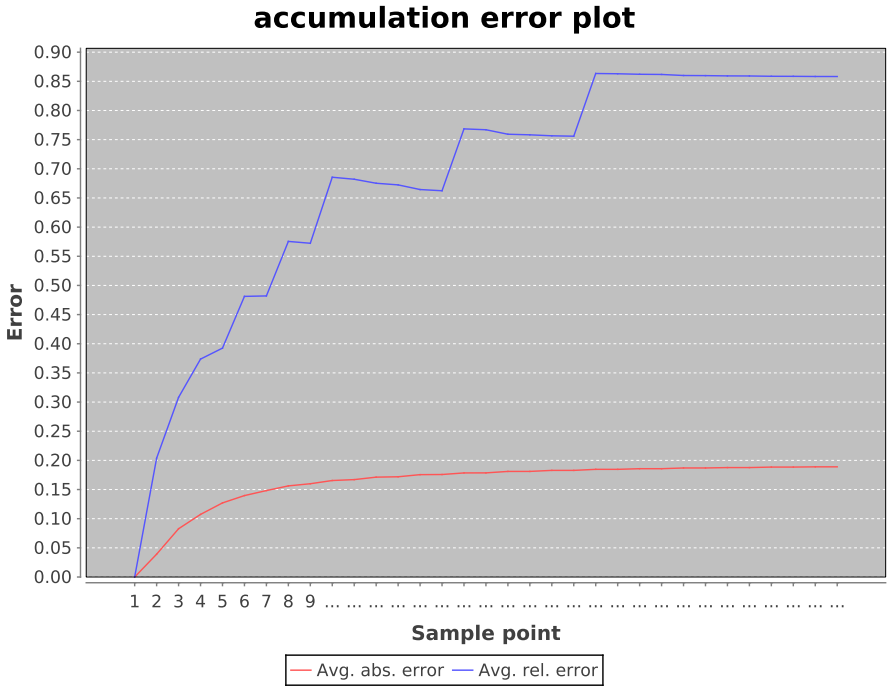


Figure 15: The errors measured after each iteration during a simulation. Both the average absolute errors and the average relative errors are presented.

is no noticeable light intensity left in the buffers at that point. This causes the accumulation buffer to remain the same after each iteration. The average absolute error exhibits a less erratic behaviour. It is also stops increasing much sooner then the average relative error.

It makes sense for the errors to be increasing gradually each iteration. After all, during the downsampling, the values of O_i are chosen to be exactly that of $\hat{O}_i$. Each iteration then introduces a small error compared to what a newly downsampled value would give. This makes O diverge from $\hat{O}$ more and more until they stop changing. Using fewer iterations will result in less coverage by high resolution levels of the octree. At the same time it will also result in less divergence from the ideal light distribution. Using many iterations can however void the problem with the higher divergence to some extent since the final solution would instead contain more of the higher resolution levels of the octree, and specifically O_0. Along with the related performance considerations it is still a trade off between speed, accuracy and visual quality.

Light bleeding and back bleeding There are two important incorrect behaviours of indirect light when using the suggested propagation scheme. The first one is that light does bleed back onto the surface that caused it, even if just slightly. The other is that indirect light propagates through solid occluding geometry to some extent. Both of these behaviours are incorrect from a physical perspective and can cause the resulting illumination to appear unrealistic.

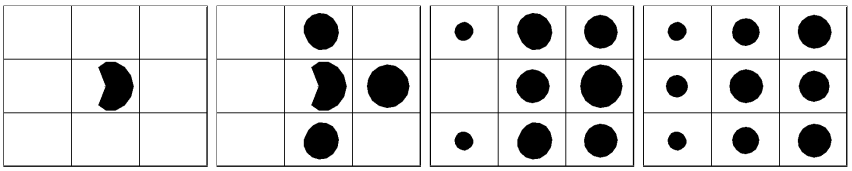


Figure 16: Illustration of the back bleeding behaviour during propagation. Initially there is only one element with any light, and it is heavily directed towards the right. After the second iteration there is clearly light on the wrong side of the original injection point. There is even more light after a third iteration.

When light is initially injected into the light propagation volume it is directed in the form of a clamped cosine lobe function along the surface normal. If this function is evaluated in a direction back towards the surface it will yield zero. So far it behaves in a realistic way. However, when the light is being propagated in all the other directions the contributions added to the adjacent elements have a different direction and often less directional information. Because of this it is possible for injected light to *loop around* and cause lighting back onto its own source surface. This happens in a process of two or more iterations. This back bleeding is illustrated in figure 16.

When it comes to the light bleeding through occluding geometry it is important to remember that the original light propagation volumes technique did not take indirect occlusion into consideration at all. This is not because it was correct without it. Rather, this light bleeding was not very strong and rarely noticeable. The cascaded light propagation volumes technique introduced the concept of using a geometry volume to enable fuzzy occlusion. A similar solution was added into the simulator and real-time implementation of this project as an optional component. The problem with this type of fuzzy occlusion is that it does not block the light completely but rather just reduces its intensity as it passes through the occluding geometry. If the occlusion is applied with too much power, then the discrete nature of the light propagation volume makes itself noticeable. If it is not applied with enough power then light will essentially pass through the geometry unhindered. As a result of these complications the indirect occlusion is considered mostly as an idea for future improvement rather then an integral part of the technique.

6.3 Performance and optimization

The performance of a technique intended for use in real-time applications is essential. The original light propagation volumes technique does run in real-time on modern hardware. It would theoretically need $k = 32$ iterations in order to propagate light throughout the entire volume, even though $k = 16$ iterations is often more then enough in practice[5]. Using the octree light propagation volumes this number was lowered to $k = 4$. The intention is that this decrease should be able to more than make up for the added time to manage the more complex data structure.

[5]This is determined using the baseline implementation and visual analysis of the result.

All of the performance data in this section is generated on a computer with a NVIDIA GeForce GTX 480 graphics card. This card is, at the time of writing, more than two years old. More recent hardware is likely able to boost the results in this section. Due to lack of time it would not have been feasible to fully optimize the entire implementation of the technique. Instead, the focus has been on optimizing the newly added steps which are closely tied to the octree representation. This also includes modifications to previously existing steps which were needed to work with the octree and the index volume. This includes the downsampling, propagation and merging steps, but generally not the creation of reflective shadow maps, light injection[6] or the final rendering.

Performance measurements The performance of the technique is measured in the real-time implementation. Different parts are timed individually from others. A total of eleven parts are timed. Each of those record both how many times they are invoked and the average and total computation time in milliseconds. Most of these parts are implemented using CUDA, which is highly parallel and performs many of the calls asynchronously by default. This can be problematic when measuring performance. Thus, application was run with the environment variable `CUDA_LAUNCH_BLOCKING=1` which disables asynchronous kernel launches. It should however be noted that this only prevents actual CUDA kernel launches from being asynchronous, and not necessarily anything else. Specifically the creation of reflective shadow maps is not even handled by CUDA at all but by the internals of OpenSceneGraph and the OpenGL rendering pipeline. Thus, it may or may not allow for accurate timings to be recorded. Additionally, some of the clear computation modules, responsible for resetting the data structures are implemented as simple calls to `cudaMemset` rather then as an actual CUDA kernel. As such, these calls may still be asynchronous.

It is also important to realize that other factors may play an important role in the timings measured when using CUDA. For example, memory transfers between host and device may be performed asynchronously, but needs to be completed before the next operation can be started. Because of that, operations that follow a memory transfer may have to wait for prior memory transfers before it can run. This can impact performance measurements even more.

The performance of the octree light propagation volumes technique is compared to that of the original, uniform light propagation volumes, technique. Implementations of both of these were created as part of this project, and it is the performance of those implementations that is compared. The implementations run for a total of 500 frames. During that period timings for the various parts are recorded. The resulting timing data is presented in table 1.

Performance comparison The results from table 1 indicate that the octree light propagation volumes technique does outperform the traditional light propagation volumes technique. More precisely, the total rendering time of each frame is on average reduced by 8%. The results do not guarantee that this will be true for all implementations of these techniques. For example, implementations utilizing more efficient light injection would get a greater improvement. The propagation is the stage of the technique which experiences the highest increase

[6] The exception being the updating of the index volume during injection. This is however a trivial operation.

		Octree	(ms)		Uniform	(ms)
Part	**Count**	**Average**	**Total**	**Diff**	**Average**	**Total**
Frame	500	106	53193	-4801	115	57994
RSMCreate	9000	0	3053	316	0	2737
GVClear	500	1	671	-50	1	721
GVInject	1500	10	15593	-110	10	15703
GVDownsample	500	0	55	55	-	-
IXClear	500	5	2923	2923	-	-
LPVClear	500	8	4461	270	8	4191
LPVInject	1500	13	19929	-2884	15	22813
LPVDownsample	500	0	142	142	-	-
LPVPropagate	500	7	3788	-5615	18	9403
LPVMerge	500	0	39	39	-	-
Total				-4914 8.28%		

Table 1: Data specifying the amount of times each part of the technique was executed and how long these executions took, both in total and on average. The data is presented both for the octree based implementation and the implementation using uniform light propagation volumes. The times are specified in milliseconds.

in speed, almost 60%. This follows from the decrease in iterations from $k = 16$, when using the uniform implementation, to $k = 4$, when using the octree implementation. This speed increase is however slightly countered by the addition of the downsampling step, the merge step and managing the index volume. These additional costs are however small in comparison to the gains.

The achieved frame rates with both implementations are fairly low at 9.4 frames per second using the octree implementation versus 8.6 frames per second using the traditional uniform implementation. Most of the time is lost during the light and geometry injection which uses up approximately 66% of the frame time for both implementations. This has almost nothing to do with the octree implementation.

The NVIDIA implementation of cascaded light propagation volumes from [6] reaches a frame rate of between 60 and 80 frames per second with similar settings, but with a single directed light source. This equals a single reflective shadow map rather then the 18 that are used in the implementations in this project.

7 Discussion

Error calculations The error calculations presented in section 6.2 have been used to guide much of the development of the technique. This way of calculating errors does however have one important weakness which has not been taken into consideration during this project. It takes fewer iterations for light in a lower resolution level to propagate than in the higher resolution levels. That means that there will be situations when the downsampled elements of $\hat{O}$ are calculated from elements in O which are still zero. This causes the reported errors to be greater then they really are. It is uncertain to what extent this problem affects the results. Either way, in order to get accurate error measurements the errors would have to be calculated in a different way. The same basic metric could be calculated, correctly, by first generating a traditional uniform light propagation volume in advance and then using that to calculate $\hat{O}$ for comparison.

Section 6.2 also presents another possible way of calculating the errors. With more time it could be beneficial to investigate both these alternatives and re-evaluate some of the choices made as a result of the current scheme. This would include for example the use of independent propagation on each octree level, the amount of iterations performed and the propagation factor.

Optimized injection As described in section 5.2, the real-time implementation utilizes atomic add operations to global GPU memory during the light injection. Section 6.3 additionally shows that the injection is by far the slowest part of the implementation. The real-time implementation in this project uses many more reflective shadow maps than the other implementations that have been investigated. Unfortunately the implementation would still be slower. Still, a more efficient implementation of the light injection would go a long way towards eliminating that difference.

The original papers on light propagation volumes [10, 11] mention the use of *point based rendering* to perform the light injection. This is also what the NVIDIA implementation [6] appears to be using. Such an approach would likely be able to perform much better than a CUDA computation kernel using atomic global memory writes. On the other hand, such an approach would require the light injection to be implemented as an OpenGL shader rather then a CUDA kernel. This would result in a more complex and less uniform implementation. With the potential of such huge performance increases, such an implementation would still be worthwhile.

Optimization of parallelism for GPGPU computing Graphics hardware is specifically optimized for particular tasks and particular uses. Even if they can be used for general purpose computations through GPGPU solutions such as CUDA, the code must still take this into consideration in order to achieve high performance. Some important things to take into consideration are:

1. Trying to ensure that all threads within each *warp*[7] follow the same path through the code, specifically by not branching differently.

2. Making efficient use of fast memory banks such as registers, local memory and shared memory wherever possible, instead of global memory.

[7] A warp is a group of threads that are executed together.

3. Ensuring that memory accesses are optimized for caching, accessing them linearly as textures wherever possible since the hardware is specifically optimized for common texture access patterns.

The technique itself as well as the real-time implementation naturally allows for a relatively uniform execution. Except for in border cases there is usually no reason for the processing of adjacent elements to cause diverging branching. While the implementation could take even more care to optimize for the first point, it already does it relatively well. When it comes to memory access however, the real-time implementation is not very good. Global memory is used for a lot of both reading and writing and is even used with slow atomic operations. Switching to point based rendering for light injection would help this situation a great deal, but it could also be further optimized in CUDA. For example by transferring data to more local, and faster, memory and working on it there rather than directly in global memory.

When it comes to actual memory access patterns, different parts of the implementation tend to use different memory access patterns. Some of them imply more random access patterns while some are already accessed linearly using accelerated texture access operations.

All in all, given more time, it would be possible to improve the CUDA code greatly to take these points into consideration where they are currently not being considered. Profiling using dedicated CUDA profiling tools is likely to reveal many possible improvements. The NVIDIA Compute Visual Profiler confirms that the implementation suffers from low throughput to global memory, that shared memory could be used to speed up data access instead of using global memory and that texture memory could also be an alternative for some of the data that is currently stored in global memory. It also states that some of the CUDA kernels are divergent, and that optimizing branching with regard to warps could increase performance additionally.

Viability in practical applications In practical applications and especially games it is essential that the performance of the technique is high enough to achieve real-time frame rates, preferably of more than 60 frames per second on modern hardware. At the same time there should not be any obvious visual artefacts and in particular no flickering. All of this should be maintained while a multitude of other unrelated effects and techniques are used at the same time.

The real-time implementation combines the octree light propagation volumes technique with shadow mapping and bump mapping, but not much more. The achieved frame rate does not even come close to the desired frame rates for games. With some improvements, such as point based rendering based light injection, and with fewer reflective shadow maps, this problem can mostly be overcome. In that case the technique might be applicable in practical applications for modern high end hardware as long as not too many other high demand techniques need to be used as well. With additional optimizations such as those already suggested in this section the technique could be made more viable for such applications. There are some visual inaccuracies and artefacts, as described in section 6.1. With proper tweaking by the artists these should however barely be noticeable. Finally, running the technique on more modern hardware is likely to provide a noticeable performance boost as well.

7.1 Future work

There are several ways that the octree light propagation volumes technique could be extended or modified in order to gain various advantages. Some such ways that were considered during the duration of this project are presented in this section.

Propagation for different resolutions Each level of the octree has its own resolution, but they all cover the same volume in world space. This means that for each level in the octree, the elements have a different size than for the other levels. The propagation scheme additionally causes the light intensity to fall off by distance. This falloff is however not individually configurable, but is rather baked into the global propagation factor and possibly also into the solid angle calculation and the reprojection into spherical harmonics. This causes the light falloff to be based on the distance in terms of the amount of grid elements rather than world space distance. Light in a higher resolution level of the octree will fall off significantly faster in terms of world space distance than light in a lower resolution level.

A falloff behaviour that acts differently for each octree level in world space is not realistic. A solution to this might also help to solve other problems such as the spatial and temporal intensity jumps described in section 6.1. It would be possible to overcome this problem by using a separate propagation factor p_i where $i = 0, \ldots, l-1$ is the level of the octree where the factor is applied. These propagation factors could be manually tweaked through a trial and error procedure, but that would be time consuming and error prone. A more promising solution would be to determine these factors analytically, allowing all of the level dependant propagation factors p_i to be derived from a global propagation factor p. Unfortunately this is not a trivial thing to do. Given more time it would be interesting to further investigate the possibility of such a solution.

Alternative octree subdivision strategies There are obviously many ways that the index volume can be populated. In this project the index volume will always point to the minimum value for i where the corresponding element of O_i is non-zero. This will cause the index volume to contain low values close to directly lit geometry and high values far from such geometry. This appears to work reasonably well without causing any noticeable artefacts. It may however be possible to improve it even more.

One possible improvement involves making the index volume contents and its falloff more symmetric. In the current technique the change to the lower resolution levels of the octree can depend greatly on where within the low resolution elements a higher resolution element is located. For example, if some light is injected into one of the elemenst closest to the center of the octree and then downsampled, then the worst case is experienced. This would involve a jump all the way from O_0 to O_{l-1} in two directions, while a nice gradual transition is created in the other directions. This situation is illustrated in figure 17.

This problem is of course greatly countered by the propagation step which, if necessary, will cause the injected light to spread out across the borders, forcing each quarter of the octree to create a gradual transition in its own direction. There can however be situations like this even after the propagation if there are too few iterations to bridge these borders in various worst case scenarios.

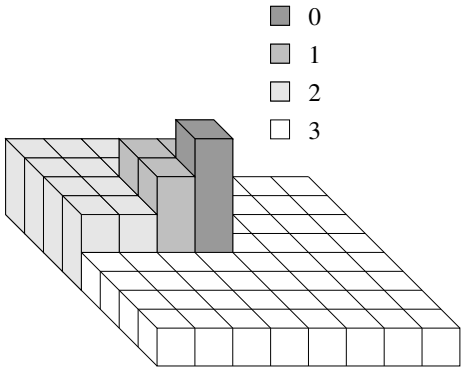


Figure 17: The jump from the highest resolution level $i = 0$ to the lowest resolution level $i = 3$ that can happen after light injection and downsampling.

A possible improvement could be to construct the index volume so that each injected value is radially expanded gradually from the injection point instead of always sampling the highest resolution level with non-zero data. A clever implementation of this would however be required in order not to have a strong adverse effect on performance.

Propagation between octree levels on the fly In this project the propagation has been performed individually on each level of the octree. This allows for a simple extension on top of the original light propagation volumes technique. On the other hand it can be slightly wasteful. Even if an element will never be sampled because a higher resolution level already provides full coverage over that entire area, it will still be calculated. The current structure of the index volume also does not take the full structure of the octree into consideration and will use only parts of some octree elements.

For this project there was one other method as a candidate for how to populate the octree. This was never developed to the point where any details, or even the feasibility of it, had been determined. In essence the propagation would only be performed once for every unique volume in world space without any overlap between levels. All other levels covering overlapping volumes in world space would remain unused. This would involve the light *jumping* between different levels of the octree during propagation. This jumping could be performed based on a criteria similar to what is currently used to create the index volume. The problem is that such a technique would be more prone to the effects of the propagation behaving differently on different levels of the octree. Specifically, this would require a solution to the problem with light falloff being resolution dependant.

One potential benefit of a solution like this would be that, no overlapping data needs to be maintained, the octree would no longer need to be a full octree. It can instead use more sparse representation and thus saving memory at the cost of an overall more complex technique.

8 Conclusion

This master thesis project set out to investigate if using an octree representation with the light propagation volumes technique would benefit the scalability or efficiency of the technique. It also evaluates the possibility of using omnidirectional point light sources.

The project concluded in a new technique and implementation for light propagation volumes using octrees. It uses a representation based on full octrees and adds a so called index volume which keeps track of which level of the octree to use for each part of the scene. On top of the original technique and the new octree representation, the technique also adds new steps of downsampling the injected light and merging the octree into a traditional uniform light propagation volume. This is all combined with the use of a proven method for injecting light from point light sources rather than just directional light sources.

The proposed technique produces an overall pleasing visual result, and performance measurements indicate that key parts of the technique can run significantly faster than their counterparts from the original technique. Unfortunately the use of six reflective shadow maps per omnidirectional point light results in a major performance hit. This makes it unrealistic to use the technique with many such light sources.

There are several visual artefacts and inaccuracies inherent to the technique which may need to be addressed or hidden prior to use in practical applications. This can likely be accomplished through artist tweaking, but future work in the field could also focus on addressing these issues in other ways.

References

[1] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering: Third Edition*, chapter 8.3 Ambient Light, pages 295–296. A K Peters, Ltd, 2008.

[2] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering: Third Edition*, chapter 8 Area and Environmental Lighting, pages 297–325. A K Peters, Ltd, 2008.

[3] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering: Third Edition*, chapter 9 Global Illumination, pages 327–437. A K Peters, Ltd, 2008.

[4] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering: Third Edition*, chapter 8.6.1 Spherical Harmonics Irradiance, pages 317–323. A K Peters, Ltd, 2008.

[5] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering: Third Edition*, chapter 9.1.4 Shadow Map, pages 348–372. A K Peters, Ltd, 2008.

[6] NVIDIA Corporation. Nvidia direct3d 11 sdk - diffuse global illumination demo. `http://developer.nvidia.com/nvidia-graphics-sdk-11-direct3d`, 0.

[7] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 203–231, New York, NY, USA, 2005. ACM.

[8] Cyril Crassin et al. Interactive indirect illumination using voxel cone tracing. http://research.nvidia.com/sites/default/files/publications/GIVoxels-pg2011-authors.pdf, 2011.

[9] Robin Green. Spherical harmonic lighting: The gritty details. http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.pdf, 2003.

[10] Anton Kaplanyan. Light propagation volumes in cryengine 3. http://www6.incrysis.com/Light_Propagation_Volumes.pdf, 2009.

[11] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. http://dx.doi.org/10.1145/1730804.1730821, 2010.

[12] Alexander Keller. Instant radiosity. http://dx.doi.org/10.1145/258734.258769, 1997.

[13] Aaron Knoll. A survey of octree volume rendering methods. http://www.cs.utah.edu/~knolla/octsurvey.pdf, 2006.

[14] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. Incremental instant radiosity for real-time indirect illumination. In *Proc. Eurographics Symposium on Rendering 2007*, pages 277–286. Eurographics Association, 2007. http://www.tml.tkk.fi/~timo/publications/laine2007egsr_paper.pdf.

[15] Morgan McGuire and David Luebke. Hardware-accelerated global illumination by image space photon mapping. http://dx.doi.org/10.1145/1572769.1572783, 2009.

[16] Frank Meinl. Cryengine 3, crytek - sponza model. http://www.crytek.com/cryengine/cryengine3/downloads/, 2010.

[17] Ravi Ramamoorthi and Pat Hanrahan. On the relationship between radiance and irradiance: Determining the illumination from images of a convex lambertian object. *Journal of the Optical Society of America (JOSA)*, 2001.

[18] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. http://dx.doi.org/10.1145/1507149.1507161, 2009.

[19] Jesper Børlum et al. Sslpv subsurface light propagation volumes. http://cg.alexandra.dk/wp-content/uploads/2011/06/SSLPV_preprint.pdf, 2011.

[20] Peter-Pike Sloan. Stupid spherical harmonics (sh) tricks. http://www.ppsloan.org/publications/StupidSH36.pdf, 2008.

[21] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 527–536, New York, NY, USA, 2002. ACM.

A Rendered images

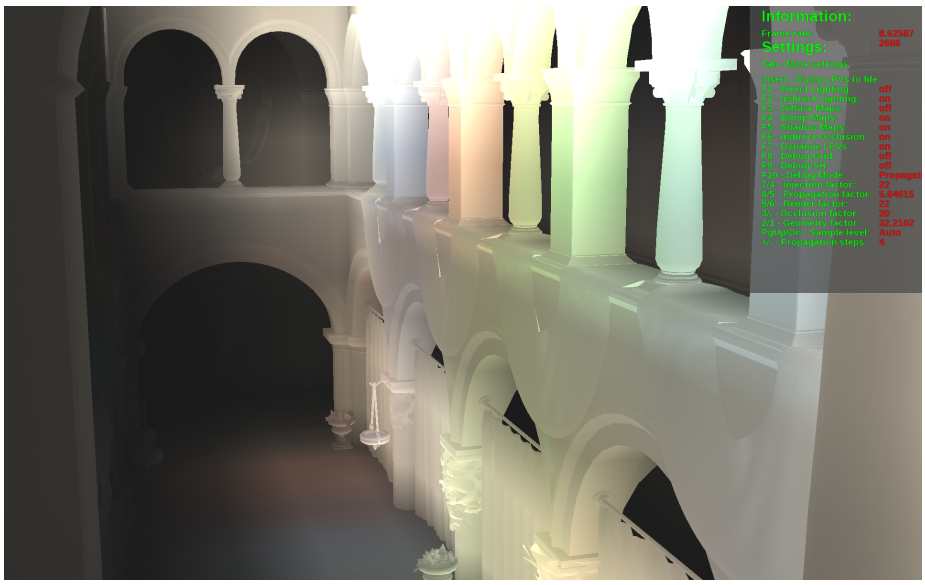


Figure 18: The sponza scene rendered using only indirect illumination and without any diffuse texturing. Note that the indirect lighting has been slightly exaggerated in this image.

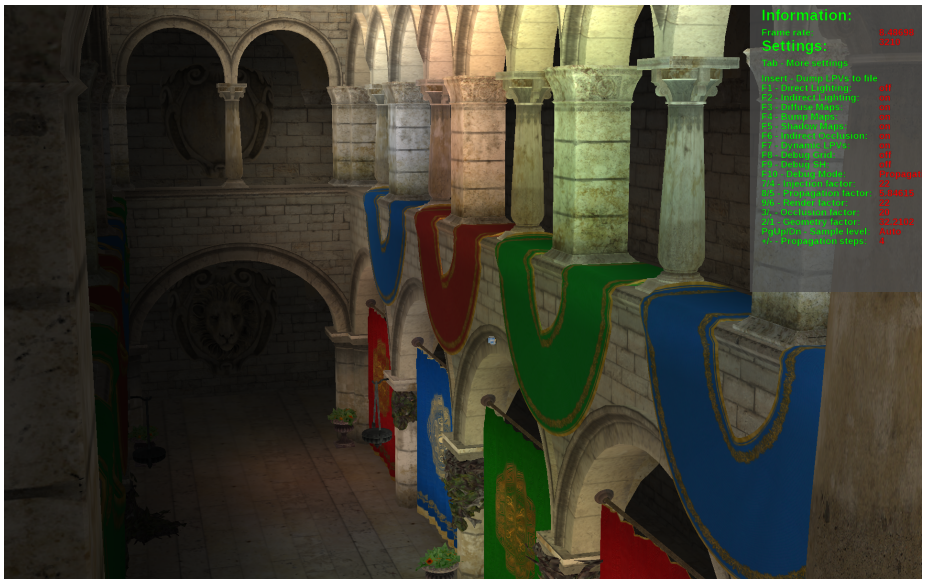


Figure 19: The sponza scene rendered using only indirect illumination but with diffuse texturing. Note that the indirect lighting has been slightly exaggerated in this image.

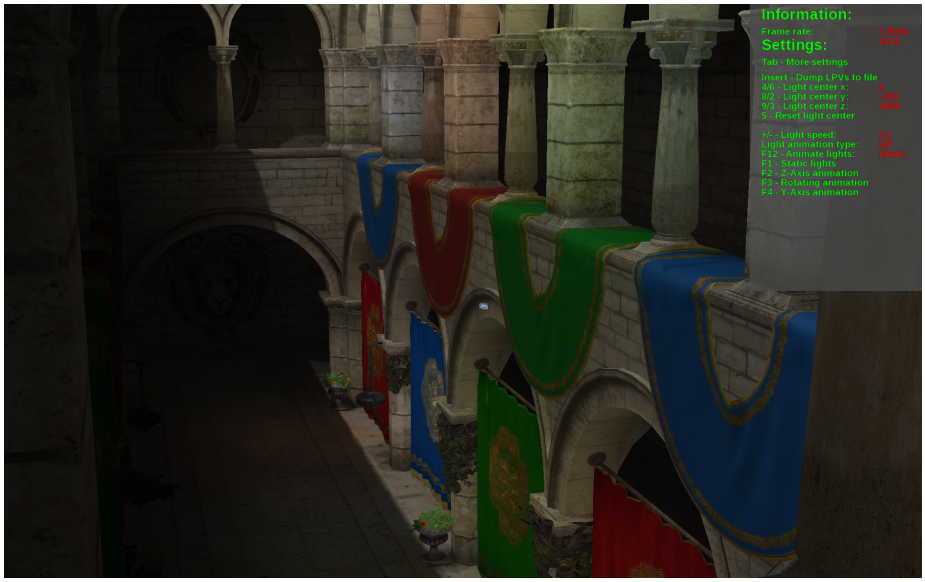


Figure 20: The final rendering of the sponza scene with both direct and indirect illumination and diffuse texturing. Note that the indirect lighting has been slightly exaggerated in this image.