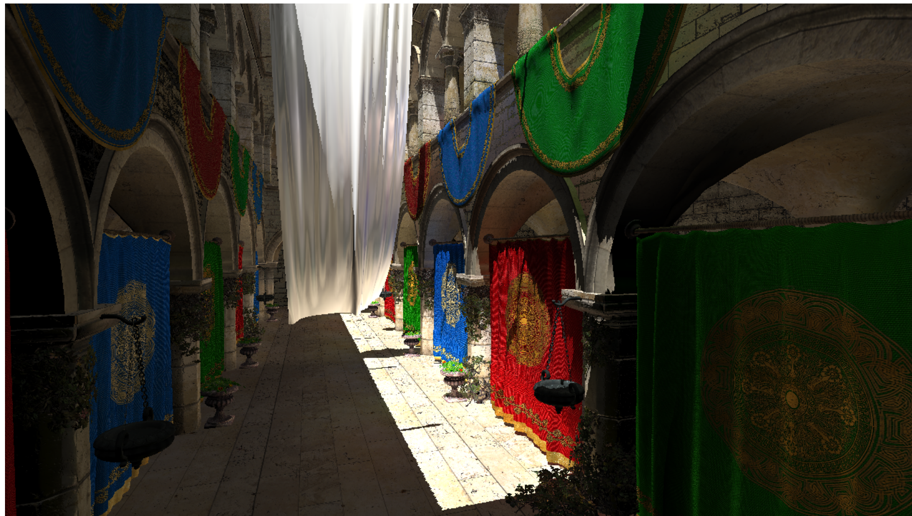
Real-time Global Illumination

A quantitative measurement of approximate global illumination algorithms



Yme ten Brug
ICA5602718

Supervisor:
Marc van Kreveld
Jacco Bikker

Utrecht University
Game & Media Technology
August 14th, 2017

Abstract

Real-time global illumination has been a goal for graphics researchers for many years. Over the years many different algorithms have been proposed to capture the complex interaction between light and an object. Comparisons between these algorithms have mostly been based on visual judgement. In this research we developed a scoring metric to quantitatively measure the visual quality of global illumination algorithms. We tested this metric on two different global illumination algorithms: Light Propagation Volumes and Voxel Cone Tracing. In the end, both algorithms scored equally well. Apart from some anomalies, the variations within the scoring are in line with the expected results.

Contents

1 Introduction **3**
- 1.1 Research Question . . . 3
- 1.2 Development . . . 3
- 1.3 Contribution . . . 4
- 1.4 Overview . . . 4

2 Related Work **5**

3 Global Illumination **6**
- 3.1 Physics of Light . . . 7
  - 3.1.1 Terminology . . . 9
  - 3.1.2 Irradiance Maps . . . 10
  - 3.1.3 Spherical Harmonics . . . 11
- 3.2 Path Tracing . . . 12
- 3.3 Light Propagation Volumes . . . 13
  - 3.3.1 Injection . . . 13
  - 3.3.2 Propagation . . . 16
  - 3.3.3 Rendering . . . 19
  - 3.3.4 Specular Reflections . . . 20
  - 3.3.5 Cascaded Light Propagation Volumes . . . 20
- 3.4 Voxel Cone Tracing . . . 21
  - 3.4.1 Voxelization . . . 22
  - 3.4.2 Cone Tracing . . . 24
  - 3.4.3 Indirect Diffuse Lighting . . . 27
  - 3.4.4 Indirect Specular Lighting . . . 28
  - 3.4.5 Spherical Harmonics . . . 29
  - 3.4.6 Cascaded Voxel Cone Tracing . . . 30
  - 3.4.7 Other Uses . . . 30
- 3.5 Ambient Occlusion . . . 32
  - 3.5.1 Screen Space Ambient Occlusion . . . 32
  - 3.5.2 Color Bleeding . . . 34
  - 3.5.3 Voxel Cone Traced Ambient Occlusion . . . 35

4 Error Metric **37**
- 4.1 Structural Similarity . . . 38
  - 4.1.1 Luminance . . . 38
  - 4.1.2 Contrast . . . 38
  - 4.1.3 Structure . . . 39
  - 4.1.4 SSIM . . . 39
  - 4.1.5 Image Quality Assesment . . . 39

4.2 Modifications 40

5 Experimental Setup **41**
5.1 Basic Scene 42
5.2 Sponza Scene 43

6 Results & Evaluation **46**
6.1 Results 46
6.2 Evaluation 48

7 Conclusion **49**
7.1 Conclusion 49
7.2 Future Work 49

Bibliography **50**

List of Figures **52**

List of Tables **53**

Appendices **54**

A Experimental Results **55**
A.1 Basic Scene Results 55
A.2 Sponza Scene Results 56
A.3 SSIM Index 56
A.3.1 Basic scene: Configuration 1 57
A.3.2 Basic scene: Configuration 2 63
A.3.3 Sponza scene: Configuration 1 69
A.3.4 Sponza scene: Configuration 2 80

B Shader Code **92**
B.1 Spherical harmonics 92
B.2 Light Propagation Volume 93
B.3 Voxelization 94
B.4 Image Atomic Average 96

1. Introduction

Global Illumination has been a widely researched topic in real-time computer graphics for many years. The ever increasing power of consumer grade hardware has made it possible for developers to explore more realistic techniques within interactive time constraints. Over the years, several algorithms and techniques have been proposed to capture the complex interaction between light and an object. Comparisons between these techniques have so far been limited to either visual judgement (i.e. one image showing one technique, another image showing a different technique) or at most a per-pixel error metric comparing one method agains a 'ground truth' solution. This research will investigate the use of an error metric to quantitatively measure the visual quality of an approximate global illumination algorithm.

1.1 Research Question

> *How can we quantitatively measure the visual quality of different approximate global illumination algorithms?*

Most current global illumination techniques conclude by either comparing their result to a ground truth solution and highlighting the differences, or to compute a per-pixel difference, like a *Root-Mean-Square* measurement. The goal of this research is to provide a comparison technique, henceforth called an error metric, that quantitatively measures the result of a global illumination method on a scene. Ideally, this error metric will take two parameters, a ground truth reference image and the image generated by the algorithm. The result of the error metric will be a score between zero and ten, indicating how close the generated solution is to the ground truth reference.

This error metric will then be used to measure two different approximate global illumination algorithms, Light Propagation Volumes and Voxel Cone Tracing.

1.2 Development

In order to conduct this experiment, a custom graphical engine is developed. This engine is written in C++ and uses OpenGL. At the start of the research the engine supported a physically based lighting model as well as a deferred rendering scheme. This engine will be extended with two global illumination algorithms, Light Propagation Volumes and Voxel Cone Tacing.

1.3 Contribution

Our contribution is an error metric capable of quantitatively measuring the visual quality of a global illumination algorithm compared to the ground truth reference.

1.4 Overview

This introduction has served to introduce the subject of the thesis. The rest of this thesis is structured as followed. Section 2 gives a short overview of previous work in the area of real-time global illumination techniques. Section 3 explains the math and physics behind global illumination and gives a detailed overview of the two methods implemented. Next in section 4, the error metric used during this research will be explained in detail. The experimental setup that is used is described in section 5. The results of this experiment and the evaulation of those results can be found in Section 6. We conclude this thesis in section 7, with our conclusion and a discussion of possible future work.

2. Related Work

The quest for real-time global illumination started with the introduction of *Virtual Point Lights* (VPL) by Keller in [Kel97]. VPLs are based on the idea that after a point has been lit, this point can act as a light source for other points. [Kel97] employs a quasi-random walk to create a set of particles that are representative of the 'path' the light follows. Each point of this walk is converted into a VPL with the appropiate color intensity. The scene is rendered for every VPL and the results are accumulated to get the final result.

The idea of virtual point lights was adapted by Dachsbacher and Stamminger for use in modern hardware, introducing the *Reflective Shadow Map* (RSM) [DS05]. Reflective Shadow Maps are an extension to the regular shadow mapping algorithm that simulates a single bounce of indirect light. A RSM stores the depth of the pixel, just as with a regular shadow map. Additionally, the RSM stores the world position, normal and the flux of a pixel. Using these four parameters, every pixel in the RSM can act as a VPL. An importance sampling strategy is used to gather indirect lighting from the RSM.

Crytek extended upon the idea of the Reflective Shadow Map and introduced *Light Propagation Volumes* (LPV) [KD09]. A Light Propagation Volume is a grid that spans the entire scene. Each cell in the grid contains the amount of indirect illumination at that point in the world. For every light, a RSM is generated and from this RSM a number of VPLs are injected into the grid. The light is then propagated throughout the grid. Light Propagation Volumes are explained in more detail in section 3.3.

More recently, Crassin, Neyret, Sainz, Green and Eisemann developed a technique to approximate global illumination by voxelizing the scene and tracing cones across this voxelization in [CNS+11]. Tracing a cone across the voxelized scene allows for a quick estimate of the amount of lighting present in a certain direction. By tracing multiple cones an estimate of the amount of indirect illumination can be made. More details on Voxel Cone Tracing and its implementation can be found in section 3.4.

Performance & Artifacts Of the four methods mentioned above, Light Propagation Volumes and Voxel Cone Tracing are of interest to this research. VPLs as introduced in [Kel97] do not perform in real-time. [DS05] mentions several methods to increase the performance of the RSM method in order to get interactive framerates.

Of the four methods Light Propagation Volumes and Voxel Cone Tracing are the ones designed with real-time performance in mind. With both being a grid based method, their performance is largely determined by the resolution of this grid. Likewise, the grid-based approach can lead to light leaking at small or thin objects.

3. Global Illumination

In computer graphics, one can make the distinction between two different kinds of lighting, direct lighting and indirect lighting. Direct lighting is the result of light directly traveling from a light source to the surface of an object. The problem is that light does not stop once it hits an object. Instead, it is scattered in a random direction from the surface of the object. This means that an object can receive light from a light source, through another object. This process is illustrated in figure 3.1.

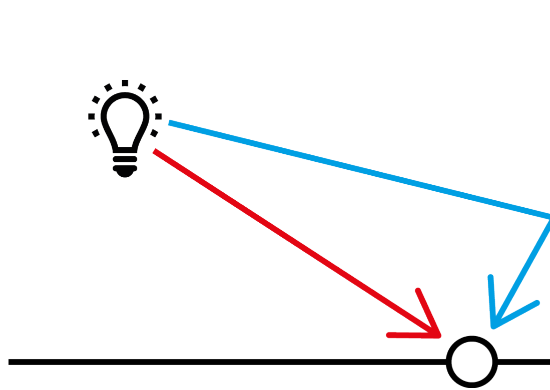


Figure 3.1: Direct and indirect lighting. The point receives light directly from the light source (the red arrow), but also through the scattering of the wall (the blue arrow).

This scattering is called indirect lighting and the sum of both the direct and indirect lighting is called 'Global Illumination'. One of the most prominent effects of this scattering is the so-called 'color bleeding' effect. The color of an object is reflected, or 'bleeds', onto another object. An example of color bleeding can be seen in figure 3.2.

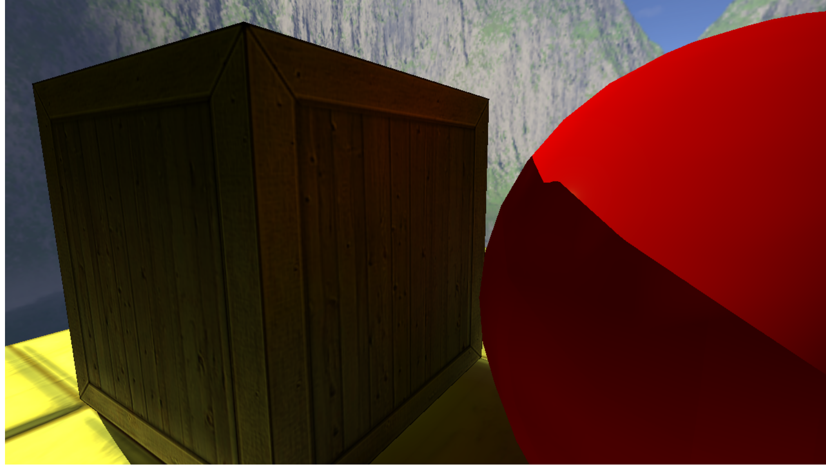


Figure 3.2: Color bleeding. Notice the red color reflected from the sphere on the top crate and the yellow color reflected from the floor at the bottom of the crate. The image was generated using the engine written for this research.

3.1 Physics of Light

In order to understand why the color bleeding effect happens, a more thorough understanding of the math and physics behind light is required. This section will only scratch the surface of light interaction and is by no means a comprehensive listing. The interested reader is encouraged to read [Hof12].

When light hits the surface of an object, part of the light will be reflected of the surface and the rest will be refracted into the object. What happens with the light inside the object depends on the material of the object. Some of the light will be absorbed, while other parts will be scattered. Scattering is caused by the light hitting particles inside the object. Because this usually happens close to the surface, the light will most likely leave the material again. When this light reaches our eyes or a camera, the object will be visible according to the light that was emitted from the object. How much light is absorbed or scattered determines the color of a material. Metals have a very high absorption rate, meaning that all light that travels into a metal object is absorbed and no scattering takes place. This means that officially, metals do not have a 'color'. The appearance of a metal object is determined by the amount of light reflected from the surface. Figure 3.3 shows the reflection and absorption for both a metal and a non-metal (dieletric) material.

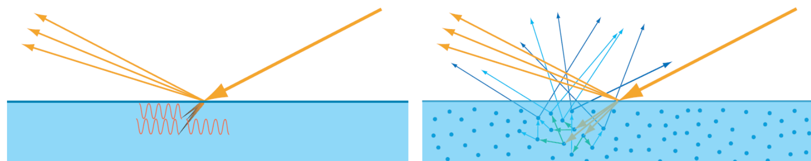


Figure 3.3: Light absorption and scattering. Some percentage of the light will be scattered based on the incoming angle of the light (the orange arrows). The remaining light will travel into the material. The material on the left is a metal, so all the light that travels into the material is absorbed. The light in the material on the right is scattered inside the object and leaves the surface again at a random angle (the blue arrows). The figure was taken from [Hof12].

The reflected light is called the specular lighting and the scattered light is called the diffuse lighting. For diffuse lighting, it is assumed that the light will leave the object at the same location uniformly distributed on the hemisphere around the normal of the surface This assumption holds for most materials, with the exception of materials with a high transluancy. Prime examples of such materials are milk, wax and human skin. To accurately render these kinds of materials, the *sub-surface scattering* needs to be taken into account. Modeling this behaviour requires complex algorithms which we will not discuss in this thesis.

The scattering of diffuse light is what causes the color bleeding effect previously discussed. The contribution of the specular lighting to global illumination is *reflections*.

Over the years, several models have been proposed to compute the diffuse and specular lighting on a surface point, with the most common today being a physically based *Bidirectional Reflectance Distribution Function* (BRDF). The BRDF is a function that depends on two factors, the incoming and the outgoing direction, and it computes how much of the light from the incoming direction is scattered or reflected in the outgoing direction. One of the most popular BRDFs today is the *Microfacet BRDF*. A microfacet is assumed to be an infinitely small plane with perfect reflection around its normal. The microfacet theory assumes that on a microscopic level, every material consists of a large number of microfacets, each oriented along a different normal. The microfacet BRDF then computes the probability that a microfacets normal is aligned so that light will be reflected from the incoming direction into the outgoing direction. Figure 3.4 illustrates the concept of microfacets.

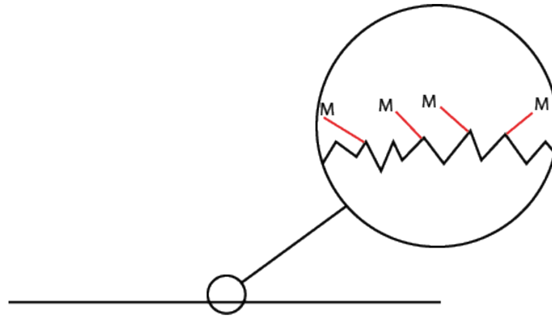
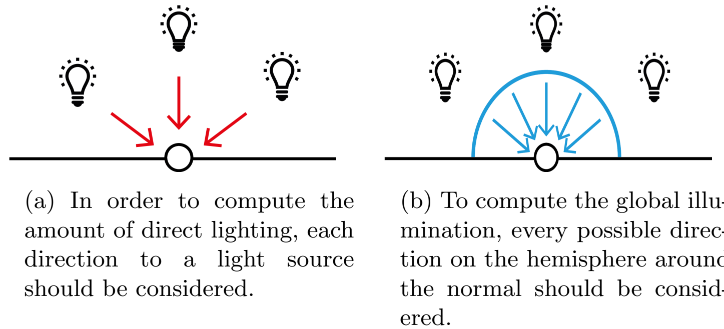


Figure 3.4: Microfacets. A microfacet is an infinitely small plane with perfect reflection along its normal (red arrow). The amount of lighting reflected in a certain direction is based on the probability that a microfacet normal is aligned in a specific direction.

When computing the direct lighting, the BRDF is evaluated for every outgoing direction that points to a light source. To include the indirect lighting, every possible direction on the hemisphere oriented around the normal of the surface should be taken into account. Figure 3.5 illustrates this concept.



(a) In order to compute the amount of direct lighting, each direction to a light source should be considered.

(b) To compute the global illumination, every possible direction on the hemisphere around the normal should be considered.

Figure 3.5: Direct and indirect lighting using a BRDF.

3.1.1 Terminology

In the field of global illumination, several mathematical quantities of light are commonly used.

Flux. The flux, or radiant flux, is the amount of energy that is emitted, reflected or transmitted from a light source. The flux decreases based on the distance, d the light has traveled, with a ratio of $\frac{1}{d^2}$.

Solid angle. The amount of energy a surface receives is dependent on its solid angle with regards to the light source. At an angle, the same amount of light is spread over a larger area. This means that the amount

of energy received from a light source decreases based on the angle between the surface normal and the vector from the surface to the light source.

Radiance. The radiance is the amount of flux emitted from a surface, per unit solid angle per projected unit area. It indicates how much energy is observed when looking at that surface from a specified viewing angle.

Irradiance. The irradiance is the total amount of energy arriving at a point, from all directions.

3.1.2 Irradiance Maps

One of the easiest and fastest ways to approximate some form of distant global illumination is in the form of *irradiance environment maps.* This technique, also called *Image Based Lighting* (IBL) is present in most modern game engines. The diffuse indirect lighting is computed by taking a large number of samples around the normal of each point in the scene. Assuming the environment is static, the results for every normal could be precomputed. This is exactly what an irradiance map represents. An irradiance map is usually presented using a cube map. Every pixel in the map represents the irradiance received from that direction. Since the assumption that the world is static does not hold up in most interactive applications, an irradiance map is usually only computed for the environment, the so-called distant lighting. To get some form of 'local lighting', most game engines support *Image Probes.* An image probe is placed in the world and captures the surroundings as if it was an environment map. By placing several probes around the world and interpolating the result based on the user's position in the world, an estimate for the global illumination is calculated.

Image based lighting has been researched extensively due to their efficient representation of (static) global illumination. Several large game engines, such as the Unreal Engine, support IBL. For a more detailed explanation of the IBL technique, the reader is encouraged to read [Lag12] and [Kar13]. The result of using irradiance maps can be seen in figure 3.6. Figure 3.6 shows the effect of using irradiance maps to determine the amount of reflection (specular lighting) on an object. A diffuse irradiance map uses the same principle, but it is strongly blurred, reminiscent to the left-most sphere, due to the fact that diffuse light is spread around the entire hemisphere.

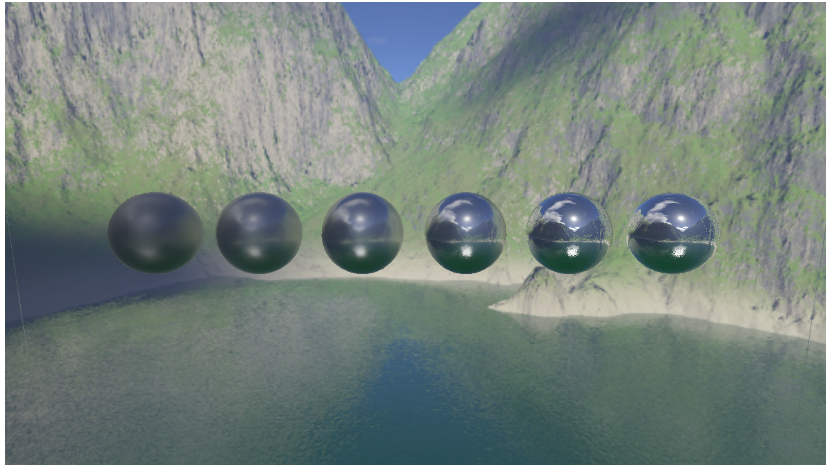


Figure 3.6: Irradiance maps. The spheres show the reflection of the environment on different levels of material roughness. A low roughness will result in a near perfect reflection (right most sphere), while a high roughness will obtain a much more blurred result (left most sphere). The image was generated using the engine written for this research.

3.1.3 Spherical Harmonics

Irradiance maps are usually stored in a cube map. Alternatively, the irradiance can be represented using *Spherical Harmonics* (SH) to represent the diffuse intensity. A spherical harmonic is the representation of an image over a sphere, just like a cube map is the representation of an image over a cube. A spherical harmonic consists of a set of N basis functions and a set of N weights. The basis functions are the same for every spherical harmonic and in the case of irradiance maps they can be seen as a direction in the same way that every pixel of a cube map can be seen as a direction. The weights are the actual data of the spherical harmonic, which is the actual RGB color in that direction. The number of basis functions theoretically go on forever and determine the number of directions the spherical harmonic can store. One can think of spherical harmonics as for example four colors in four different directions. When evaluating the spherical harmonic with a specific direction, the colors are interpolated to represent that direction. More basis functions equals more directions that can be stored, leading to a better interpolation. In the case of diffuse lighting we usually restrict ourselves to the first nine basis functions. To store this kind of spherical harmonic, four values are required for each color component. This means that a diffuse irradiance map can be represented using just twelve float values, instead of using a cube map. Due to the low frequency of this type of spherical harmonic, it is only suited for diffuse lighting. The nature of spherical harmonics is too complex to explain in this thesis, but the interested reader can read [Gre03] and [RH01].

3.2 Path Tracing

Path traced solutions solve the global illumination problem by tracing a potential path the light can follow. Since we are only interested in paths that end up at the camera (i.e. are 'visible'), the path is traced in reverse order. From the camera a ray is traced through each pixel of the screen. When this ray hits an object, a random direction is chosen to continue the path. This direction is used as outgoing direction for the BRDF and the lighting received from that direction is computed. This path continues until a light source has been reached or a set number of iterations have occurred. This is repeated a large number of times and the result is averaged to get a final solution. This is called *Monte-Carlo path tracing* and it is mathematically proven that when consequently choosing a random direction, the solution will eventually converge to the correct solution.

The main problem with this approach is the huge number of samples required to get a decent estimate. Depending on the scene complexity rendering an image using path tracing can take anywhere from several minutes upwards to several hours. A multitude of techniques exists to speed up the convergence of a path traced solution without sacrificing image quality. Figure 3.7 shows an example of different global illumination effects that can be achieved using path tracing. Figure 3.7 was generated by the path tracer.

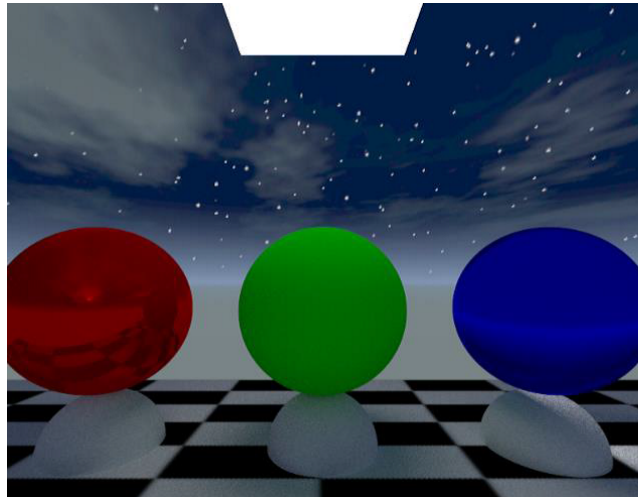


Figure 3.7: Path tracing. Global illumination is visible in the reflective red sphere, the refractive blue sphere (note how the light travels through the sphere and illuminates the ground behind it) and the green color bleeding under the green sphere. The image was generated using the path tracer used to create the reference images.

3.3 Light Propagation Volumes

Light Propagation Volumes (LPV) were first introduced by Crytek in their CryEngine3. They originally presented a course at SIGGRAPH 2009 [Kap09] and the algorithm was later presented in [KD09] by Kaplanyan and Dachsbacher.

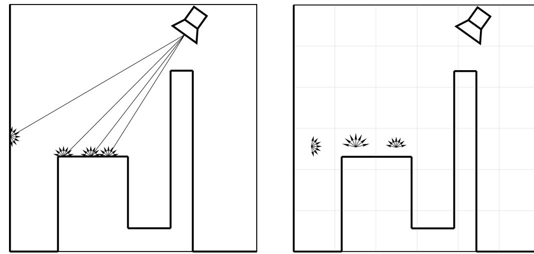
A light propagation volume is a grid that spans the entire scene. Each cell in the grid contains the amount of indirect lighting at that point in the world. The intensity of a cell is represented using spherical harmonics to preserve the directional information of the illumination. The paper uses spherical harmonics up to the second band, which amounts to four coefficients. The LPV is completely recomputed each frame.

The LPV algorithm consists of three steps, *injection* (section 3.3.1), *propagation* (section 3.3.2) and *rendering* (section 3.3.3). The injection phase fills the grid with an initial configuration of indirect lighting. The propagation phase then 'smears' this lighting across the grid. The final rendering phase reads the indirect lighting from the grid.

3.3.1 Injection

The first phase, injection, is used to generate an initial configuration of indirect lighting. It does this by generating and inserting a large number of *Virtual Point Lights* (VPL) into the grid. Indirect lighting is caused when a point, A, receives light from a light source through another point, B. The point B can be considered a *virtual* light source for point A. The VPLs are inserted by generating a *Reflective Shadow Map* (RSM) for each light source. A RSM is an extension to a regular shadow map. A RSM renders the scene from the perspective of the light, the same way a shadow map does. Alongside the depth, a RSM stores the world position, normal and the reflected flux of every pixel. The reflected flux is computed as the flux emitted through a pixel, multiplied by the material color. According to the original RSM paper [DS05], the flux should not be modulated by a distance attenuation term. In the case of a spot light, the flux decreases with the cosine to the spot direction, due to the decreasing solid angle. For any other type of uniform light, the flux is a constant value.

For every pixel of the RSM (or a subset of the pixels) a VPL is injected into the grid. When injecting a VPL into the grid, its location is used to determine the cell the VPL resides in. To prevent self lighting and shadowing, each VPL is virtually moved by half the cell spacing in the direction of the VPL normal. This ensures that a VPL that points away from a cell center is injected into the adjacent cells instead. Figure 3.8 illustrates the concept of creating a VPL from a light source and injecting it into the grid.



(a) An RSM is created for the light source. For several pixels of the RSM, a VPL is created.

(b) Each VPL created from the RSM is injected into the grid.

Figure 3.8: Light Propagation Volumes injection step. The figures were taken from the powerpoint accompanying [KD09].

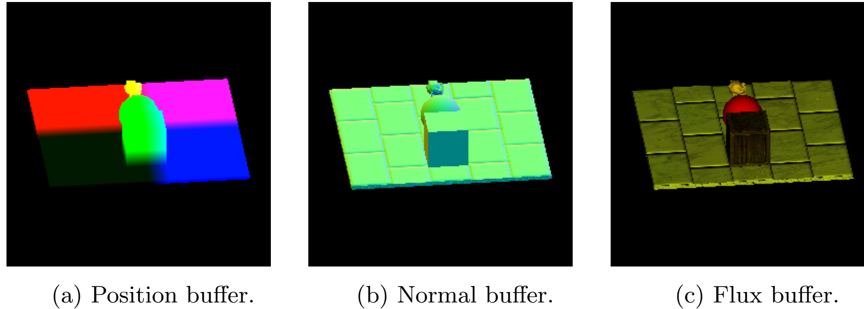
Low-frequency lighting, such as the illumination from environment maps or particle systems, can also be injected into the grid. This is achieved by creating a large number of VPLs from these light sources and injecting those VPLs into the grid. VPLs from an environment map for example are injected into the outer layer of the grid.

Geometry Volume

In addition to the LPV with light intensity, a separate grid holding a fuzzy representation of the scene is computed. This *Geometry Volume* (GV) holds an occlusion probability when traveling from one cell to the other. As with the LPV, the GV is recomputed each frame. The GV is constructed from the depth buffers used by the RSMs created for the injection as well as the depth buffer of the camera view. The GV has the same resolution as the LPV, but it is shifted by half a cell. This ensures that the center of the GV is located at the corners of the LPV cels. [KD09] claims this results in better interpolation of the occlusion propability.

Implementation Details

To store a spherical harmonic, four values are required for every color component. This means that we use one texture for the red, green and blue component, resulting in a total of three textures. The textures need to have a floating-point format, as the spherical harmonic can be negative or larger than one. For each participating light source, a reflective shadow map is generated. An example of a RSM can be seen in figure 3.9.



(a) Position buffer. (b) Normal buffer. (c) Flux buffer.

Figure 3.9: Reflective Shadow Map for a scene. The figures were created using the engine written for this research.

To inject the VPLs of the RSM into the grid, points are drawn onto the RSM. Each pixel, or a subset of the pixels, is drawn as a point. The vertex shader determines the position, normal and flux of this point, by sampling from the textures generated during the RSM pass. The position is used to calculate the cell index of the point in the grid. The geometry shader is then used to select the correct layer of the grid to render the final result to. The fragment shader transforms the flux to a spherical harmonic and stores the values in the grid. Appendix B.1 shows example shader code for computing the spherical harmonic components from a direction and the flux. Using this approach, we can write to all the appropiate grid cells in a single draw call. By enabling additive blending, the same cells can receive multiple VPLs. The geometry volume is filled using a similar approach, but the input for the geometry injection shader is the position and normal buffer from the RSM. In a deferred renderer, the position and normal buffer from the G buffer can also be used to fill the geometry volume.

One other aspect to take into account is the weight of each injected VPL. This weight is dependent on the area the RSM covers and area for the LPV. In order to simplify further computations, [KD09] makes several assumptions. Firstly, we assume that the area of a texel in the RSM is much smaller than the area of a cell in the LPV. Thus, we do not have to worry about discontinuity during the injection. Secondly, the projected area of the RSM is assumed to be equal to the projected area of the slice of the LPV perpendicular to the light direction. This is achieved by projecting each corner point of the box that contains the LPV onto the plane with the light direction as normal. From maximum width and height between these points is determined and these values are used for the orthographic projection of the RSM. Since the area for the RSM and the LPV are now equal, the weight of each VPL is only detemined by the amount of texels in the RSM and cells in the LPV. The weight is computed using equation 3.1. Note that in this equation, we use twice the LPV resolution, instead of three times, because we estimate the number of cells within the projected area

(the slice perpendicular to the light direction).

$$weight = \frac{(lpvResolution * lpvResolution)}{(rsmWidth * rsmHeight)} \tag{3.1}$$

The result of the injection step can be seen in figure 3.10.

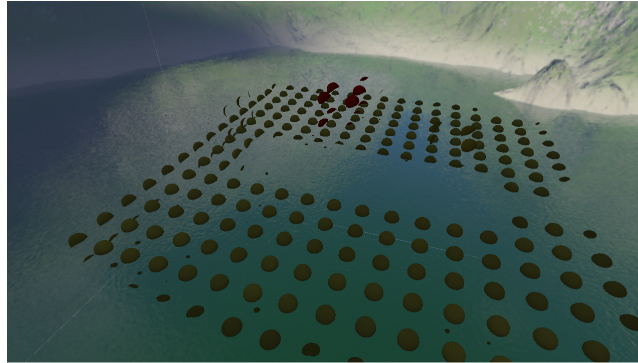


Figure 3.10: Intensity in the grid after the injection phase. The intensity in the grid is represented using a sphere for every texel of the grid. The normal of the sphere is used to evaluate the SH for visualization. The figure was created using the engine written for this research.

3.3.2 Propagation

After the injection phase, the grid contains the lighting information for just a few cells. In the propagation phase, the injected light is propagated throughout the grid. This means that the light is 'smeared' into the neighbouring cells, creating a color bleeding effect. The propagation is carried out in several iterations, each iteration spreads the light further away. Figure 3.11 shows the result after several iterations.

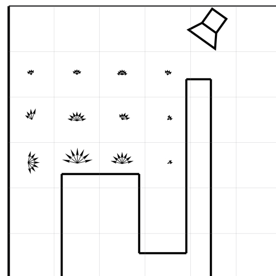


Figure 3.11: After several iterations, the light has been spread throughout the grid. The figure was taken from the powerpoint accompanying [KD09].

The initial input for the propagation is the grid from the injection phase.

The input to each subsequent iteration pass is the result of the previous propagation pass. During propagation, light is propagated to each of the cell's direct neighbours. In the three dimensional case, each cell has six direct neighbours. When propagating from one cell to the other, each face of the destination cell is considered. For each face the total amount of flux reaching the face from the source cell is computed. This flux needs to be converted to intensity again so it can be used during the next iteration. To compute the amount of flux the algorithm considers the solid angle of the face and determines the central direction ω of that solid angle. This means that they take the intensity in direction ω as average intensity over the solid angle.

Blocking of the light due to scene geometry is integrated in the propagation step. The GV stores an anisotropic occlusion probability that is modulated with the light intensity during propagation. This blocking is not considered in the first propagation iteration, to prevent self shadowing.

The quality of the propagation is dependent on the number of iterations performed. The paper proposes to use two times the longest side of the LPV as a heuristic for the number of iterations.

Implementation Details

The engine uses a ping-pong structure to switch between grids for the propagation. A key note in the propagation step is that the input for the propagation is the result of the previous propagation step. This is not the total accumulated intensity. So to properly do the propagation phase, one needs at least three grid structures. One to read the intensity from (the result of the previous propagation step), one to write the result of this propagation step to (the input for the next propagation step) and one that holds the accumulated intensity of all propagation steps. This ping-pong structure is illustrated in listing 3.1.

Listing 3.1: Propagation ping-pong

```
LPV propagate(int steps, LPV orig){
    LPV accum, A, B;
    A = orig;

    for (i = 0; i < steps; i++){
        // propagate light from A and store the result in B
        B = propagateFrom(A);
        // update accumulated intensity
        accum += B;
        // swap A and B for next propagation step
        swap(A, B);
    }
    return accum;
}
```

The injection step rendered each pixel of the RSM to inject light. For the propagation phase, we need to render each cell of the LPV. This ensures the

propagation touches upon every cell of the LPV. Propagation so far has been described as being spread throughout the grid. However, it is not possible to write to multiple cells of the grid at the same time. Therefore the actual implementation *gathers* the intensity from its surrounding cells. The principle remains the same, but instead of pushing its intensity towards the neighbouring cells, we compute how much intensity a cell receives from each neighbour. In order to get a correct propagation, we need to consider the amount of intensity projected onto the five faces of each neighbouring cell. The sixth face is the one adjoining both cells, so we do not consider that one. The reason we need to consider all five faces instead of just the 'main' direction, is because of the spherical harmonics. Spherical harmonics are a spherical representation of light that we are storing in a cubic grid. If we would ignore the side faces of a cell, we would have noticable gaps in the final result. Figure 3.12 shows a 2D illustration of this gathering process.

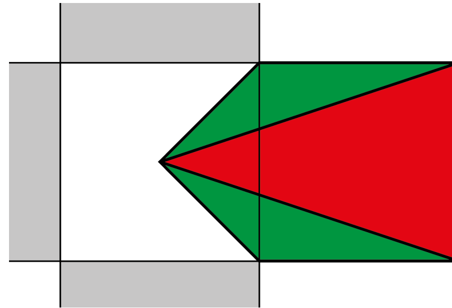
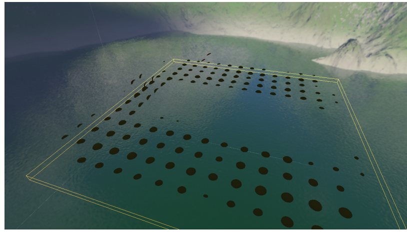


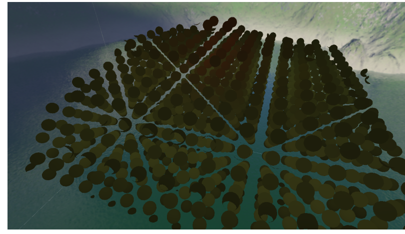
Figure 3.12: 2D example of the gathering operation done during the propagation phase. Besides the main direction(red), we also need to consider the side faces (blue and green).

In order to do the actual propagation, we loop through the six neighbours of a cell. For each neighbour, we compute the direction towards that neighbour and read the SH intensity from its cell. The amount of intensity reaching the main cell from this neighbour is computed by transforming the direction to the neighbour cell to a spherical harmonics. This intensity is modulated by the solid angle to the face and the blocking potential. For the main direction, the solid angle is the angle of the red cone in figure 3.12, for the side faces, the solid angle is the angle of the green cones. This process is then repeated for each side face of the neighbouring cell. Appendix B.3 shows pseudo code for the propagation phase.

Figure 3.13 shows the result of doing one propagation step versus thirty propagation steps.



(a) Result of doing one propagation step.



(b) Result of doing thirty propagation steps.

Figure 3.13: Visualization of the propagation steps for the LPV. The images were generated using the engine written for this research.

3.3.3 Rendering

The final part of the algorithm is to apply the intensity computed in the LPV to the final rendering result. Each cell in the LPV represents the light intensity at that point in the world. When shading a point, the cell this point resides in is determined. The inverse of the normal of the point is transformed to the SH (since we want to know how much light is going towards the surface) and combined with the result stored in the grid cell to get the amount of indirect lighting. The result of light propagation volumes can be seen in figure 3.14.

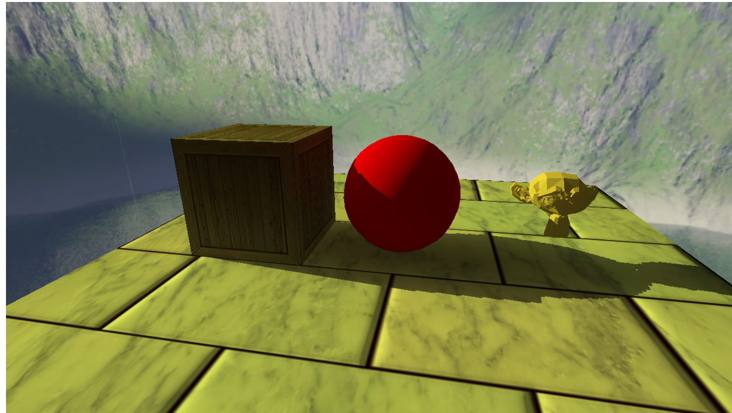


Figure 3.14: The result of using an LPV on a scene. For demonstration purposes the indirect lighting has been exaggerated.

Implementation Details

Implementing the rendering part of the LPV algorithm is relatively straighforward. Appendix B.2 shows example shader code to transform the normal to the SH and to retrieve the intensity value from the grid. While conducting our

experiments, we ran into an issue when using the LPV with a high grid resolution (128 in our case). During each propagtion step we 'lose' some intensity. This means that the total amount of light intensity for each propagation step is a bit less than the total amount of light intensity for the previous step. This is required, otherwise the light will propagate indefinitely. A downside of this approach is that there is a fixed number of steps before there is no more light to propagate. This means that the distance the light travels with the higher resolution is much lower than the distance traveled with a lower resolution. In the case of a grid resolution of 128, the light does not propagate far enough to correctly illuminate the surrounding geometry. All the figures presented in [Kap09] and [KD09] that detail the resolution of the LPV, show the resolution to be 32. Future research will be required to determine if the displayed behaviour is an error in our implementation or if high resolution grids are not possible with Light Propagation Volumes.

3.3.4 Specular Reflections

Light Propagation Volumes are designed for indirect diffuse lighting, as evidenced by their use of spherical harmonics. According to the paper, LPVs can be used to achieve specular reflections. The paper proposes to raymarch across several grid cells and averaging the intensity of all cells for the ray march direction, divided by the squared distance to the cell that they pass through. This can be seen as undoing the propagation steps. Propagation spreads the light throughout the grid, the ray marching gathers the light back to a single cell again. Since this research will not consider the specular part of global illumination, this part of the algorithm was not implemented.

3.3.5 Cascaded Light Propagation Volumes

Performance of the LPV is largely dominated by the number of propagation steps and the resolution of the grid. With a small scene, a small grid will suffice. If we for example have a scene with dimensions of 10 meter x 10 meter x 10 meter and a grid resolution of 32 x 32 x 32, then each cell of the grid will occupy an area $\frac{10}{32} = 0.3125$ meter in the scene. Since diffuse indirect lighting is 'blurry' by its nature, these dimensions will suffice for most cases. But what if we have a much larger scene? If the scene has a dimenson of 100 meter x 100 meter x 100 meter, each cell will occupy an area of more than three meters. We lose to much detail in this case. A possible solution is to increase the resolution of the grid, but this will also reduce performance and cannot be done indefinitely. The paper proposes to solve this issue by utilising a cascaded structure of nested LPV grids. Each grid has the same resolution, but the area it covers doubles each level. The grids are centered around the camera. Figure 3.15 shows the cascaded structure.

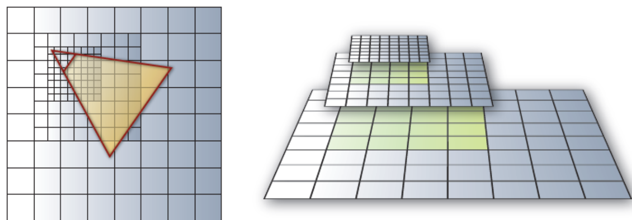


Figure 3.15: Cascaded Light Propagation Volumes. The area a grid covers quadruples each level. The figure was taken from [KD09].

The cascaded approach makes it possible to cover a large area at interactive frame rates. Centering the grids around the camera ensures that we get high precision close to the camera, while far away objects have much less detail. Each level of the cascade is treated separately and the injection and propagation phase are done for each grid in the cascade. Since the projected area of the RSM used for the injection is set to be the same as the project area of the LPV, each level in the cascade has its own corresponding RSM.

The contents of a LPV cell can be considered the average of all the lighting in that location. A small change in the grid position can noticably change the average of each cell, leading to flickering when moving the grid around the scene. This is a common problem with cascaded solutions and it is solved by snapping the position of each grid so that it is always in the corner of a grid cell. This ensures that the same cluster of VPLs is injected together in a grid cell, which leads to temporal stability and reduces flickering.

During rendering, we locate the smallest grid in the cascade that contains the point to be shaded and sample the indirect lighting from that grid. In order to get a smooth transition between two levels in the grid, we adopted the approach from the LPV paper. Their approach is based on the work done in [LH04]. [LH04] proposes to set a transition width, w, that determines the strength of the interpolation. Given the center of the grid, C, and the point to be shaded, P, the interpolation factor, a, is computed using equation 3.2.

$$a = \frac{P - C - (\frac{gridArea}{2} - w)}{w} \tag{3.2}$$

a is clamped to be in the range of 0 to 1. Equation 3.2 returns a three dimensional vector, and the actual interpolation factor is the largest component of this vector. The clipmap paper proposes to set w using the heuristic $w = n/10$, with n being the area of the grid.

3.4 Voxel Cone Tracing

Path traced solutions solve the global illumination problem by tracing a large number of rays in a random direction. Cone tracing presents an alternative that provides an estimate of a large number of rays in a single query. Cones, like

rays, have an origin and a direction. Unlike a ray, a cone has an angle. As the sampling point travels farther from the origin of the cone, the sampling radius gets bigger. Figure 3.16 shows an example of a cone.

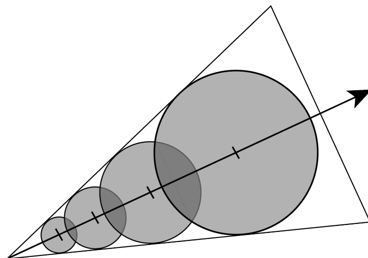


Figure 3.16: Example of a cone. The sampling radius gets bigger as the distance between the sampling point and the origin of the cone increases.

Using cones for the purpose of global illumination was first introduced in [CNS+11] by Crassin et al. [CNS+11] proposes to store a voxelized version of the scene in a sparse octree. This tree is then filtered. The filtering implies that each node in the higher levels of the tree holds the approximate illumination of all its child nodes. When tracing a cone through this tree, the radius of the cone at the current sampling point determines the level of the tree the illumination is retrieved from.

By tracing several combinations of cones, different forms of global illumination can be approximated, for example indirect diffuse lighting, specular reflections and refractions. In order for voxel cone tracing to work, the scene needs to first be *voxelized*.

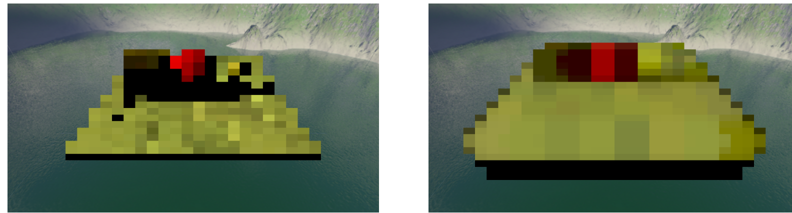
3.4.1 Voxelization

The voxelization approach implemented for this research is based on the chapter *Octree-Based Sparse Voxelization Using the GPU Hardware Rasterizer* of the *OpenGL Insights* book [CG12], written by the same author as the voxel cone tracing paper. OpenGL inherently did not have the capability to write to a random texel of a texture, so previous approaches to voxelization had to render each 'slice' of the 3D space separately. OpenGL 4.3 however introduces *ImageStore* functions, which allows to write values to an arbitrary location in the texture. This makes it possible to voxelize an entire mesh in a single draw call. The octree in the original paper requires two passes to fill with data. The first step voxelizes the scene and constructs an octree. A second step renders the scene from the perspective of each light source to inject radiance into the octree. Next, the tree is filtered. The filtering means that each node in the tree will hold the approximate irradiance of all its child nodes. The paper uses a Gaussian weight to compute the final irradiance of an octree node.

For this thesis we opted to use a 3D grid instead of an octree. Using a grid compared to an octree has several advantages. The first one is that it is a lot

easier to implement. The second advantage is that it allows us to compute the lighting at each voxel directly, using a forward shading pass. This mitigates the use of a separate light render pass. The third advantage is that the 3D grid is implemented using a 3D texture, which means that filtering can be done using the build in *glGenerateMipMaps()* function. The major downsides of using a grid is the large memory requirement and that it divides the space equally. The sparse voxel octree proposed in the paper puts more nodes, and therefore greater detail, in locations where triangles are located. The cubic shape of the grid means that some grid cells will be empty, and could therefore be considered 'wasted'.

Figure 3.17 shows the voxelized result of a scene at the base level, as well as the filtered result at level two.



(a) Base level of a voxelized scene. (b) Second level of a voxelized scene.

Figure 3.17: Voxelized result of a scene. The images were generated using the engine written for this research.

Implementation Details

Initially, the voxel grid was implemented using a single 3D texture, with the red, green and blue components holding the irradiance of the voxel and the alpha component storing an opacity factor. This opacity factor is required to determine if a voxel is empty or filled. Unfortunately, this led to a lot of flickering within the voxelized scene. The cause of this is race conditions. As each object is rendered, the fragments are processed in a random order. Since the voxelization will overwrite the current value of the voxel, the final result is largely dependent on the order in which fragments were processed. This is especially true at voxels with multiple intersecting objects.

[CG12] solved this issue using OpenGLs *imageAtomicCompSwap* method. When writing to a texel using this method, it will atomically compare the current result in the texture with the one we are trying to store. If they are equal, the result is stored in the grid, otherwise, the result is discarded. In either case the original value of the texel is returned. The solution in the OpenGL book uses the alpha component of the color as a counter to average the result of the voxelization. When a GPU thread tries to store the result into a voxel, the imageAtomicCompSwap is used to determine if another thread is also trying to access that voxel. If the value returned by imageAtomicCompSwap is not the same as the value we are trying to store, then another thread has accessed

the voxel. We then update the value of our result to represent an average of the value in the texel and our own value. Using this method, we can produce a flicker-free voxelized scene. Appendix B.5 shows example code to implement this atomic averaging. Unfortunately, we have lost the alpha channel to the atomic counter, so we introduced a separate alpha grid to store the opacity of the voxels.

In order to ensure maximum coverage of each voxelized triangle, every triangle is projected onto its dominant axis. The dominant axis is the one that provides the maximum value for $\mathbf{n} \cdot \mathbf{v}$, with $\mathbf{n}$ being the triangle normal and $\mathbf{v}$ being one of the three primary axes in the scene. In the geometry shader of the voxelization program, the triangle is projected using an orthographic projection along the dominant axis. This simple approach to voxelization does not produce a 'correct' vozelization. Because a triangle will only provide a fragment if it covers the center of the pixel, this voxelization approach can lead to holes in the final voxelized scene. The solution is to use a more precise *conservative rasterization* approach. The general idea is to enlarge the projected triangle by half a pixel. This is done by shifiting the edges of each triangle outward in the geometry shader. Since the exact shape of the bounding polygon that does not overestimate the coverage of the triangle is a polygon, a bounding box is computed alongside the triangle. The fragments outside of this box are killed in the fragment shader. Figure 3.18, taken from [CG12], shows the bounding box of an enlarged triangle and which fragments will be killed in the fragment shader. The conservative rasterization approach ensures that a fragment is generated for every pixel that is touched by a triangle. Listing B.4 shows example geometry shader code for conservative voxelization.

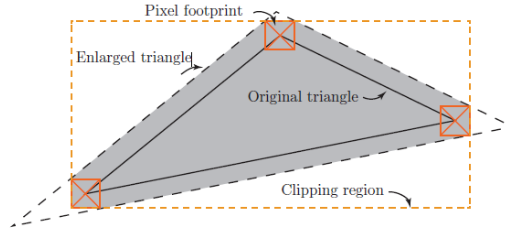


Figure 3.18: Conservative rasterization. The bounding polygon of the enlarged triangle kills off the excess fragments in the corners of the triangle.

3.4.2 Cone Tracing

A cone has an origin, a direction and a radius. When sampling a value from the grid, the sampling point moves along the direction of the grid. The radius at the sampling point determines the level of the mip-map to sample from. Using simple geometry, the radius, r, for sampling point p at distance d, of a cone with angle A, can be computed as $r = \tan(A/2) * d$, see figure 3.19. From this radius the level in the grid to sample from can be computed using equation 3.3.

In equation 3.3, *voxelSize* is the size of a voxel at the lowest level. In our case, this corresponds to the Euclidian distance between two corners of a voxel.

$$level = \log_2(\frac{2 * r}{voxelSize}) \tag{3.3}$$

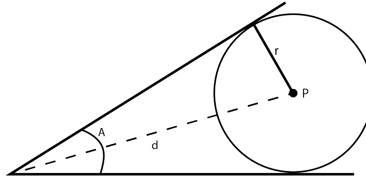


Figure 3.19: Using simple geometry, the radius r at point p at distance d of a cone with angle A, can be computed as $r = \tan(A/2) * d$.

After sampling from the grid, the sampling distance is moved by $2 * r$, see figure 3.20. This means that there is some overlap in the irradiance we retrieve, as well as corners we will miss.

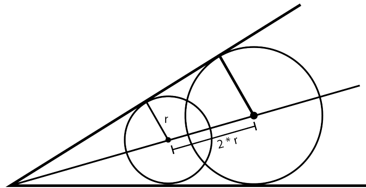


Figure 3.20: After a sample has been taken, the sampling distance is moved by $2 * r$.

A big issue with the cone tracing method is avoiding self-intersection. In path tracing this is done by adding a small nudge factor to each ray. With cones we also add a nudge factor. Intuitively, the minimum nudge factor should be the size of the smallest voxel plus the sampling radius of the point, to make sure we don't fetch from the voxel the cone originated from. Rauwendaal [Rau13] observed that this is not enough if the direction of the cone differentiates from the surface normal. Figure 3.21 illustrates this concept. The nudge factor is computed using equation 3.4.

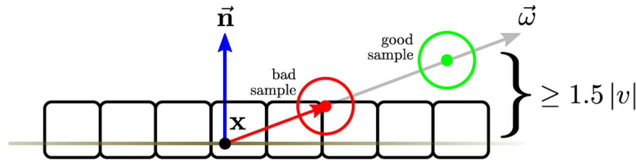


Figure 3.21: A nudge factor is computed based on the angle between the surface normal, n, and the cone direction ω. The figure was taken from [Rau13].

$$nudgeFactor = \frac{voxelSize}{n \cdot \omega} \tag{3.4}$$

The cone tracing is terminated after the accumulated opacity has reached one, or when a certain distance has been reached. Cones with a bigger cone angle take larger steps through the volume, so they terminate quicker. Performance-wise this means that a cone with a large angle is usually faster than one with a small angle.

Implementation Details

Listing 3.2 shows pseudo code to implement the cone tracing algorithm.

Listing 3.2: Cone tracing pseudo code

```
vec4 coneTrace(vec3 origin, vec3 direction, vec3 normal,
   ↪ float coneAngle, float maxDist, float voxSize,
   ↪ float nudge){
   vec4 accum = vec4(0);
   float tanA = tan(coneAngle/2);

   float h = nudge * voxSize;
   float nDotS = dot(normal, direction);

   float dist = h / nDotS;

   while (dist < maxDist && accum.a < 1.0){
      float sampleRadius = tanA * dist;
      float sampleDiameter = 2 * sampleRadius
      float sampleLOD = log2(sampleDiameter / voxSize);
      vec3 samplePos = origin + direction * distance;

      vec4 sampleValue = fetchVoxelValue(samplePos,
         ↪ sampleLod);

      accum += (1.0 - accum.a) * sampleValue;
```

```
        dist += sampleDiameter;
    }
    return accum;
}
```

In listing 3.2, the *coneAngle* parameter is the angle of the cone in radiance, *maxDist* is the maximum distance a cone can travel, *voxSize* is the size of the smallest voxel in our scene. In our case of isotropic voxels, this is the Euclidean distance between two opposing corners of a voxel from the lowest level grid. The final parameter, *nudge*, is a user specified parameter to influence how far the cone is pushed outwards from the starting point. This helps prevent self intersection, but may cause small objects to be 'missed' by the cone tracing.

The $fetchVoxelValue$ function samples the voxel volume at the specified sample level to retrieve the irradiance. In our isotropic case, this is just a single texture fetch in the 3D texture (actually, we have two texture fetches, one for the colors and one for the opacity factor). Alternative solutions exists, such as anisotropic voxels or spherical harmonics.

The default nudge factor in our implementation is a value of two, but even then self intersection can not be fully avoided. This is caused by the filtered structure of the voxelization. The starting distance is computed to make sure we do not sample from the same voxel as the cone originates from, with regard to the lowest level of the filter chain. Especially with cones with a large angle, it is very possible that subsequent steps sample from a higher level in the filter structure, which means it is still possible to have some form of self intersection.

3.4.3 Indirect Diffuse Lighting

Indirect diffuse lighting can be approximated by tracing several cones with a large angle around the normal of the point to be shaded. This means that we need to find a set of cones that cover a hemisphere as closely as possible. For this research we tested different options, with a varying number of cones and cone angles. In the end, we decided to use nine cones with an angle of 60 degrees. One cone is traced in the direction of the normal, the other eight are traced to the top corners and the sides of the box around the normal. Figure 3.22 shows the result of using voxel cone tracing for diffuse global illumination.

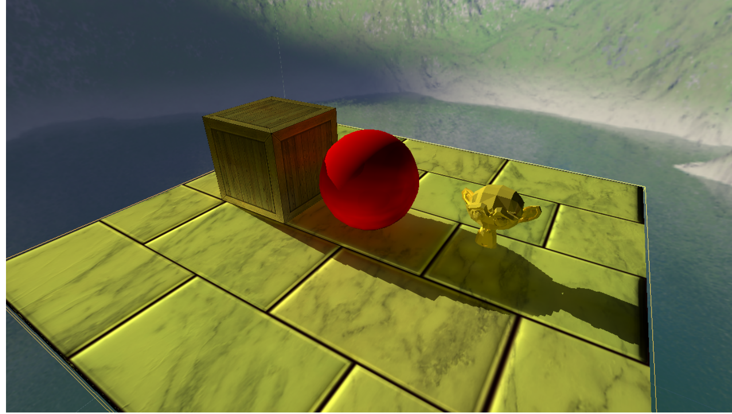


Figure 3.22: Diffuse global illumination through voxel cone tracing. Note the colored highlights on the bottom of the box and the sphere. For demonstration purposes the indirect lighting has been exaggerated. The image was created using the engine written for this research.

3.4.4 Indirect Specular Lighting

Specular reflections are approximated by tracing a single cone in the direction of the camera reflected around the normal. The angle of the cone is linearly interpolated between an one degree and an eighty degree cone angle based on the roughness of the material. A better method would be to match the angle of the cone based on the BRDF of the material. Since for this research we will only consider the diffuse part of global illumination, no extra effort was used to implement a proper BRDF fitting. Figure 3.23 shows the result of using cone tracing in combination with irradiance maps for specular reflections.

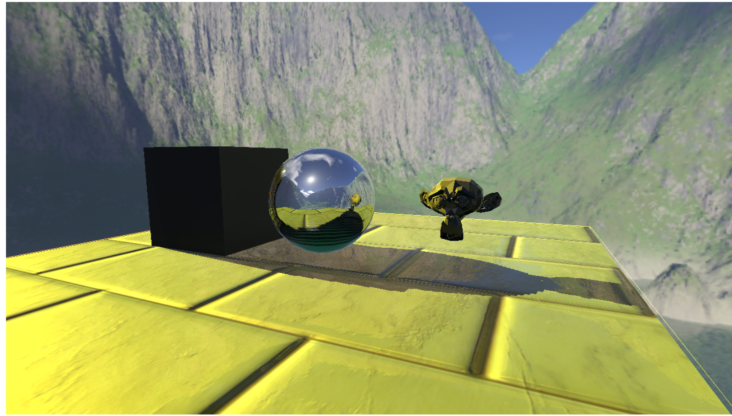


Figure 3.23: Specular reflections using voxel cone tracing. The environment reflection retrieved using irradiance maps is blended with the voxel cone tracing result using the opacity factor from the cone tracing. For demonstration purposes, the sphere is a perfect mirror. The image was created using the engine written for this research.

3.4.5 Spherical Harmonics

So far we have stored isotropic values in our voxels. This means we have discarded any form of directional information. As with the Light Propagation Volumes algorithm described in 3.3, we can retain directional information by storing spherical harmonics in the voxels. This requires two extra textures, since we now need one texture for each color component. The implementation details of spherical harmonics are very similar to the one used for the LPV algorithm and we refer the reader to the relevant part of section of 3.3 for more information. When we use the atomic average method described in the voxelization chapter, we lose the alpha component of the three color textures. We therefore require four textures to correctly store the SH information for each color channel. Another issue arises with the atomic average method used to store the voxel values. In the case of spherical harmonics, we need to write to different textures so we need to call this function multiple times. This led to race conditions, with the result being that the fragment shader never completes. The solution was to add a counter to the code of the image atomic average function. The count makes sure that the while loop of the function is executed a maximum of 1000 times. This led to the behavior expected, without a noticable loss in quality.

The $fetchVoxelValue$ function of listing 3.2 for spherical harmonics transforms the inverse of the direction of the cone to the spherical harmonics and accumulates the intensity by sampling from the three textures.

3.4.6 Cascaded Voxel Cone Tracing

The same problem described in the cascaded section of the Light Propagation Volume algorithm described in section 3.3.5 can be applied to the voxel grid. The same solution of cascaded voxel grids can also be applied. The details of the cascaded structure are the same as with the LPV algorithm, so we will not repeat those steps here. The reader is encouraged to read the relevant part of section 3.3.5 for more information.

During the implementation we encountered a problem when moving the cascaded grids throughout the scene. The movement caused a lot of flickering, something that should have been fixed by snapping the grid positions to the grid cells. The problem with cone tracing compared to light propagation volumes, is that the cones also sample from higher mip-map levels. Snapping the position to a grid cell makes the base level of the grid temporally stable. Instead of snapping to a single grid cell, we snap the position to a multiple of the grid cell. We found that a factor of $gridresolution/8$ provides good results. Intuitively, the divide by eight means that all but the top three ($8 = 2^3$) mip-map levels of the texture are stable. There is still some very slight flickering, due to these high level mip-maps not being stable, but the effect is barely noticable.

3.4.7 Other Uses

The result of a cone trace through a voxelized scene is the approximation of the result of firing a large number of rays from the cone origin in the direction of the cone. This approximation can be used to get estimates of different effects at interactive rates.

Emissive Lighting Voxel cone tracing has for now been used as a supplement to direct lighting, by calculating the amount of indirect lighting. Similarly, cone tracing can be used to evaluate direct lighting. Instead of storing the accumulated irradiance in the voxels, the voxels store the actual radiance at that location. Using this technique, a lot of lights can be evaluated at once or emissive materials can be rendered. Figure 3.24 shows the result of using voxel cone tracing to light a scene using several emissive cubes instead of regular light sources.

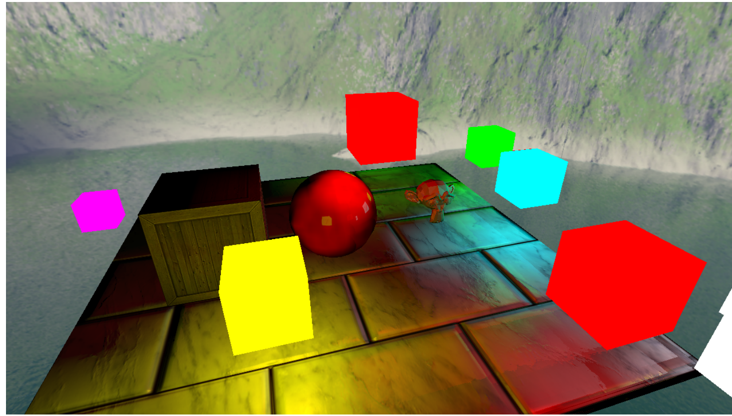


Figure 3.24: Emissive lighting using voxel cone tracing. The only lighting present in this scene are the emissive cubes. All the light, both diffuse and specular, is gathered using voxel cone tracing. The image was created using the engine written for this research.

Refraction Very similar to how a single cone can approximate specular reflections, a single cone can also simulate refraction. Instead of tracing the cone away from the surface of the material, a cone is traced through the inside of the material. This cone is based on the refractive index of the material and the transparency. See figure 3.25 for an example of refractive cone tracing.

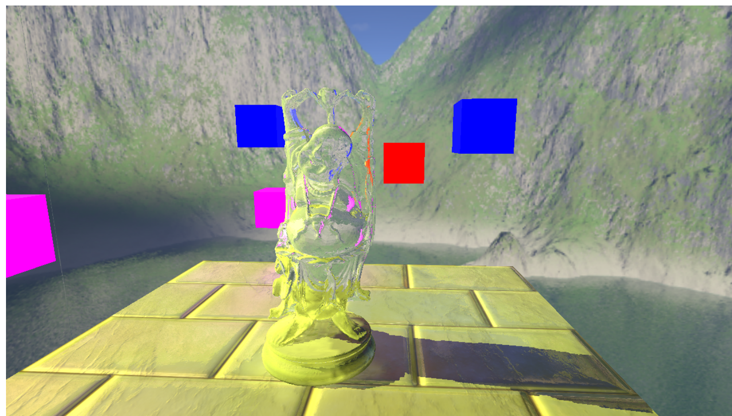
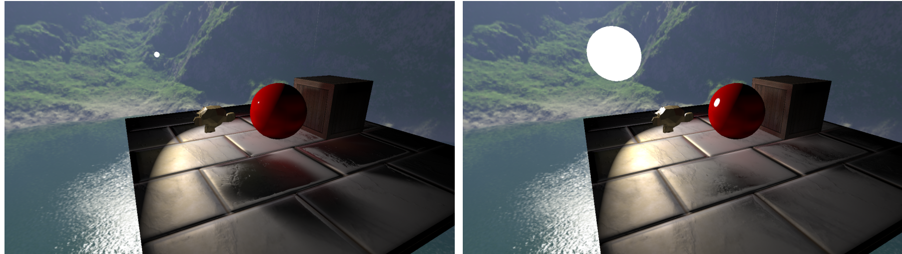


Figure 3.25: Refractions using voxel cone tracing. The environment reflection retrieved using irradiance maps is blended with the voxel cone tracing result using the opacity factor from the cone tracing. The image was created using the engine written for this research.

Soft Shadows Cone tracing can also be used to simulate soft shadows. By tracing a cone towards the light source and accumulating the occlusion along

the way, the amount of shadow can be computed. The angle of the cone is based on the radius of and the distance to the light source, with larger light sources generating a softer shadow. Figure 3.26 shows the result of using voxel cone tracing to approximate soft shadows.



(a) Soft shadow approximation using voxel cone tracing. The angle of the cone is based on the radius of the light source and the distance.

(b) A bigger light radius results in a softer shadow.

Figure 3.26: Soft shadows using voxel cone tracing. The image was created using the engine written for this research.

3.5 Ambient Occlusion

Indirect lighting is also called ambient lighting, since it has no clear location or direction, but is present everywhere. The amount of ambient lighting a location receives is determined partly by its surroundings. A corner or crevice will receive less light than a flat surface. Simulating this *Ambient Occlusion* can give a scene a sense of depth, leading to a more believable result. Several ambient occlusion algorithms have been devised over the years, with some of the most common ones today being based on *Screen Space* techniques.

Screen space techniques use the geometry output from a deferred renderer to produce different kind of effects, examples of screen space techniques are *Screen Space Reflections* and *Screen Space Ambient Occusion* (SSAO).

3.5.1 Screen Space Ambient Occlusion

The original SSAO algorithm was developed by crytek for the Crysis games. SSAO takes the depth buffer of the scene as input and uses this as a course approximation of the geometry. For each fragment, points are sampled in a sphere around the original sampling point, see figure 3.27.

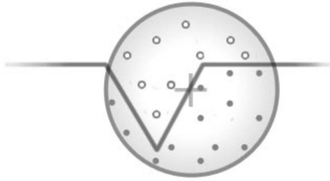


Figure 3.27: SSAO Point sampling in a sphere. The figure was taken from [Cha11].

Each sample point is projected into screen space to get the value of the depth buffer at that sample point. If the sample position is behind the sampled depth, i.e. it is 'inside' the geometry, it is considered occluded and it contributes to the ambient occlusion. The final occlusion is then determined as the percentage of samples that is considered inside the geometry. Since this approach samples a sphere, on average at least half the samples are occluded, leading to a dark look. Additionally, corners appear brighter, because on average only one quarter of the samples are occluded. The SSAO algorithm implemented for this research is based on a blog post by John Chapman [Cha11] and it mitigates this problem by sampling around a hemisphere instead, see figure 3.28.

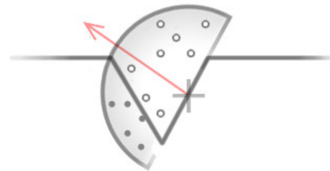


Figure 3.28: SSAO Point sampling in a hemisphere. The figure was taken from [Cha11].

The hemisphere requires that per-pixel normal data is available and this is the case in a deferred renderer. Samples for the algorithm are generated within the hemisphere, oriented along the Z axis. We use an accelerating interpolation function to make sure that more points are generated closer to the origin. This puts greater weight at the close surroundings of a pixel. To orient the samples along the fragment normal, a tangent vector is computed using a randomly generated noise texture. This noise texture is tiled accross the screen. This tiling introduces banding into the occlusion result. This banding is later removed in a separate blur pass after the ambient occlusion has been computed. Figure 3.29 shows the occlusion result for SSAO on a scene.

Figure 3.29: Ambient occlusion using Screen Space Ambient Occlusion. The image was created using the engine written for this research.

Over the years, several improvements on the original SSAO algorithm have been made in the form of for example *Horizon Based Ambient Occlusion* (HBAO) [Bra08] and *Screen Space Directional Occlusion* (SSDO) [RGS09]. Due to time constraints and the relatively minor effect of ambient occlusion on the final image, these improvements were not implemented for this research.

3.5.2 Color Bleeding

Ritschel, Grosch and Seidel introduced an improvement to SSAO in [RGS09] called *Screen Space Directional Occlusion*. Alongside SSDO, they provides a way to introduce a single bounce of indirect lighting. Samples that are considered occluders during the SSAO pass can potentially bounce light in the direction of the sampling position. The amount of indirect lighting is computed using equation 3.5.

$$L_{ind}(\mathbf{P}) = \sum_{i=1}^{n} L_{pixel} V(i) \frac{A_s \cos\theta_{si} \cos\theta_{ri}}{d_i^2} \tag{3.5}$$

In equation 3.5, d_i is the distance between the fragment position $\mathbf{P}$ and occluder i, θ_{si} and θ_{ri} are the angles between the sender/receiver normal and the transmittance vector, defined as $P_i - \mathbf{P}$. A_s is the area the occluder occupies. Since the samples are defined around a hemisphere, the sample area can be approximated as $A_s = \pi r^2/N$, with r being the sampling radius. $V(i)$ is the visibility factor, which is the amount sample i contributes to the final occlusion value.

Figure 3.30 shows the result of applying equation 3.5 in the original SSAO algorithm to compute indirect lighting.

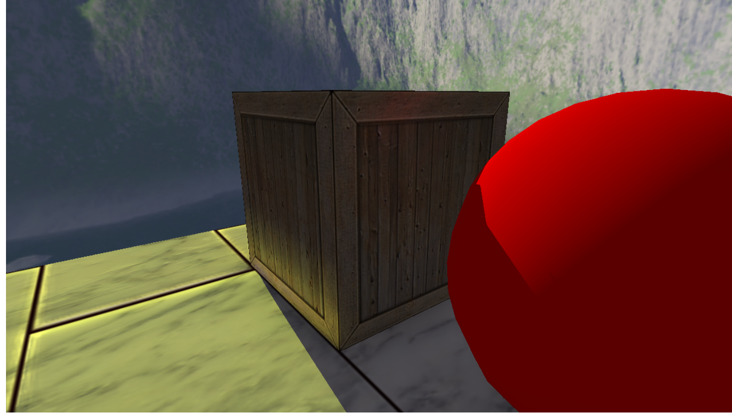


Figure 3.30: Indirect lighting using SSAO. Notice the yellow color at the base of the cube and the (very) slight red color on the cube. For demonstration purposes the indirect lighting has been exaggerated. The image was created using the engine written for this research.

The same limitations as with all screen space techniques still apply here. This is mainly visible on the crate in figure 3.30. It shows a red highlight from the sphere, but this highlight is cutoff towards the center of the cube. This is because it only has the screen space information available and all the normals of the sphere point away from the cube. It will therefore incorrectly assume no light is bounced in the direction of the cube.

During the implementation we found that the result of the indirect lighting is largely dependent on the configuration of the samples. Even with a lot of samples, color bleeding can simply not occur if for certain configurations of objects the samples are oriented poorly. The accelerating interpolation function, used to generate more points closer to the origin, amplifies this problem.

3.5.3 Voxel Cone Traced Ambient Occlusion

Our voxel structure has all the elements required to compute ambient occlusion. In fact, the alpha component of our cone tracing holds the opacity of the trace, which is the ambient occlusion. To compute ambient occlusion with voxel cone tracing, we repeat the steps required for diffuse indirect lighting, but we ignore the color components of the trace. The result of using this *Voxel Ambient Occlusion* (VAO) can be seen in figure 3.31.

Figure 3.31: Ambient occlusion using Voxel Cone Tracing. The image was created using the engine written for this research.

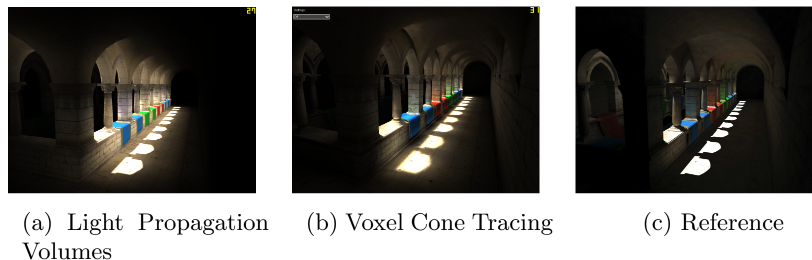
As both figure 3.29 and 3.31 show, voxel cone traced ambient occlusion gives a greater sense of depth to a scene.

4. Error Metric

An error metric is a mathematical function that measures the amount of error between the parameters of a ground truth reference and a set of parameters whose error is measured. Over the years many error metrics for many different purposes have been devised. For global illumination we are mostly interested in perceptually based metrics. This means that two images can be mathematically very different, for example because one image is slightly brighter or darker than the other, but most people will not notice this difference.

One of the most commonly used error metrics today is the *Mean Squared Error* (MSE). MSE is the average of the squared intensity difference of two input signals. Mathematically, the MSE is computed using the following formula: $\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$. By taking the root of the mean squared error, one obtains the *Root Mean Squared Error* (RMSE). The RMSE can be thought of as the distance (error) between two data points. In our case of image analysis, each datapoint is a pixel. RMSE is a very simple metric, which does not take into account the intricacies of the human visual system (HVS). Different *perceptual* metrics have been devised that try to mimic the complex internal workings of the HVS.

Because it is difficult to capture the intricacies of global illumination in a single metric, most algorithms have relied on a visual comparison for their assessment. This usually takes the form of showing their method, next to images of a ground truth reference and an older method, to showcase the improvement in their technique. Figure 4.1 shows how Crassin et al compared their Voxel Cone tracing Technique against Light Propagation Volumes and a reference.



(a) Light Propagation Volumes (b) Voxel Cone Tracing (c) Reference

Figure 4.1: Visual comparison between Light Propagation Volumes, Voxel Cone Tracing and ground truth. The images were taken from [CNS+11].

4.1 Structural Similarity

For this research we decided to use the *Structural Similarity* (SSIM) based metric, introduced by Wang in [WBSS04]. The paper introduces the SSIM index, which tries to evaluate two sets of images by separating the image in three components, luminance, contrast and structure. These components work relatively independently of each other. Intuitively it makes sense that this metric will give better estimates than something like RMSE, since our example case of having one image slightly brighter or darker will only affect the luminance component and therefore have a lessened impact on the total error.

4.1.1 Luminance

Firstly, the luminance of an image x, μ_{image}, is estimated as the mean intensity of an image, shown in equation 4.1.

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.1}$$

The luminance compare function, $l(\mathbf{x},\mathbf{y})$ is then a function of μ_x and μ_y. The luminance compare function is defined in equation 4.2.

$$l(\mathbf{x},\mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4.2}$$

The C_1 factor is a constant included to avoid instability when $\mu_x^2 + \mu_y^2$ is close to zero. The paper proposes to compute this factor as $C_1 = (K_1L)^2$, with L being the dynamic range of the pixel values (255 in the case of 8-bit images) and $K_1 << 1$.

4.1.2 Contrast

To estimate the amount of contrast in an image, SSIM uses the standard deviation as an estimate. The standard deviation σ_{image} is defined in equation 4.3.

$$\sigma_x = (\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)^2)^{\frac{1}{2}} \tag{4.3}$$

The contrast comparison $c(\mathbf{x},\mathbf{y})$ is the comparison of σ_x and σ_y. The contrast comparison functions has a similar form as the luminance compare function. It is defined in equation 4.4.

$$c(\mathbf{x},\mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{4.4}$$

In equation 4.4, $C_2 = (K_2L)^2$, and $K_2 << 1$.

4.1.3 Structure

The final component of the metric, the structure comparison, is conducted on a normalized version of the images. The images are normalized by dividing them by their own standard deviation, $(\mathbf{x}-\mu_x)/\sigma_x$. The structure comparison is then computed using equation 4.5.

$$s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3} \tag{4.5}$$

[WBSS04] proposes to set C_3 as $\frac{C_2}{2}$. σ_{xy} can be estimated using equation 4.6.

$$\sigma_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y) \tag{4.6}$$

4.1.4 SSIM

Using all of the three functions defined previously, the SSIM index between images $\mathbf{x}$ and $\mathbf{y}$ is computed using equation 4.7.

$$SSIM(\mathbf{x},\mathbf{y}) = [l(\mathbf{x},\mathbf{y})]^{\alpha} \cdot [c(\mathbf{x},\mathbf{y})]^{\beta} \cdot [s(\mathbf{x},\mathbf{y})]^{\gamma} \tag{4.7}$$

In equation 4.7, α, β and γ are parameters used to adjust the relative imporance of the three components. If we set $\alpha = \beta = \gamma = 1$ and $C_3 = C_2/2$, then the SSIM index results in the form given in equation 4.8.

$$SSIM(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4.8}$$

4.1.5 Image Quality Assesement

The paper proposes to apply the SSIM index locally rather than globally. There are several reasons for this approach. Firstly, image features are usually spatially nonstationary. Secondly, image distortions may also be space-variant. Thirdly, only a local area in the image can be perceived with high resolution by a human observer, due to the foveated nature of the HVS. The paper uses an 11 x 11 circular Gaussian weighted window to compute the local SSIM index. In order to get a single quality value of the entire image, the mean SSIM (MSSIM) is computed using equation 4.9.

$$MSSIM(\mathbf{X},\mathbf{Y}) = \frac{1}{m} \sum_{i=1}^{m} SSIM(\mathbf{x}_i, \mathbf{y}_i) \tag{4.9}$$

In equation 4.9, $\mathbf{X}$ and $\mathbf{Y}$ are the reference image and the image to compare, $\mathbf{x}_i$ and $\mathbf{y}_i$ are the SSIM indices at the ith local window, with a total of m windows. m is the total number of pixels in the image, with a window being generated for every pixel.

[Hof12] proposes to use the metric to automatically detect if an image conversion results in an 'acceptable' loss in quality, for example when compressing a png image into a jpg format. One can reason that using an approximate global illumination method will result in a loss in quality.

4.2 Modifications

The most prominent effect of diffuse global illumination is color bleeding. However, by computing the luminance based on the average color, we lose all color information in this step. We therefore opted to compute the SSIM index on each color component separately. The final index is computed as the average of the MSSIM of an image.

5. Experimental Setup

For this experiment two test scenes were utilized. The first scene is a simple demo scene that is set up specifically to show the properties of diffuse global illumination (color bleeding). The second scene is the Sponza scene, by Crytek. This is a scene of a roman style atrium that was first used as an example for the LPV algorithm. It was specifically designed to highlight the problems faced by global illumination. Since then, most global illumination algorithms have included the Sponza scene in their test set. For each scene, we took two camera configurations, one that gives a rough overview of the entire scene and one that should highlight the color bleeding effect.

The reference images were created using a custom path tracer. Using this custom path tracer, we are ensured that color details such as gamma correction are identical between the reference images and the images to compare. It also made it very easy to build the scene and set up the camera in the path tracer. Unfortunately, the path tracer has no support for specular surfaces. This means that we can only compare the diffuse contribution of global illumination in this research.

For the metric we implemented the SSIM index described in section [**?**]. We adopted the same parameters as the one proposed in [WBSS04]. Specifially, $K_1 = 0.01$, $K_2 = 0.03$ and $C_3 = \frac{C_2}{2}$. We also adopted their approach of computing the index in a local 11 x 11 Gaussian window. The result of this metric is a similarity score between zero and one, that indicates how similar the two images are. A higher score equals a better estimate.

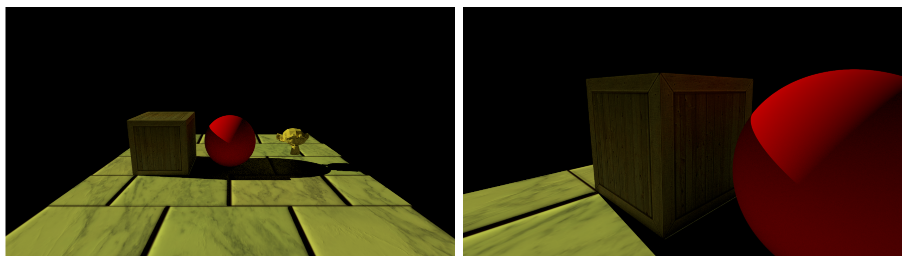
During the implementation of the metric we found that the intensity component of the SSIM index was almost solely responsible for the error. In hindsight this makes sense, as both the compare and the reference image show the same scene, with the reference image having more intensity (global illumination) in the dark parts of the image. When using the SSIM index given in equation 4.8, this resulted in a very high score. Since both images show the same scene from the same angle, the contrast and structure comparisons will be almost identical. As it turns out the intensity factor was not enough to make a large difference on the similarity score. All the tests scored a 9.999 or higher. We therefore opted to raise the score by a power of 1000, making our final SSIM index, $index(\mathbf{x}, \mathbf{y}) = SSIM(\mathbf{x}, \mathbf{y})^{1000}$. The value of 1000 is an arbitraly chosen value that brings the range of the similarity score within an acceptable range. The SSIM score is multiplied by a factor of ten, to get a grade between zero and ten.

In the case of the basic test scene, we noticed that the scene does not take up a majority of the screen. This means that a large part of the screen is black in both the reference and the compare image. The SSIM index in these parts will result in one (i.e. no difference). This also contributed to the very high

score. To circumvent this issue, we modified the SSIM index to ignore pixels where both the compare and the reference pixel are black.

5.1 Basic Scene

The two reference images used for the basic scene can be seen in figure 5.1. The dimension of these images is 1280 by 720 pixels.



(a) Reference image 1.
This image shows a general overview of the scene. Global illumination is mostly visible at the bases of the cube and the sphere, with some slight red color bleeding on the floor.

(b) Reference image 2.
This image focuses on the colorbleeding of the sphere on the side of the cube.

Figure 5.1: Reference images for the basic scene. Created using the custom path tracer.

The basic scene is compared using the two algorithms described in this thesis, Light Propagation Volumes and Voxel Cone Tracing. Light Propagation Volumes are executed with a grid size of 32 and 64, respectively, while Voxel Cone Tracing is executed with a grid size of 32 and 128. The resolution of the Light Propagation Volumes was dropped because of the issues discussed in section 3.3.3. In addition, each algorithm is tested with Screen Space Ambient Occlusion enabled and disabled. In the case of Voxel Cone Tracing, Voxel Ambient Occlusion is tested as well. Voxel Cone Tracing is also tested using isotropic voxels and with spherical harmonics. In all cases the area of the grid is set to cover the entire scene. The nudge factor for the Voxel Cone Tracing is set to 1 for this scene. This leads to the following list of algorithmic configurations that are tested for the basic scene:

Light Propagation Volumes, dimensions 32 x 32 x 32.

Light Propagation Volumes, dimensions 64 x 64 x 64.

Light Propagation Volumes, dimensions 32 x 32 x 32, with SSAO.

Light Propagation Volumes, dimensions 64 x 64 x 64, with SSAO.

Voxel Cone Tracing, dimensions 32 x 32 x 32.

Voxel Cone Tracing, dimensions 128 x 128 x 128.

Voxel Cone Tracing, dimensions 32 x 32 x 32, with SSAO.

Voxel Cone Tracing, dimensions 128 x 128 x 128, with SSAO.

Voxel Cone Tracing, dimensions 32 x 32 x 32, with voxel ambient occlusion.

Voxel Cone Tracing, dimensions 128 x 128 x 128, with voxel ambient occlusion.

Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32.

Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128.

Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32, with SSAO.

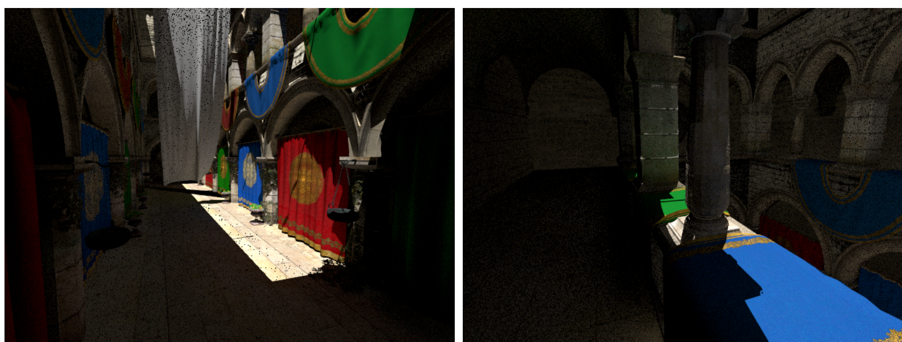
Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128, with SSAO.

Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32, with voxel ambient occlusion.

Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128, with voxel ambient occlusion.

5.2 Sponza Scene

The two reference images used for the sponza scene can be seen in figure 5.2. The dimension of these images is 640 by 480 pixels.



(a) This image shows a general overview of the scene. Without global illumination, the entire left portion of the screen is dark.

(b) This image focuses on the color-bleeding of the cloth on the sides of the pillar.

Figure 5.2: Reference images for the sponza scene. Created using a custom path tracer.

The same algorithmic configurations that were used for the basic scene were also tested with the sponza scene. In addition, for both Light Propagation Volumes and Voxel Cone Tracing a cascaded approach is tested. The small size of the basic scene does not make it interesting to test the cascaded approach. The sponza scene, however, is much bigger, requiring finer detail. For the cascaded approach four cascaded grids were used. The covered area for the lowest grid is set to 10x10x10, with the area doubling each subsequent cascade. For Voxel Cone Tracing, the nudge factor was set to 0.5. This leads to the following list of algorithmic configurations for the sponza scene:

Light Propagation Volumes, dimensions 32 x 32 x 32.

Light Propagation Volumes, dimensions 64 x 64 x 64.

Light Propagation Volumes, dimensions 32 x 32 x 32, with SSAO.

Light Propagation Volumes, dimensions 64 x6432 x 64, with SSAO.

Cascaded Light Propagation Volumes, dimensions 32 x 32 x 32.

Cascaded Light Propagation Volumes, dimensions 64 x 64 x 64.

Cascaded Light Propagation Volumes, dimensions 32 x 32 x 32, with SSAO.

Cascaded Light Propagation Volumes, dimensions 64 x 64 x 64, with SSAO.

Voxel Cone Tracing, dimensions 32 x 32 x 32.

Voxel Cone Tracing, dimensions 128 x 128 x 128.

Voxel Cone Tracing, dimensions 32 x 32 x 32, with SSAO.

Voxel Cone Tracing, dimensions 128 x 128 x 128, with SSAO.

Voxel Cone Tracing, dimensions 32 x 32 x 32, with voxel ambient occlusion.

Voxel Cone Tracing, dimensions 128 x 128 x 128, with voxel ambient occlusion.

Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32.

Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128.

Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32, with SSAO.

Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128, with SSAO.

Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32, with voxel ambient occlusion.

Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128, with voxel ambient occlusion.

Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32.

Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128.

Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32, with SSAO.

Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128, with SSAO.

Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32, with voxel ambient occlusion.

Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128, with voxel ambient occlusion.

Cascaded Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32.

Cascaded Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128.

Cascaded Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32, with SSAO.

Cascaded Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128, with SSAO.

Cascaded Voxel Cone Tracing using Spherical Harmonics, dimensions 32 x 32 x 32, with voxel ambient occlusion.

Cascaded Voxel Cone Tracing using Spherical Harmonics, dimensions 128 x 128 x 128, with voxel ambient occlusion.

6. Results & Evaluation

6.1 Results

Table 6.1 and 6.2 shows the scores, rounded to the nearest decimal, for all algorithmic and camera configurations for the basic and the sponza scene. Appendix A shows the images used as comparison and the resulting SSIM image.

Basic scene		
Algorithm	**Configuration 1**	**Configuration 2**
LPV 32x32x32	7.4	7.5
LPV 32x32x32 with SSAO	7.4	7.5
LPV 64x64x64	7.5	7.3
LPV 64x64x64 with SSAO	7.6	7.0
VCT 32x32x32	6.7	5.7
VCT 32x32x32 with SSAO	6.8	5.7
VCT 32x32x32 with VAO	6.9	6.2
VCT 32x32x32 with SH	7.9	8.0
VCT 32x32x32 with SSAO and SH	8.0	8.1
VCT 32x32x32 with VAO and SH	8.1	8.8
VCT 128x128x128	7.7	8.1
VCT 128x128x128 with SSAO	7.8	8.2
VCT 128x128x128 with VAO	7.8	8.4
VCT 128x128x128 with SH	8.1	8.2
VCT 128x128x128 with SSAO and SH	8.2	8.2
VCT 128x128x128 with VAO and SH	8.2	8.2

Table 6.1: Error metric scores (rounded to the nearest decimal) for the basic scene.

Sponza scene		
Algorithm	**Configuration 1**	**Configuration 2**
LPV 32x32x32	6.9	7.2
LPV 32x32x32 with SSAO	7.1	7.1
Cascaded LPV 32x32x32	7.2	7.1
Cascaded LPV 32x32x32 with SSAO	7.3	7.1
LPV 64x64x64	7.1	7.2
LPV 64x64x64 with SSAO	7.1	7.2
Cascaded LPV 64x64x64	6.8	7.2
Cascaded LPV 64x64x64 with SSAO	6.8	7.1
VCT 32x32x32	6.0	5.7
VCT 32x32x32 with SSAO	6.3	6.2
VCT 32x32x32 with VAO	7.4	7.0
VCT 32x32x32 with SH	7.5	7.3
VCT 32x32x32 with SSAO and SH	7.5	7.2
VCT 32x32x32 with VAO and SH	6.7	7.2
Cascaded VCT 32x32x32	6.5	6.1
Cascaded VCT 32x32x32 with SSAO	6.6	6.5
Cascaded VCT 32x32x32 with VAO	7.5	7.4
Cascaded VCT 32x32x32 with SH	7.6	7.4
Cascaded VCT 32x32x32 with SSAO and SH	7.6	7.3
Cascaded VCT 32x32x32 with VAO and SH	7.0	7.4
VCT 128x128x128	6.8	6.4
VCT 128x128x128 with SSAO	6.9	6.6
VCT 128x128x128 with VAO	7.4	7.0
VCT 128x128x128 with SH	7.3	7.4
VCT 128x128x128 with SSAO and SH	7.3	7.3
VCT 128x128x128 with VAO and SH	7.0	7.3
Cascaded VCT 128x128x128	7.6	6.8
Cascaded VCT 128x128x128 with SSAO	7.6	6.8
Cascaded VCT 128x128x128 with VAO	6.5	7.1
Cascaded VCT 128x128x128 with SH	6.9	7.4
Cascaded VCT 128x128x128 with SSAO and SH	6.8	7.3
Cascaded VCT 128x128x128 with VAO and SH	6.5	7.2

Table 6.2: Error metric scores (rounded to the nearest decimal) for the sponza scene.

Figure 6.1 shows a plot of the scores for each algorithmic configuration for the sponza scene. Appendix A contains the figures for the other configurations and scenes.

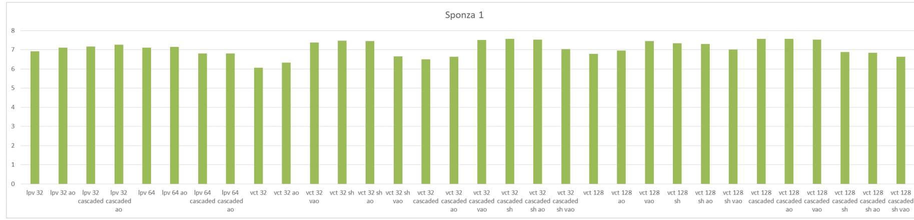


Figure 6.1: Results for the first configuration of the sponza scene.

6.2 Evaluation

Looking at figure 6.1 and the corresponding figures in appendix A, several interesting phenomena can be deduced. Light Propagation Volumes are designed with diffuse global illumination in mind and that shows in their score. There is barely any difference between the different settings for the LPV algorithm.

Voxel Cone Tracing is a method that facilitates diffuse global illumination, amongst other effects. This versatility is evident in the wider variety in scores for the different configurations. The scores show that Voxel Cone Tracing is much more affected by the set of parameters and these results are quite like we expected. A higher grid resolution leads to a better scoring, as does using a cascaded approach versus a non-cascaded one. The largest improvement can be made by storing spherical harmonics in a voxel compared to the isotropic case. Finally, there is a small improvement when using Screen Space Ambient Occlusion, but usually a much larger improvement for Voxel Ambient Occlusion. Interestingly, the combination of spherical harmonic and ambient occlusion leads to a worse score compared to using spherical harmonics only. In these cases, Voxel Ambient Occlusion leads to a lower score than Screen Space Ambient Occlusion.

Voxel Cone Tracing requires much more tweaking compared to Light Propagation Volumes. Examples of tweakable parameters are the number of cones for the diffuse illumination, the angle of these cones and the nudge factor to avoid self-intersection. An interesting notion that can be observed from table 6.1 and 6.2 is that for the VCT case, increasing the resolution sometimes leads to a worse estimate. A possible explanation for this can be found in the way cones are traced, see figure 3.16. With a higher resolution, the gaps we miss by stepping along the cone are much more noticable.

In general, we find that Light Propagation Volumes gives much more stable results. This is due to the strong blending effect that is inherent of the injection and propagation phase as well as the use of spherical harmonics. There are hardly any seams, hotspots or artefacts, leading to a stable result, regardless of resolution. Voxel Cone Tracing is much more reliant on its parameters to avoid artifacts.

7. Conclusion

7.1 Conclusion

In this research we developed a scoring metric to quantitatively measure the visual quality of an approximate global illumination algorithm. The scores for the different configurations of the algorithm, as shown in table 6.1 and 6.2, are consistent with the expected output for the scoring metric. Examples of this are a higher score when using ambient occlusion, spherical harmonics, or a cascaded approach. As discussed in section 6.2, the combination of spherical harmonics and ambient occlusion leads to a lower score. Future research will be required to determine the cause of this abnormality.

7.2 Future Work

This research has focused solely on diffuse global illumination. For future experiments we would like to look into specular global illumination as well in order to measure the full spectrum of global illumination. In addition, the scenic setup has been relatively simple in our case. We would like to expand this research with scenes with more interesting phenomena, such as a sky box, and particles. This would also allow us to compare more effects than only diffuse and specular global illumination, such as sub-surface scattering and refraction.

Additionally, we would like to conduct an experiment that will ask several test subjects to grade the resulting global illumination methods, to see if their results match up with the scores for the error metric. Additionally, the focus for this research has been on final image quality. In the future, we also would like to take performance characteristics such as computation time and memory requirement into account when determining the final score of an algorithm.

For future research we propose to take multiple camera orientations within each scene, more than the two used during this research, and average the results of all these orientations in order to get an score for a scene, instead of for a camera configuration.

Real-time global illumination remains an unsolved issue, that is still extensively researched. This means that improvements to existing algorithms and entirely new ones are expected to arrive over the years. For this research we focussed on the implementation of Light Propagation Volumes and Voxel Cone Tracing. There are many more interesting approximate gobal illumination algorithms, such as *Layered Reflective Shadow Maps* and *Signed Distance Fields*. In addition, many algorithms exists that require some form of preprocessing, such as Lightcuts. Ideally, the error metric should be tested with these algorithms as well as Light Propagation Volumes and Voxel Cone Tracing.

Bibliography

[Bra08] Louis Bravoil. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*, number 22, 2008.

[CG12] Cyril Crassin and Simon Green. *Octree-based Sparse Voxelization Using the GPU Hardware Rasterizer*, chapter 22, pages 303 – 321. CRC Press, 2012.

[Cha11] John Chapman. Ssao tutorial, 2011.

[CNS+11] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. In *Symposium on Interactive 3D Graphics and Games*, pages 207–217, 2011.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective Shadow Maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231, 2005.

[Gre03] Robin Green. Spherical harmonic lighting: The gritty details, 2003.

[Hof12] Naty Hoffman. Background: Physics and Math of Shading. In *Practical Physically Based Shading in Film and Game Production Course - SIGGRAPH 2012*, 2012.

[Kap09] Anton Kaplanyan. Light propagation volumes in cryengine 3. In *Advances in Real-Time Rendering in 3D Graphics and Games Course SIGGRAPH 2009*, 2009.

[Kar13] Brian Karis. Real Shading in Unreal Engine 4. In *Physically Based Shading in Theory and Practice Course - SIGGRAPH 2013*, 2013.

[KD09] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 99–107, 2009.

[Kel97] Alexander Keller. Instant Radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, 1997.

[Lag12] Sébastien Lagarde. Image-based Lighting and parallax-corrected cubemap, 2012.

[LH04] Frank Losasso and Hugues Hoppe. Geometry clipmaps: Terrain rendering using nested regular grids. In *ACM SIGGRAPH 2004 Papers*, pages 769–776, 2004.

[Rau13] Randall Rauwendaal. *Voxel Based Indirect Illumination using Spherical Harmonics*. PhD thesis, Oregon State University, 2013.

[RGS09] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating Dynamic Global Illumination in Screen Space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 75–82, 2009.

[RH01] Ravi Ramamoorthi and Path Hanrahan. An Efficient Representation for Irradiance Environment Maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 497–500, 2001.

[WBSS04] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. In *IEEE Transactions on Image Processing*, volume 13, pages 600–612, 2004.

List of Figures

3.1 Direct and indirect lighting 6
3.2 Color bleeding 7
3.3 Light absorption and scattering 8
3.4 Microfacet theory 9
3.5 BRDF direct and indirect lighitng 9
3.6 Irradiance maps 11
3.7 Path tracing 12
3.8 Light Propagation Volumes injection step. The figures were taken from the powerpoint accompanying [KD09]. 14
3.9 Reflective Shadow Map for a scene. The figures were created using the engine written for this research. 15
3.10 Intensity in the grid after the injection phase 16
3.11 Propagation after several iterations 16
3.12 Light Propagation Volumes gather operation 18
3.13 Visualization of the propagation steps for the LPV. The images were generated using the engine written for this research. 19
3.14 The result of using an LPV on a scene. For demonstration purposes the indirect lighting has been exaggerated. 19
3.15 Cascaded Light Propagation Volumes 21
3.16 Example of a cone 22
3.17 Voxelized result of a scene 23
3.18 Conservative rasterization 24
3.19 Calculation to determine the sampling radius for a cone 25
3.20 Calculations to determine the next sampling point in a cone . . . 25
3.21 Calculation to compute a nudge factor to prevent self intersection 26
3.22 Indirect diffuse lighting with VCT 28
3.23 Specular reflections using VCT 29
3.24 Emissive lighting using voxel cone tracing 31
3.25 Refractions through voxel cone tracing 31
3.26 Soft shadows using voxel cone tracing. The image was created using the engine written for this research. 32
3.27 SSAO Point sampling in a sphere. 33
3.28 SSAO Point sampling in a hemisphere. 33
3.29 Ambient occlusion using Screen Space Ambient Occlusion. 34
3.30 Indirect lighting using SSAO 35
3.31 Ambient occlusion using Voxel Cone Tracing. 36

4.1 Visual comparison for global illumination 37

5.1 Reference images for the basic scene 42
5.2 Reference images for the sponza scene 43

6.1 Results for the sponza scene 48

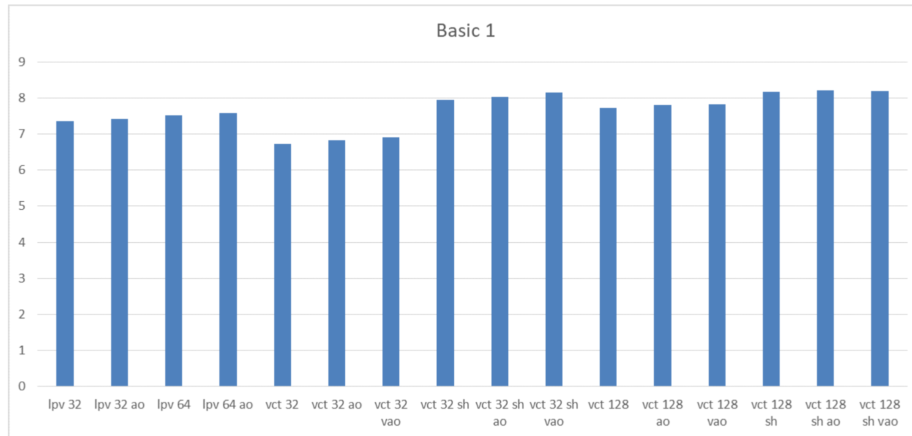
List of Tables

6.1 Error metric scores for the basic scene. 46
6.2 Error metric for the sponza scene 47

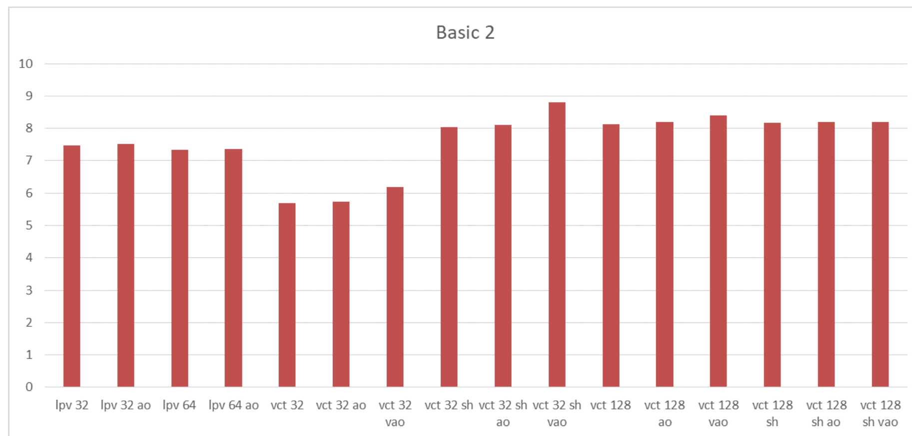
Appendices

A. Experimental Results

A.1 Basic Scene Results

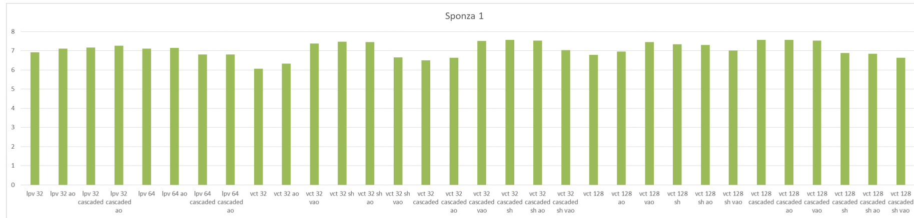


Results for the first configuration of the basic scene.

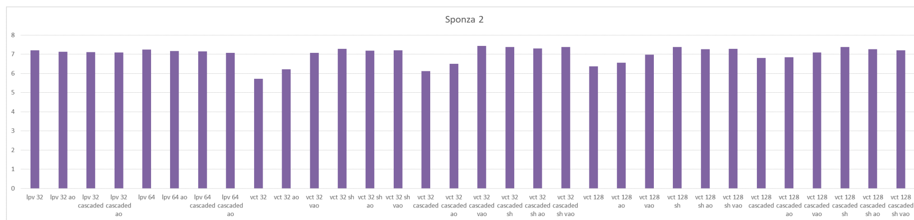


Results for the second configuration of the basic scene.

A.2 Sponza Scene Results



Results for the first configuration of the sponza scene.



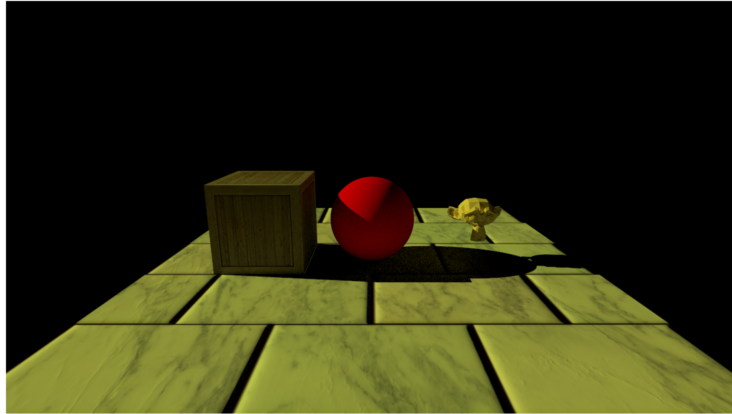
Results for the second configuration of the sponza scene.

A.3 SSIM Index

This chapter shows the results of all the tests, as described in section 5. For every test, the image generated with the algorithm, as well as the SSIM image from the index is shown. The SSIM image shows the SSIM index for each color component of each pixel of the image. An index of one means the color component of the image is completely equal to the component of the reference. A value of zero means it is completely different

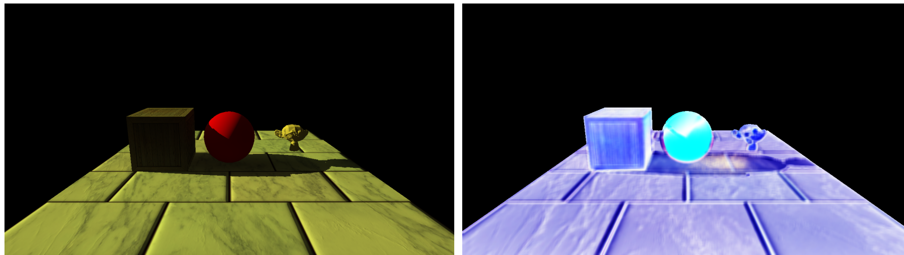
A.3.1 Basic scene: Configuration 1

Reference

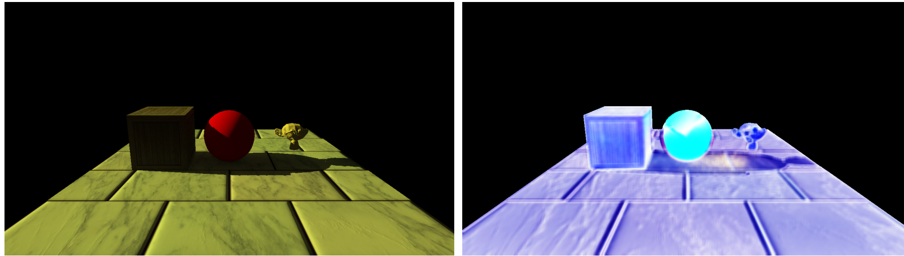


Reference image for first camera configuration for the basic scene.

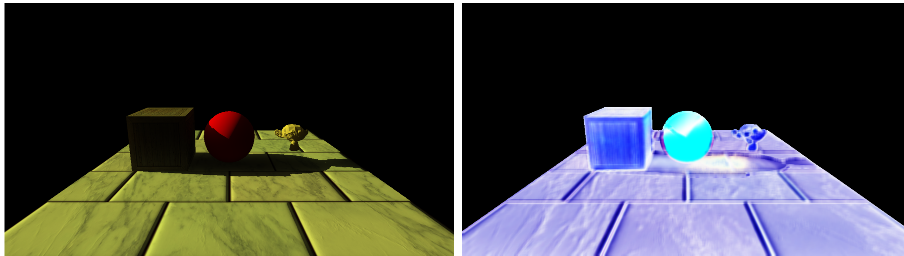
Results



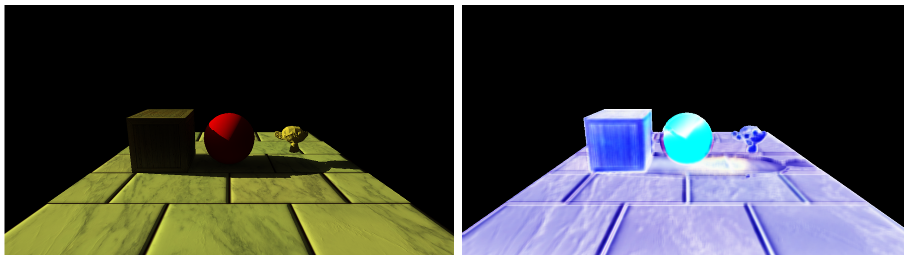
Basic scene. Camera configuration 1. Light Propagation Volumes, dimensions 32 x 32 x 32. Score: 7.35972



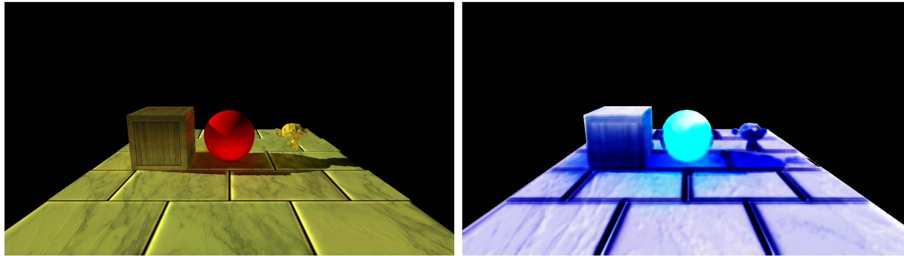
Basic scene. Camera configuration 1. Light Propagation Volumes, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.41518



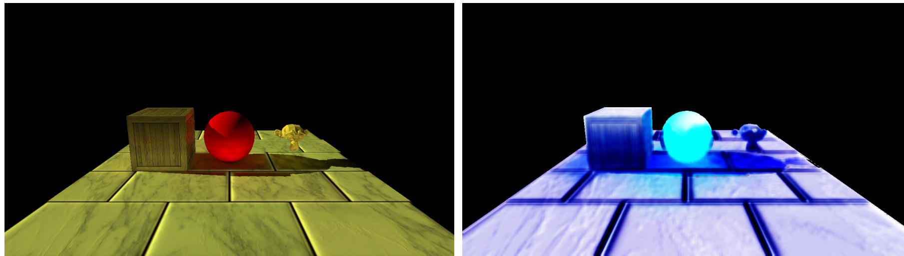
Basic scene. Camera configuration 1. Light Propagation Volumes, dimensions 64 x 64 x 64. Score: 7.52197



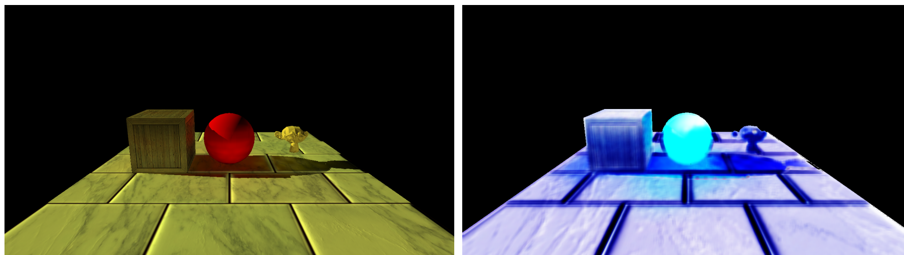
Basic scene. Camera configuration 1. Light Propagation Volumes, dimensions 64 x 64 x 64 with Ambient Occlusion. Score: 7.57575



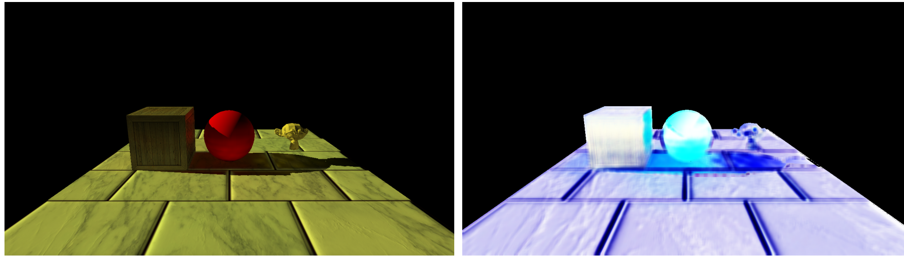
Basic scene. Camera configuration 1. Voxel Cone Tracing, dimensions 32 x 32 x 32. Score: 6.725



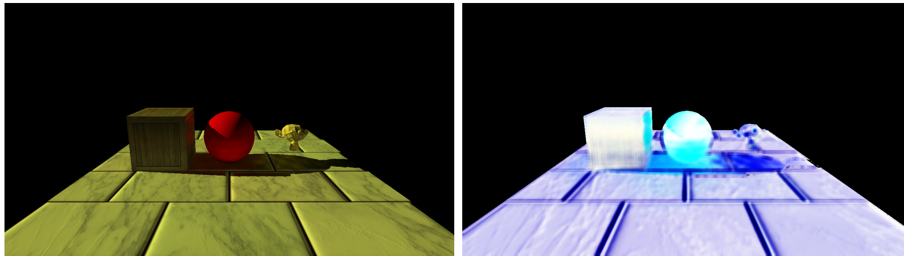
Basic scene. Camera configuration 1. Voxel Cone Tracing, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 6.83063



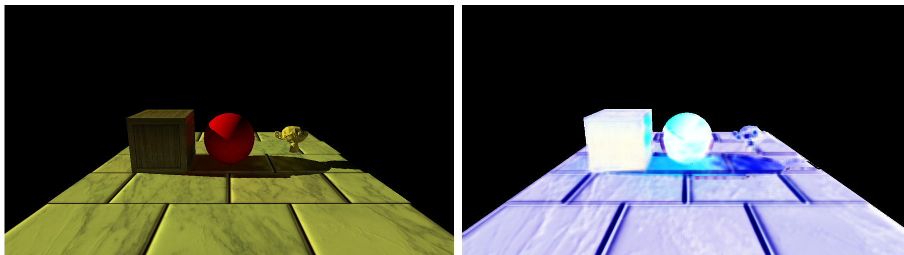
Basic scene. Camera configuration 1. Voxel Cone Tracing, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 6.90634



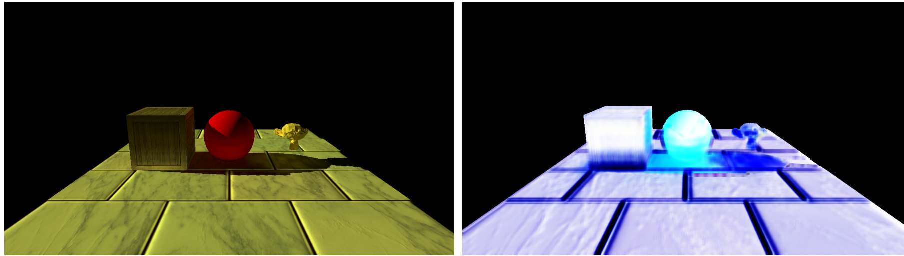
Basic scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32. Score: 7.94017



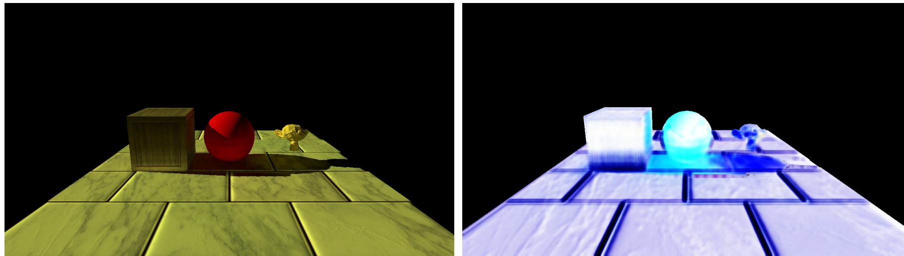
Basic scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 8.02049



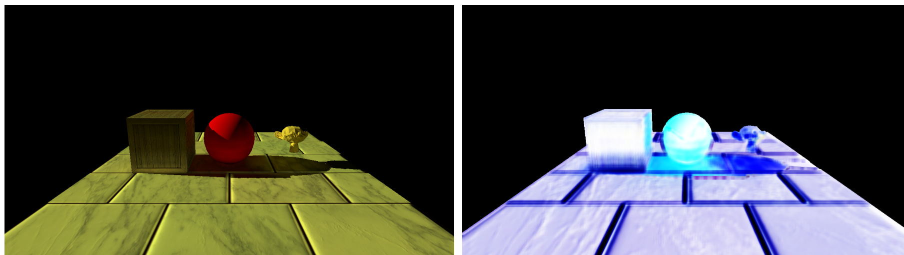
Basic scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 8.14196



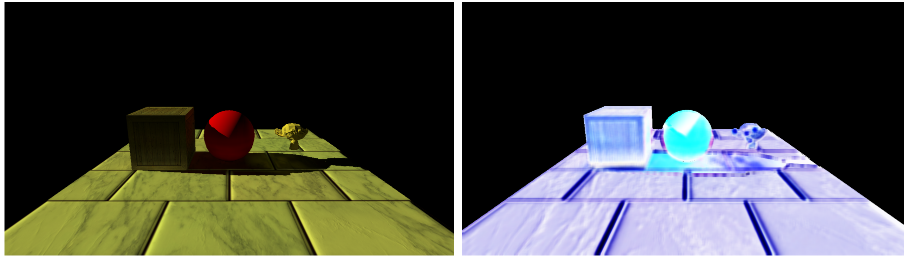
Basic scene. Camera configuration 1. Voxel Cone Tracing, dimensions 128 x 128 x 128. Score: 7.72177



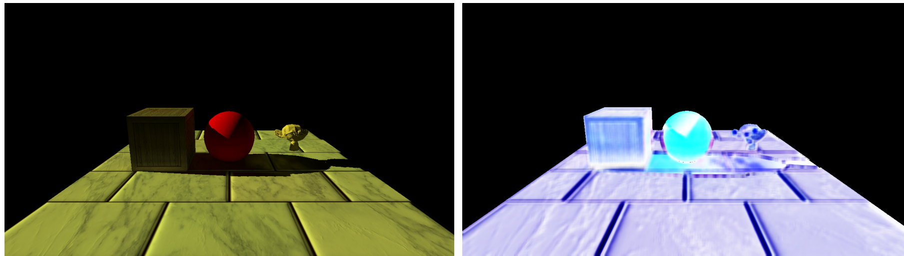
Basic scene. Camera configuration 1. Voxel Cone Tracing, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 7.80809



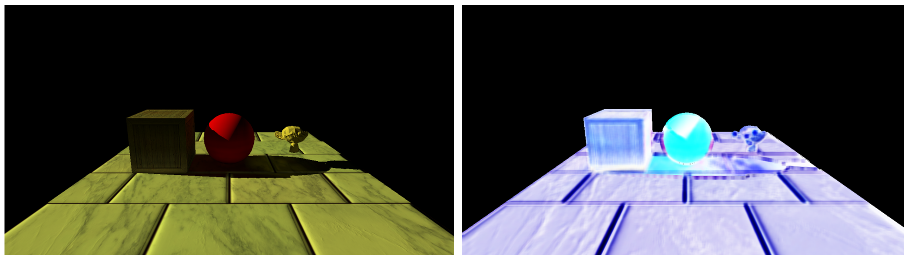
Basic scene. Camera configuration 1. Voxel Cone Tracing, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 7.82198



Basic scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128. Score: 8.1662



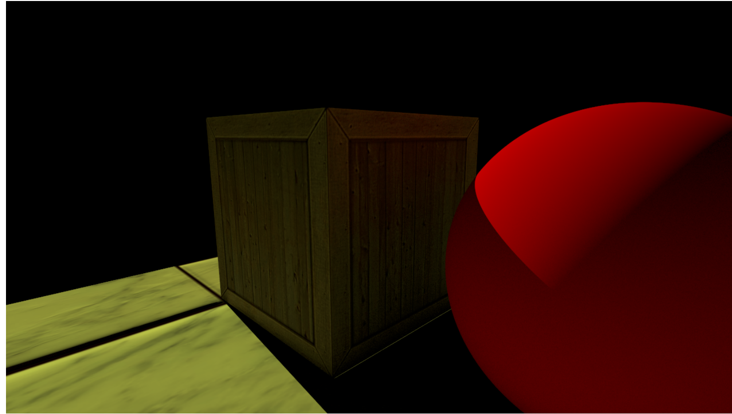
Basic scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 8.21534



Basic scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 8.19092

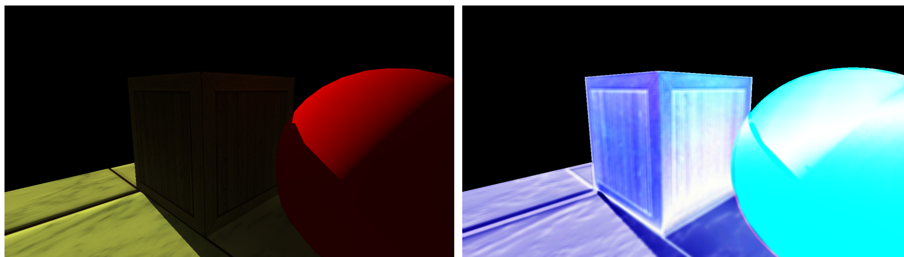
A.3.2 Basic scene: Configuration 2

Reference

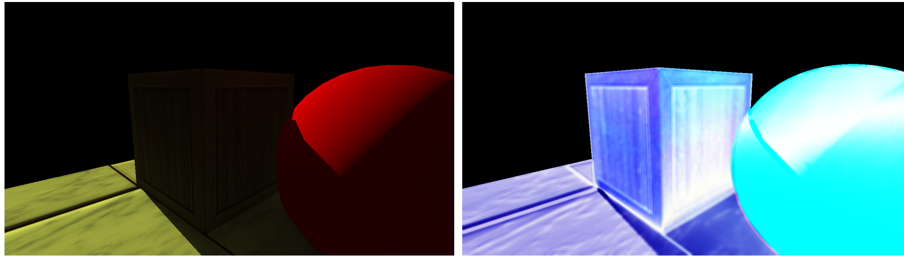


Reference image for second camera configuration for the basic scene.

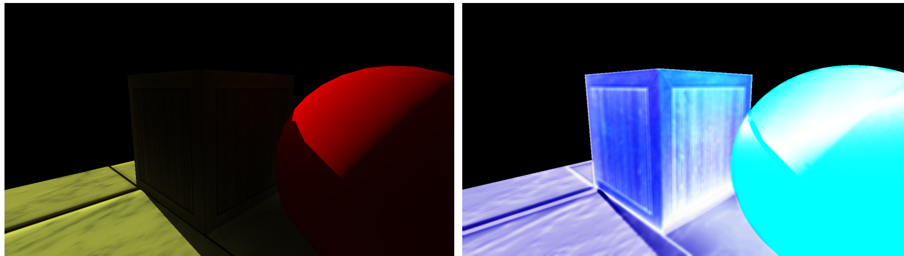
Results



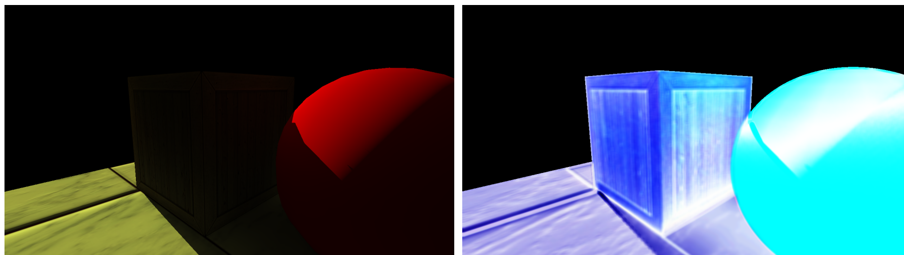
Basic scene. Camera configuration 2. Light Propagation Volumes, dimensions 32 x 32 x 32. Score: 7.46911



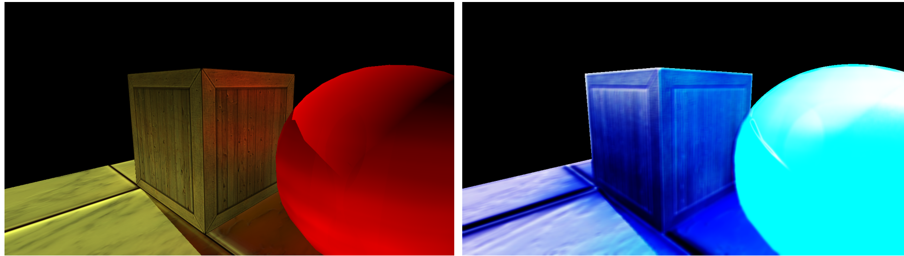
Basic scene. Camera configuration 2. Light Propagation Volumes, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.50752



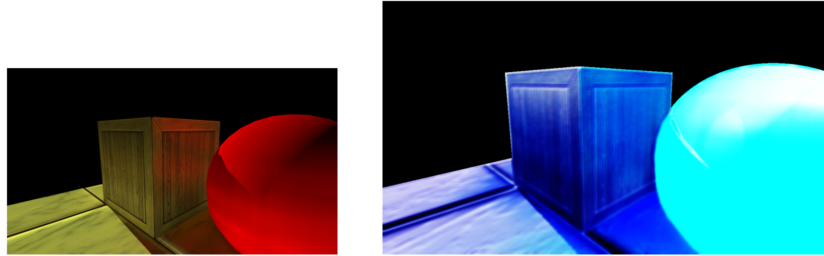
Basic scene. Camera configuration 2. Light Propagation Volumes, dimensions 64 x 64 x 64. Score: 7.34371



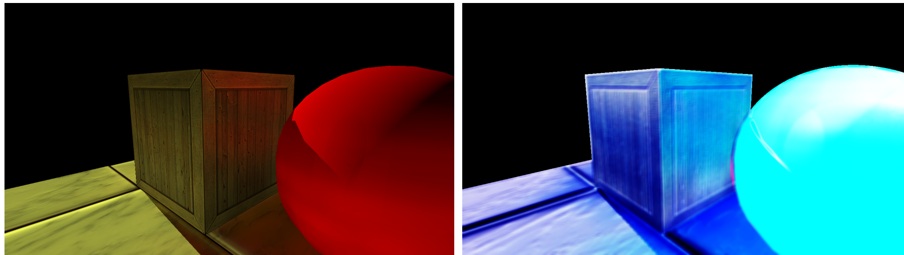
Basic scene. Camera configuration 2. Light Propagation Volumes, dimensions 64 x 64 x 64 with Ambient Occlusion. Score: 7.0



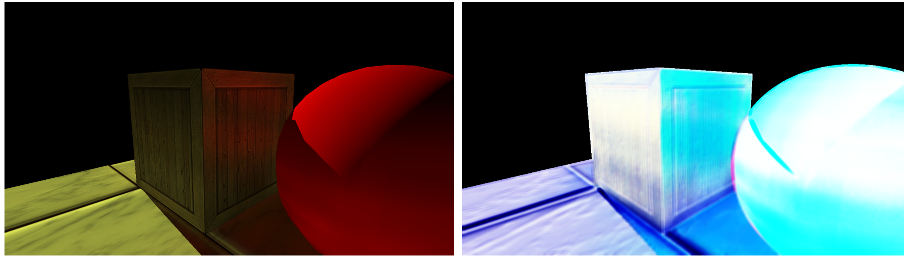
Basic scene. Camera configuration 2. Voxel Cone Tracing, dimensions 32 x 32 x 32. Score: 5.68233



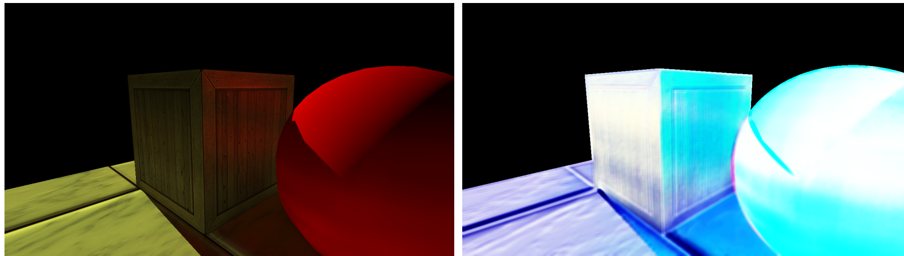
Basic scene. Camera configuration 2. Voxel Cone Tracing, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 5.73776



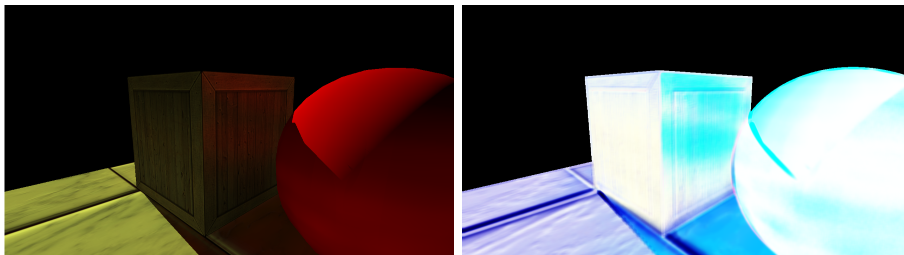
Basic scene. Camera configuration 2. Voxel Cone Tracing, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 6.17783



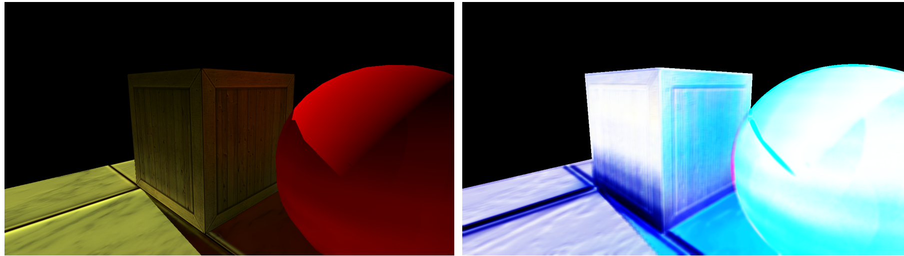
Basic scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32. Score: 8.04488



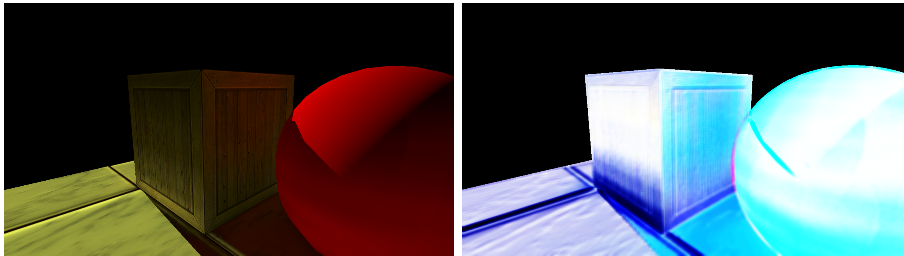
Basic scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 8.11504



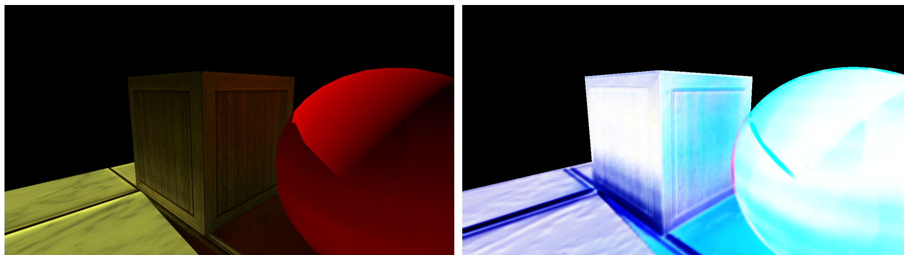
Basic scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 8.80377



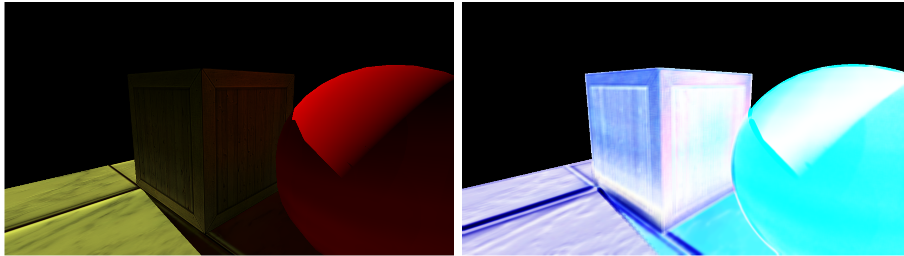
Basic scene. Camera configuration 2. Voxel Cone Tracing, dimensions 128 x 128 x 128. Score: 8.12149



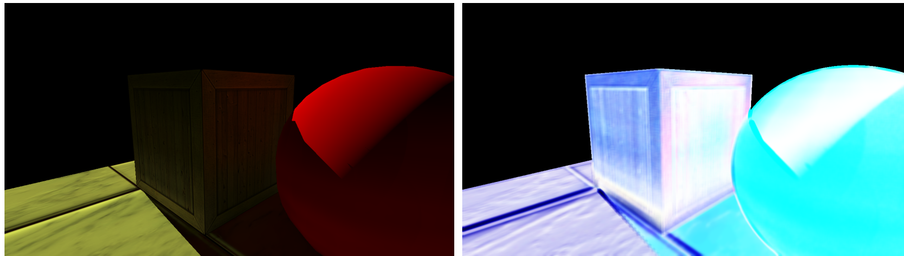
Basic scene. Camera configuration 2. Voxel Cone Tracing, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 8.19119



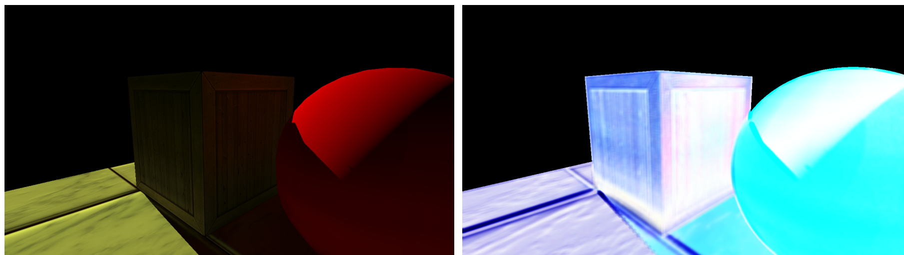
Basic scene. Camera configuration 2. Voxel Cone Tracing, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 8.40112



Basic scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128. Score: 8.16055



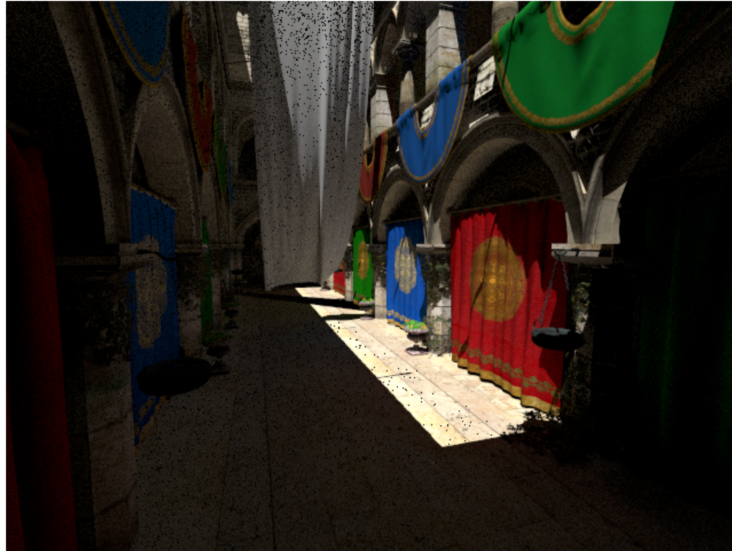
Basic scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 8.18867



Basic scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 8.18941

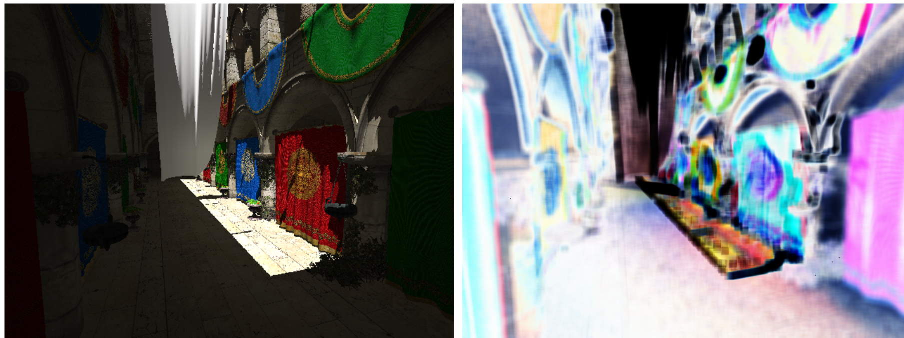
A.3.3 Sponza scene: Configuration 1

Reference

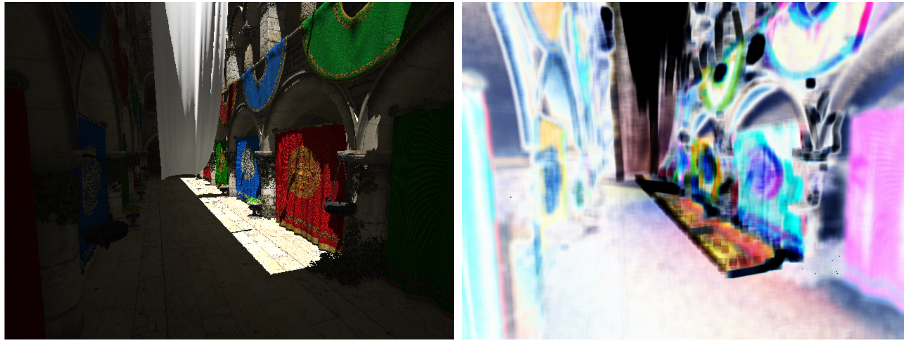


Reference image for first camera configuration for the sponza scene.

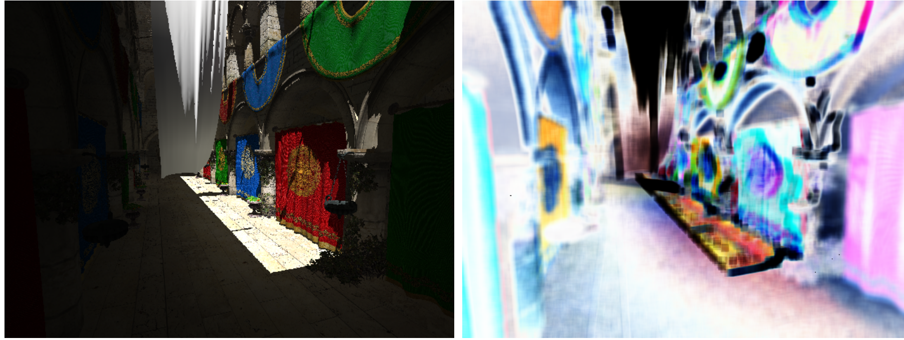
Results



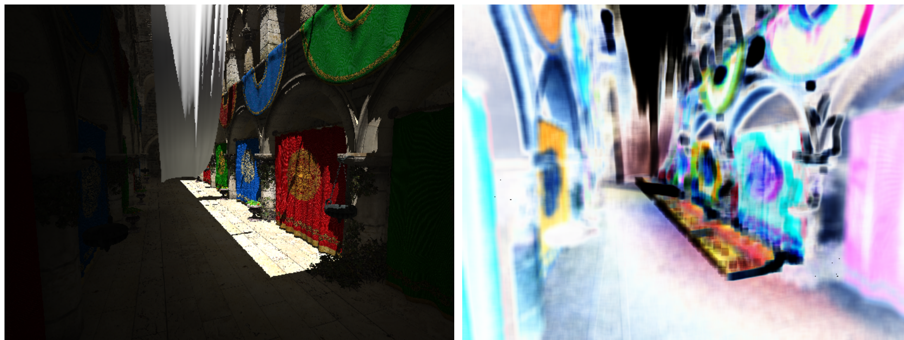
Sponza scene. Camera configuration 1. Light Propagation Volumes, dimensions 32 x 32 x 32. Score: 6.92271



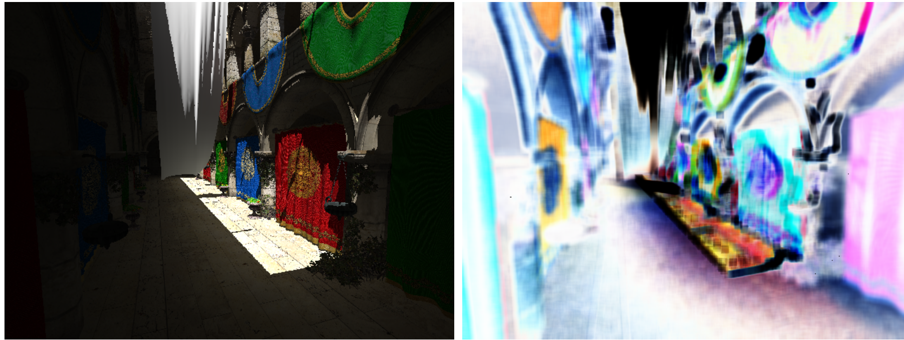
Sponza scene. Camera configuration 1. Light Propagation Volumes, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.1096



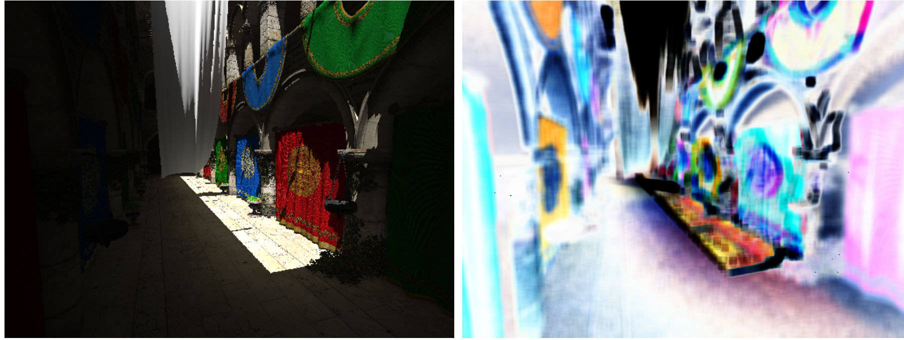
Sponza scene. Camera configuration 1. Cascaded Light Propagation Volumes, dimensions 32 x 32 x 32. Score: 7.15292



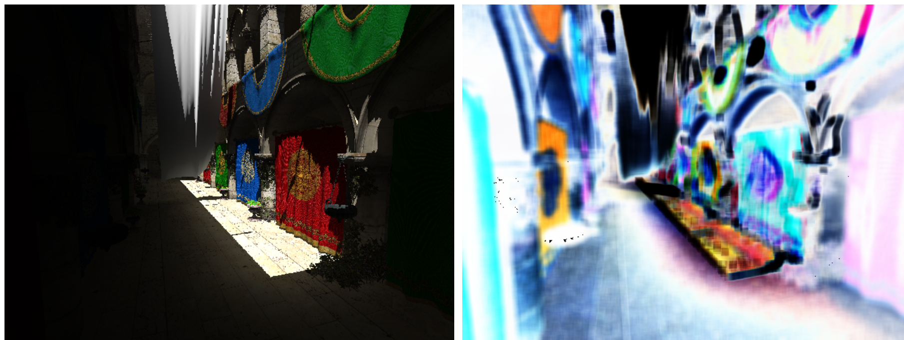
Sponza scene. Camera configuration 1. Cascaded Light Propagation Volumes, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.25342



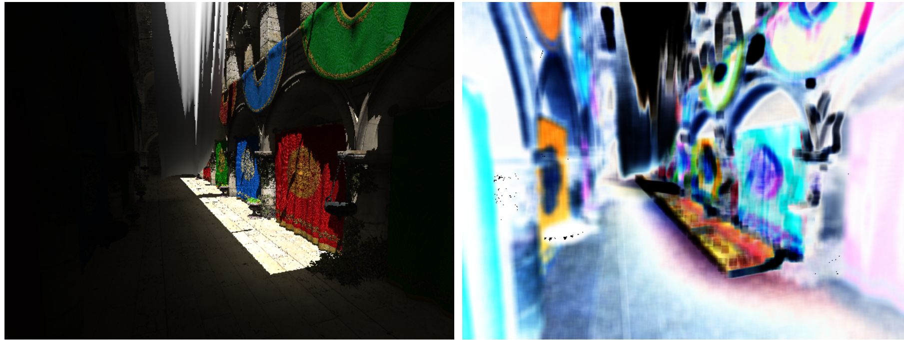
Sponza scene. Camera configuration 1. Light Propagation Volumes, dimensions 64 x 64 x 64. Score: 7.10905



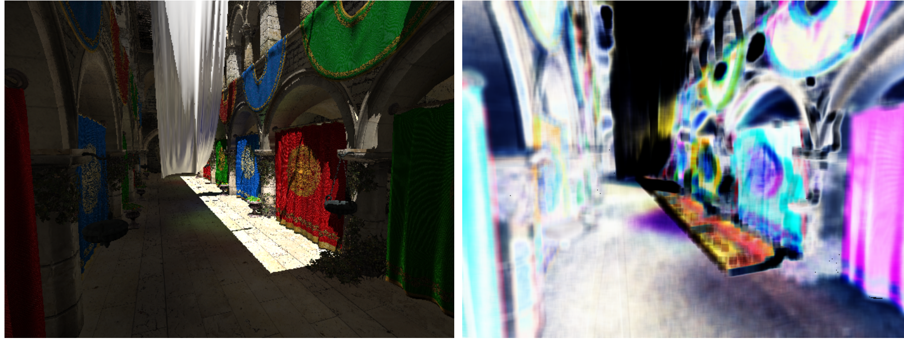
Sponza scene. Camera configuration 1. Light Propagation Volumes, dimensions 64 x 64 x 64 with Ambient Occlusion. Score: 7.13586



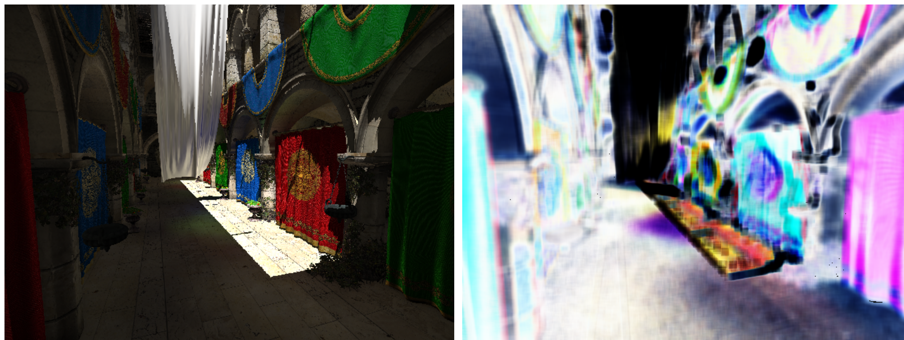
Sponza scene. Camera configuration 1. Cascaded Light Propagation Volumes, dimensions 64 x 64 x 64. Score: 6.80618



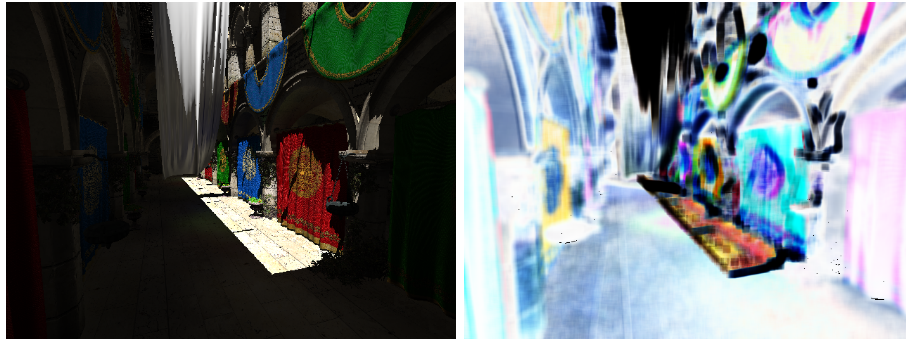
Sponza scene. Camera configuration 1. Cascaded Light Propagation Volumes, dimensions 64 x 64 x 64 with Ambient Occlusion. Score: 6.80623



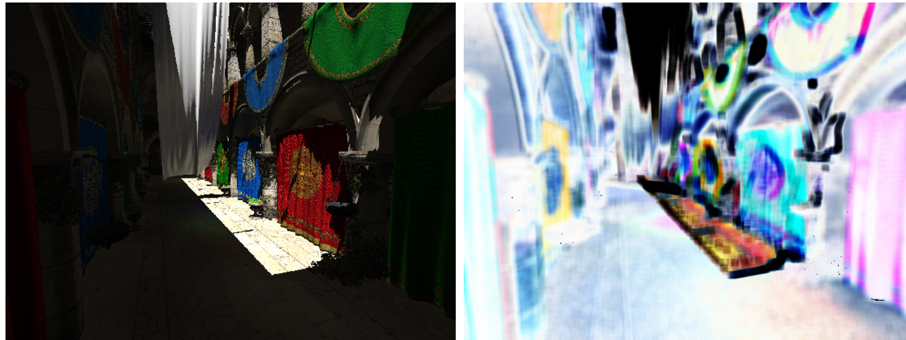
Sponza scene. Camera configuration 1. Voxel Cone Tracing, dimensions 32 x 32 x 32. Score: 6.04972



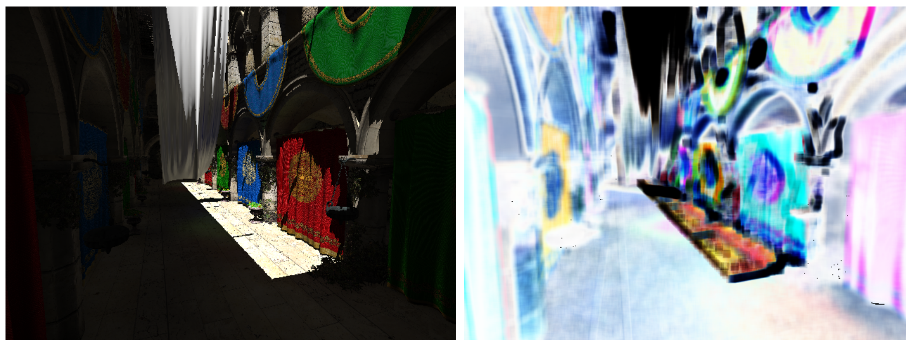
Sponza scene. Camera configuration 1. Voxel Cone Tracing, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 6.32192



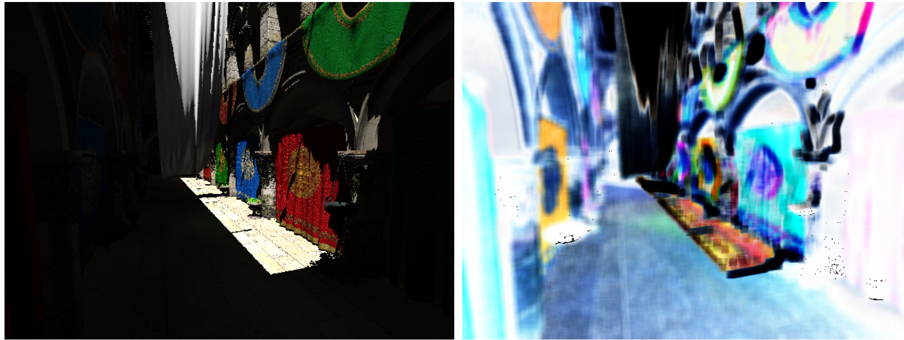
Sponza scene. Camera configuration 1. Voxel Cone Tracing, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 7.3811



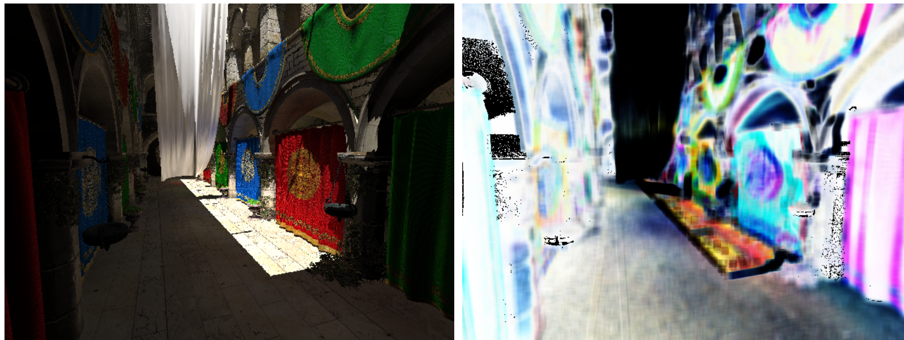
Sponza scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32. Score: 7.47038



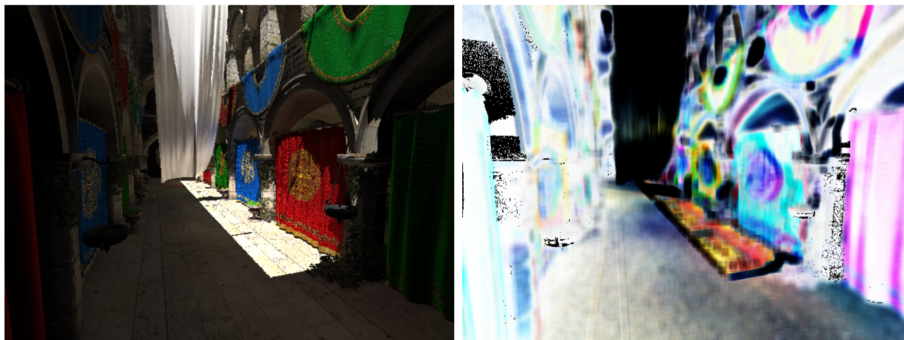
Sponza scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.45351



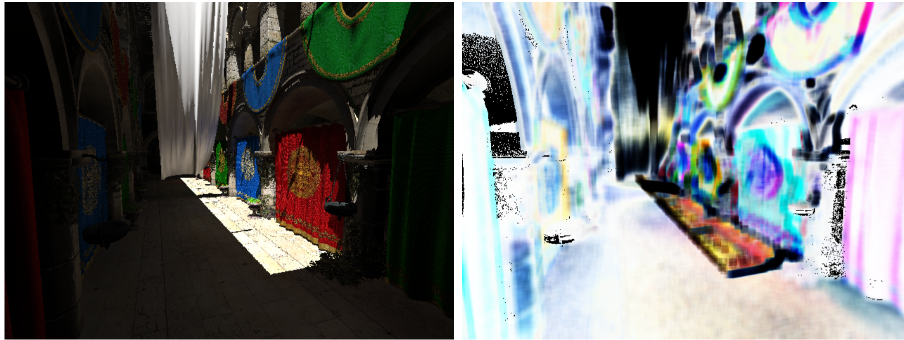
Sponza scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 6.65364



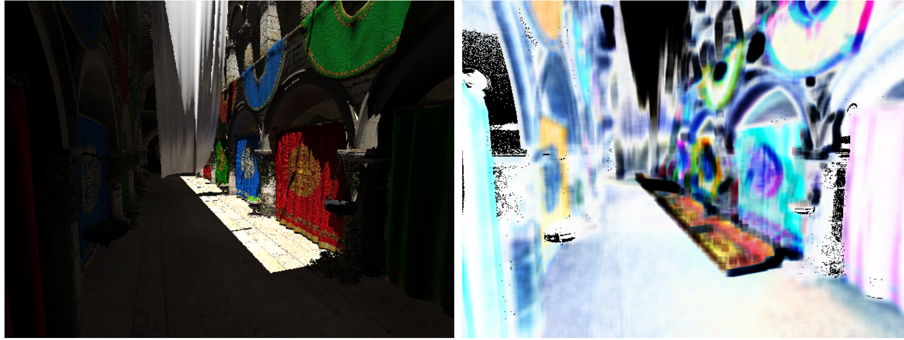
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32. Score: 6.50038



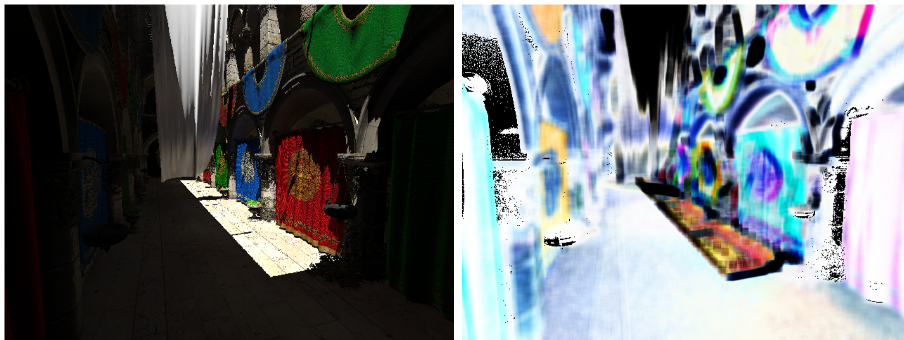
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 6.63723



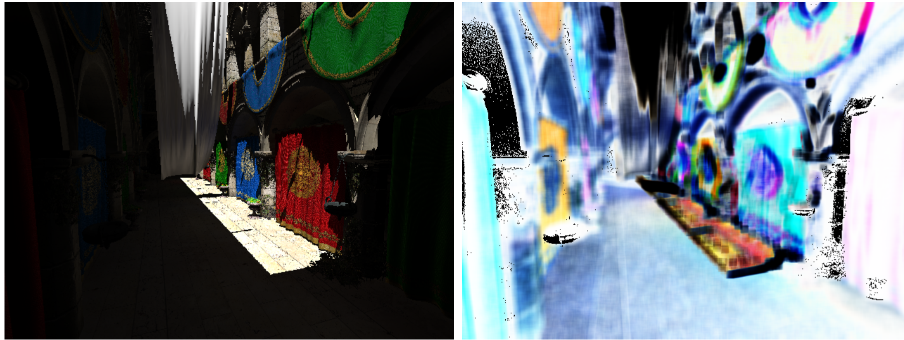
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 7.49734



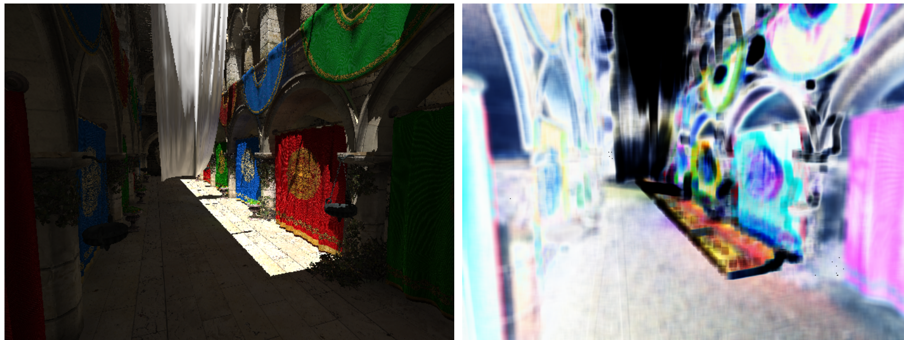
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32. Score: 7.56519



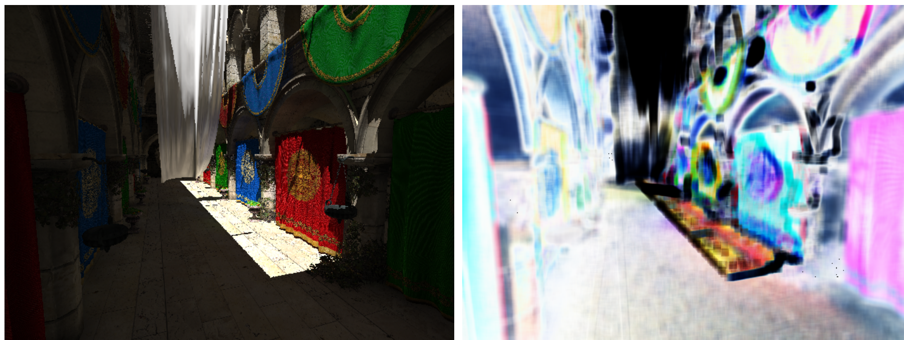
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.53097



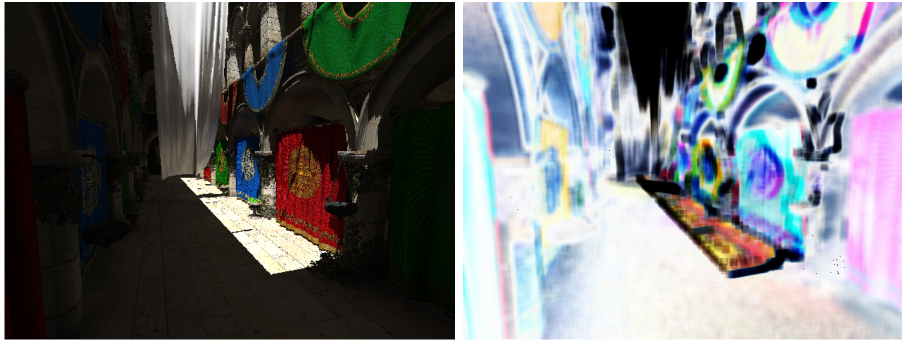
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 7.03117



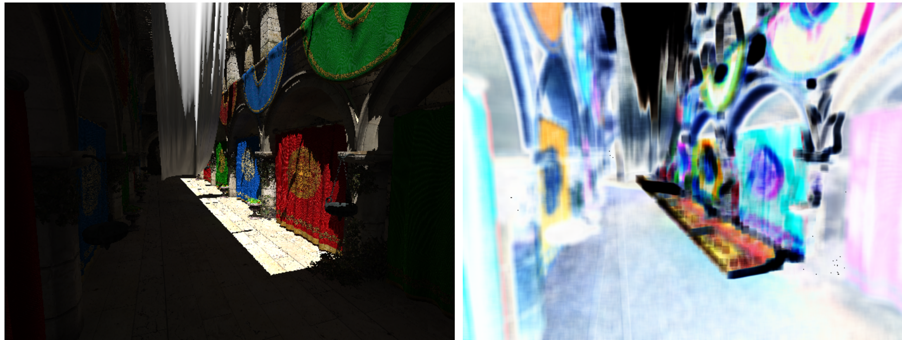
Sponza scene. Camera configuration 1. Voxel Cone Tracing, dimensions 128 x 128 x 128. Score: 6.7757



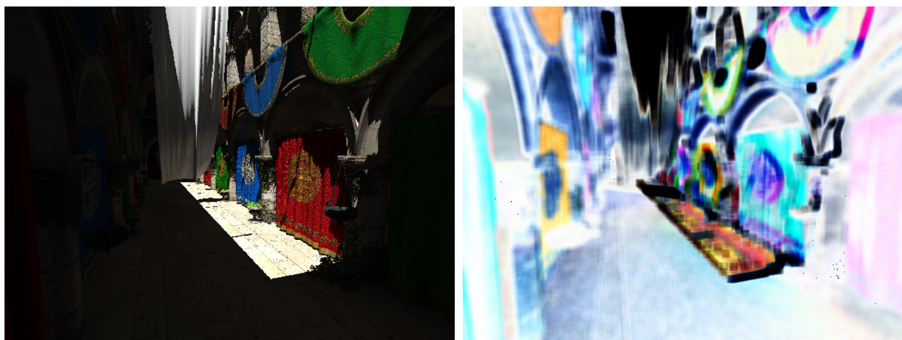
Sponza scene. Camera configuration 1. Voxel Cone Tracing, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 6.94764



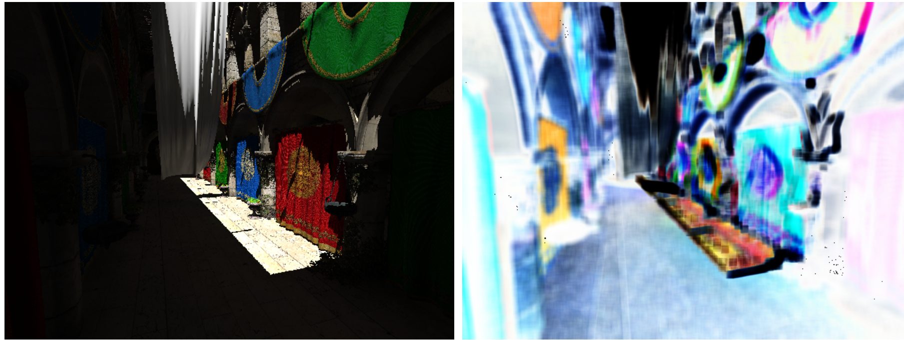
Sponza scene. Camera configuration 1. Voxel Cone Tracing, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 7.44524



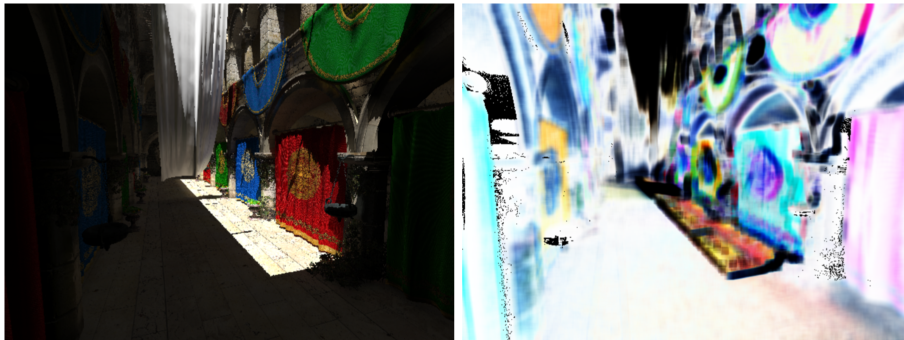
Sponza scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128. Score: 7.33576



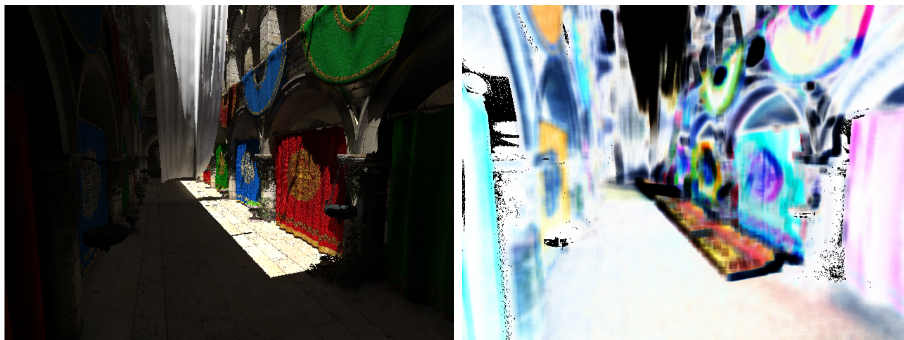
Sponza scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 7.28739



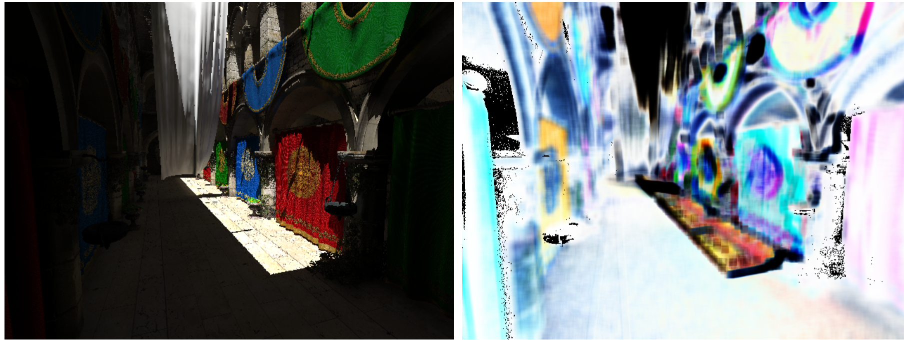
Sponza scene. Camera configuration 1. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 7.01389



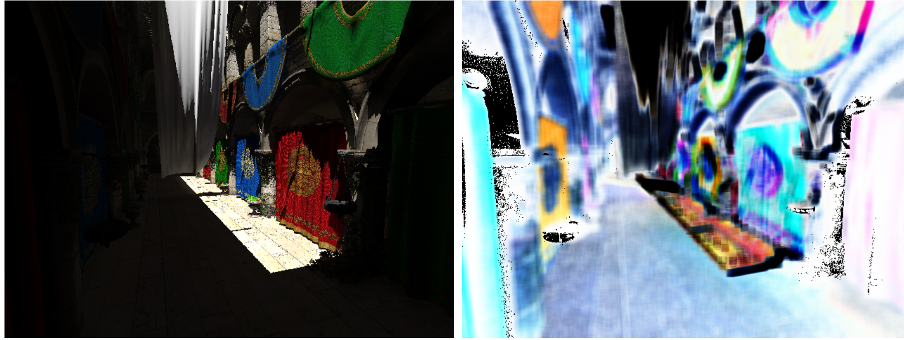
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128. Score: 7.5559



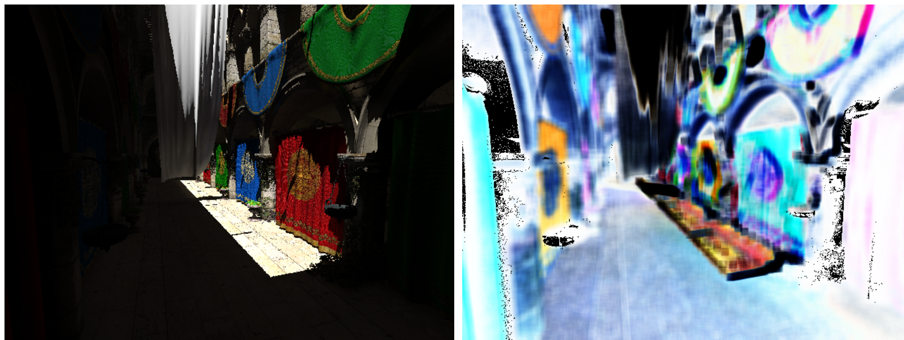
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 7.56632



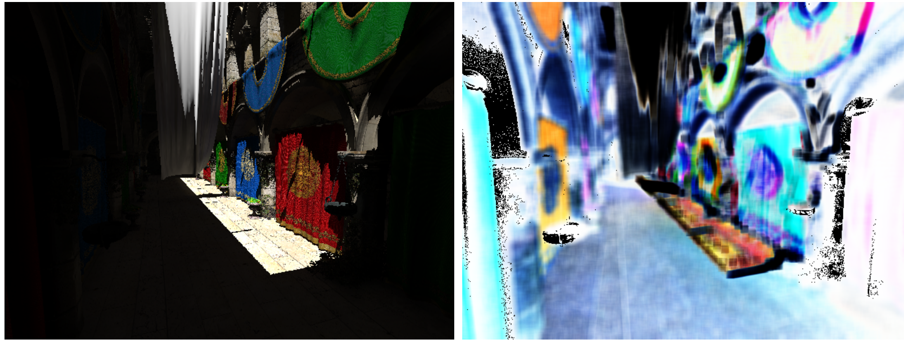
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 7.52922



Sponza scene. Camera configuration 1. Cacaded Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128. Score: 6.88213



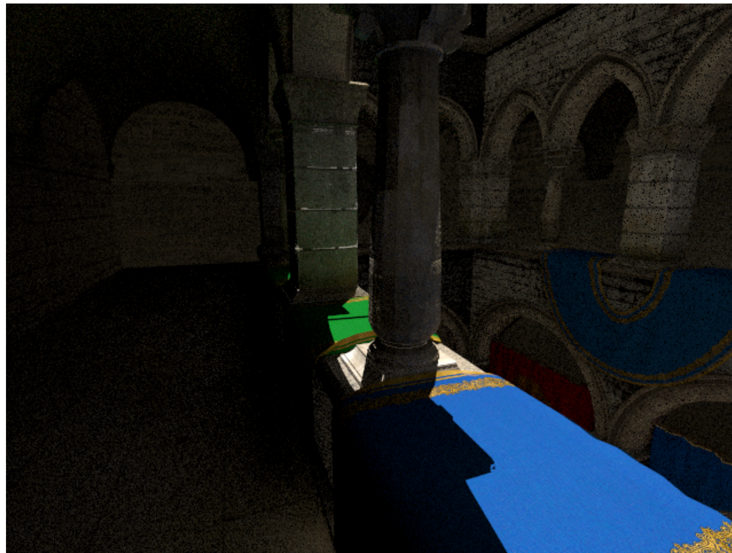
Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 6.83252



Sponza scene. Camera configuration 1. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 6.62664

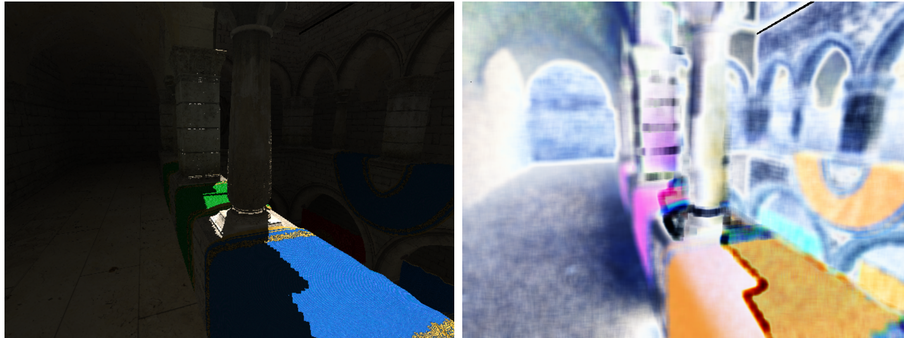
A.3.4 Sponza scene: Configuration 2

Reference

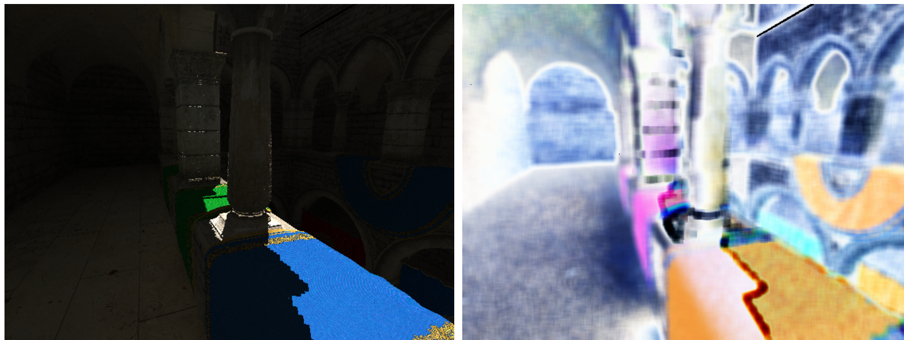


Reference image for second camera configuration for the sponza scene.

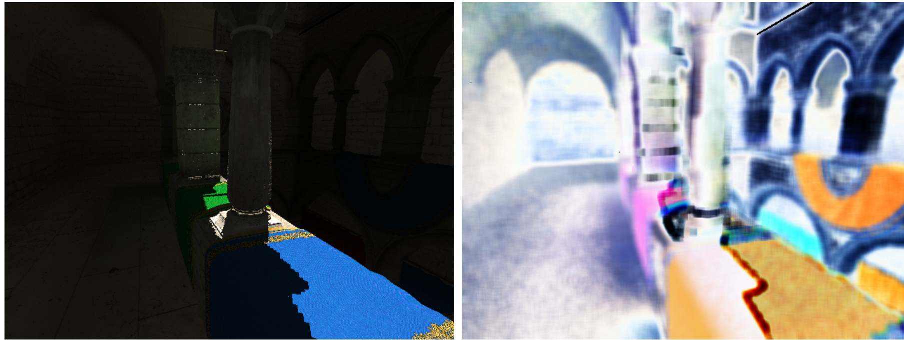
Results



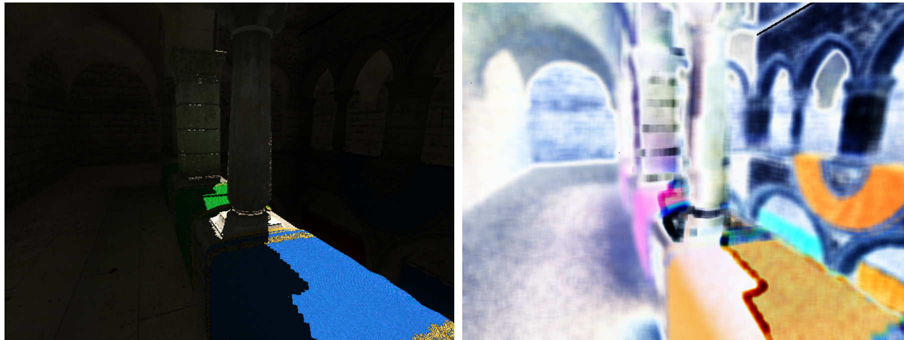
Sponza scene. Camera configuration 2. Light Propagation Volumes, dimensions 32 x 32 x 32. Score: 7.20196



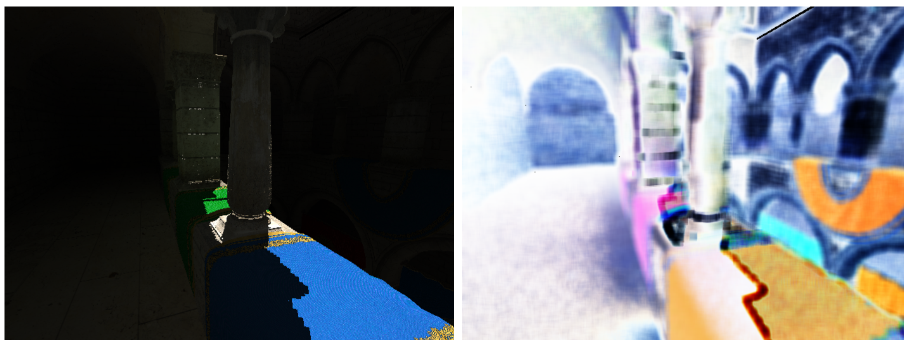
Sponza scene. Camera configuration 2. Light Propagation Volumes, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.12059



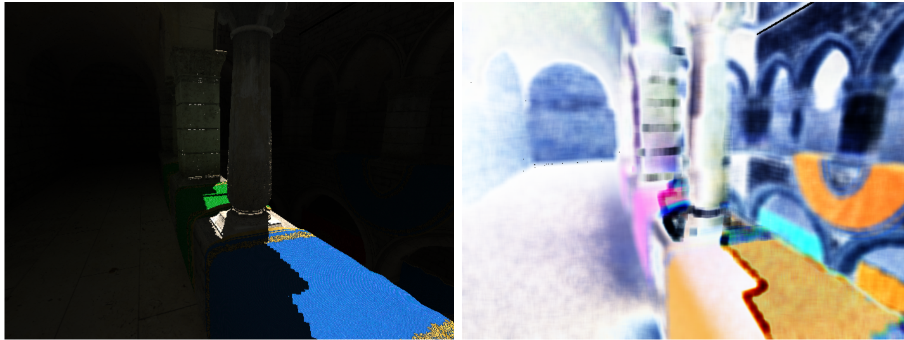
Sponza scene. Camera configuration 2. Cascaded Light Propagation Volumes, dimensions 32 x 32 x 32. Score: 7.10172



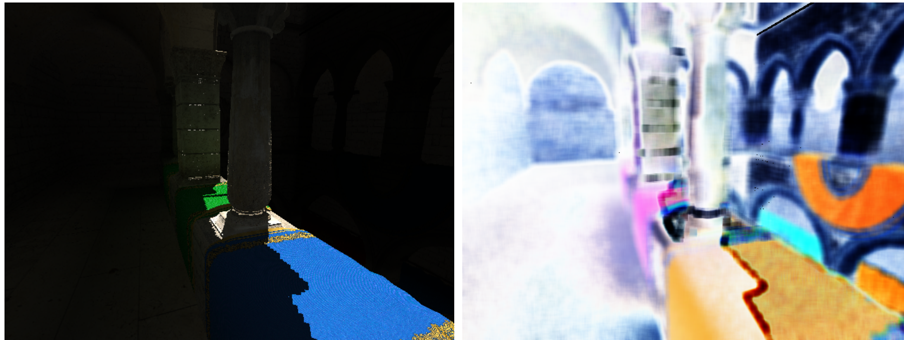
Sponza scene. Camera configuration 2. Cascaded Light Propagation Volumes, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.08763



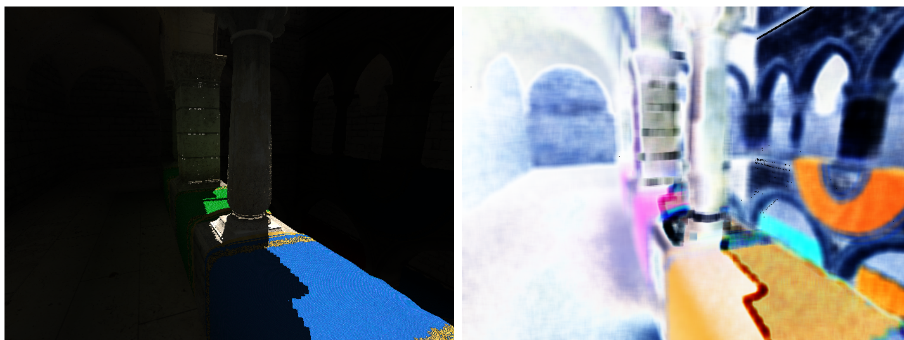
Sponza scene. Camera configuration 2. Light Propagation Volumes, dimensions 64 x 64 x 64. Score: 7.23175



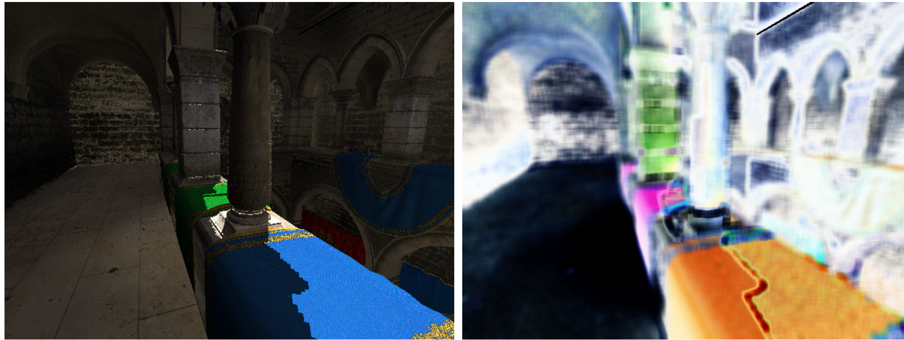
Sponza scene. Camera configuration 2. Light Propagation Volumes, dimensions 64 x 64 x 64 with Ambient Occlusion. Score: 7.16263



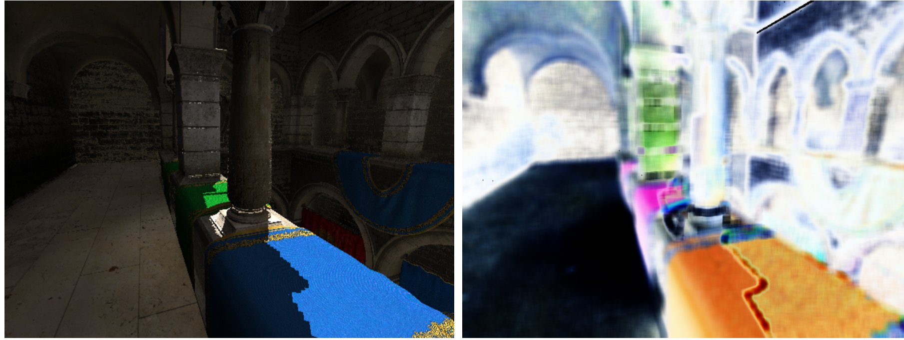
Sponza scene. Camera configuration 2. Cascaded Light Propagation Volumes, dimensions 64 x 64 x 64. Score: 7.14802



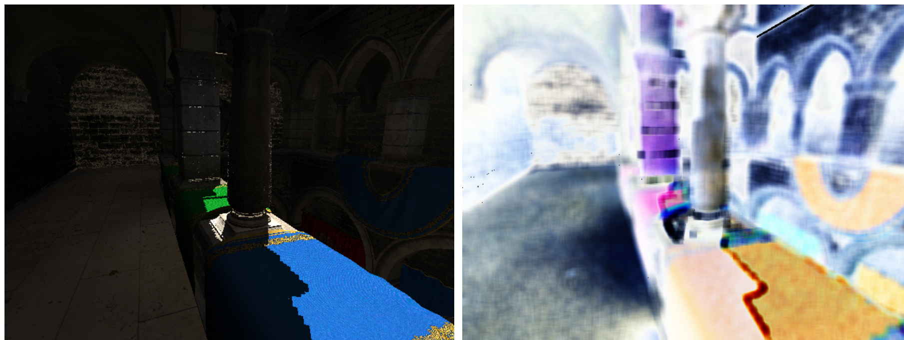
Sponza scene. Camera configuration 2. Cascaded Light Propagation Volumes, dimensions 64 x 64 x 64 with Ambient Occlusion. Score: 7.07308



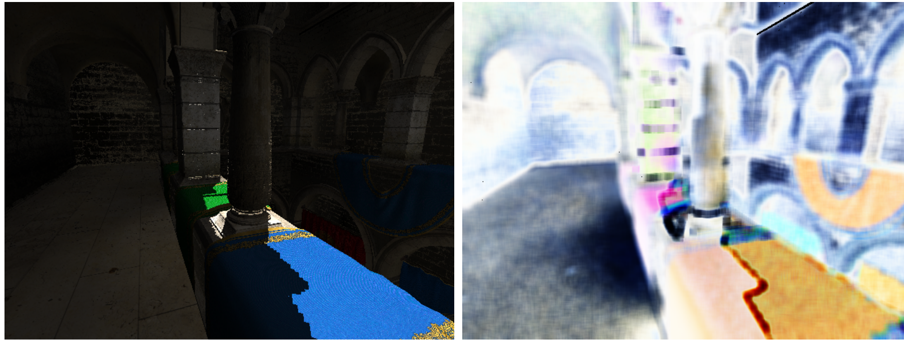
Sponza scene. Camera configuration 2. Voxel Cone Tracing, dimensions 32 x 32 x 32. Score: 5.72353



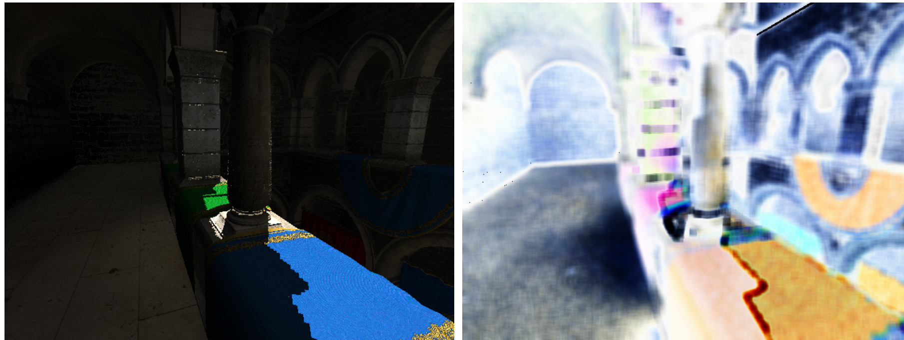
Sponza scene. Camera configuration 2. Voxel Cone Tracing, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 6.2168



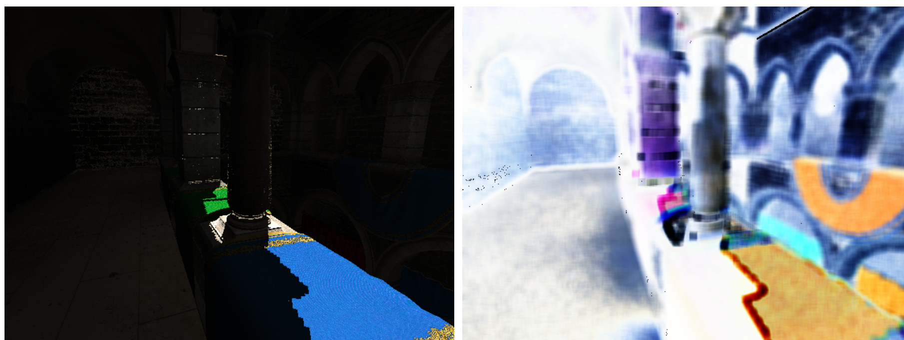
Sponza scene. Camera configuration 2. Voxel Cone Tracing, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 7.07407



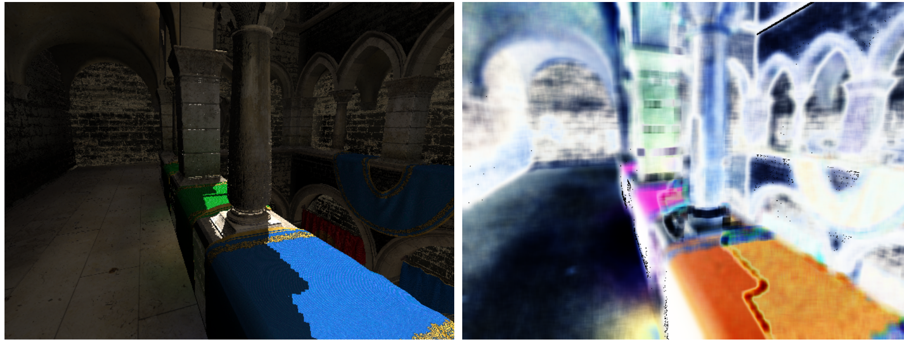
Sponza scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32. Score: 7.27794



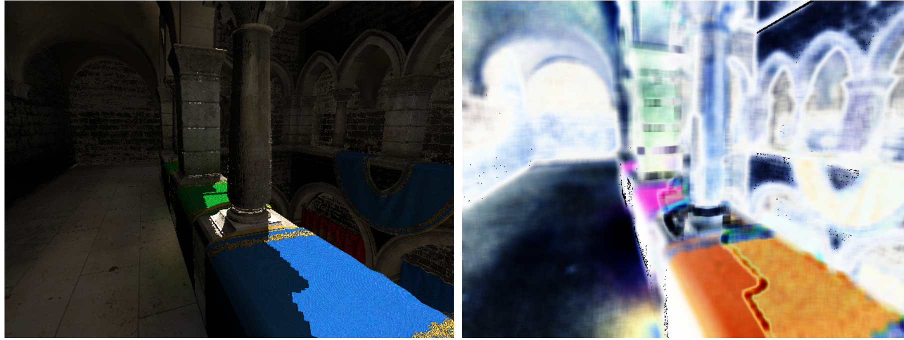
Sponza scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.18962



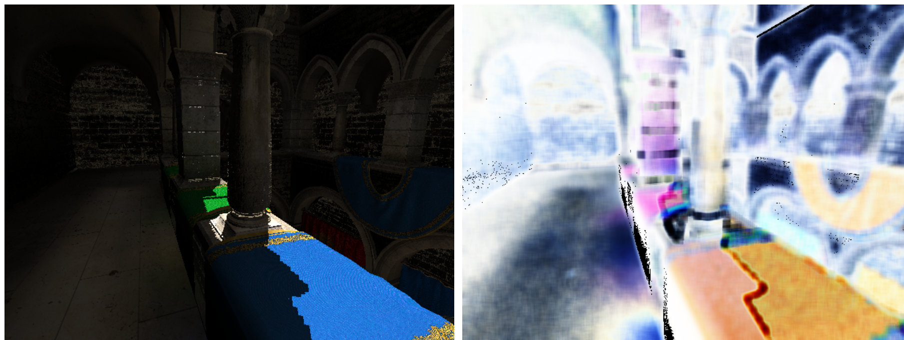
Sponza scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 7.20493



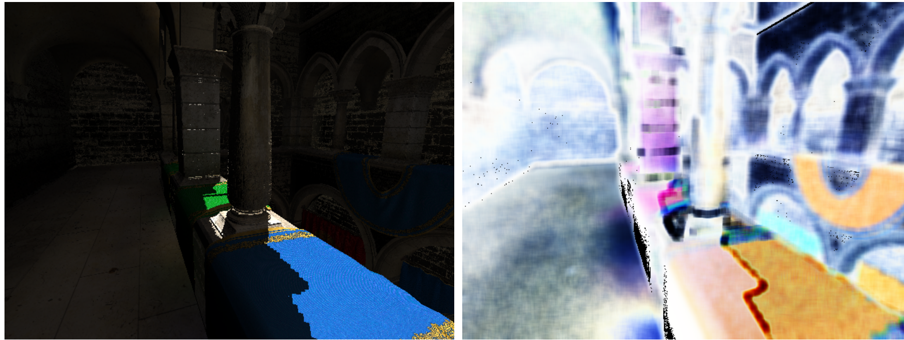
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32. Score: 6.11409



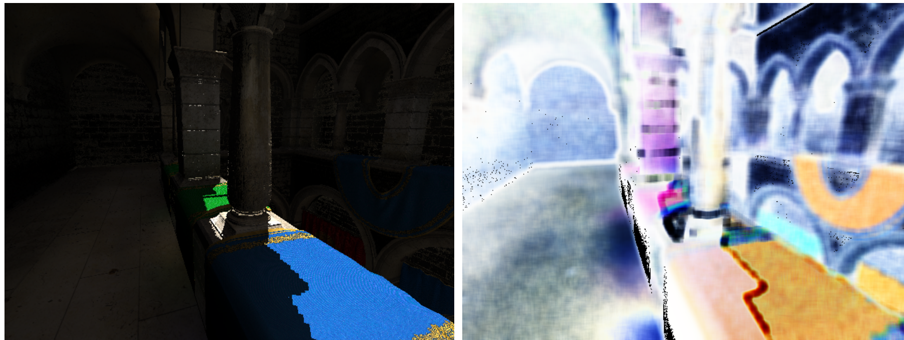
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 6.49928



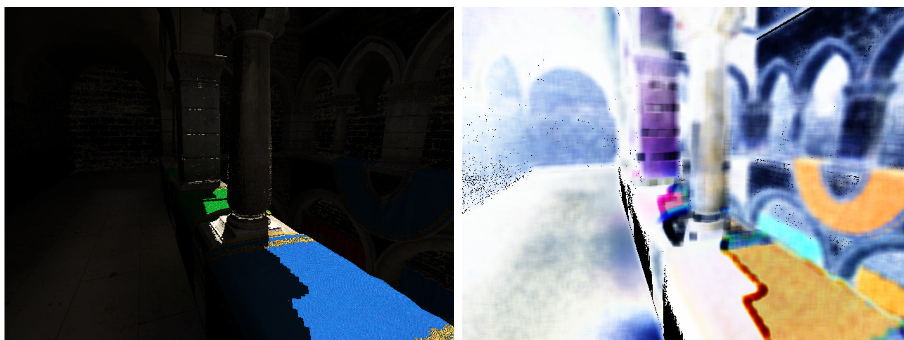
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 7.43055



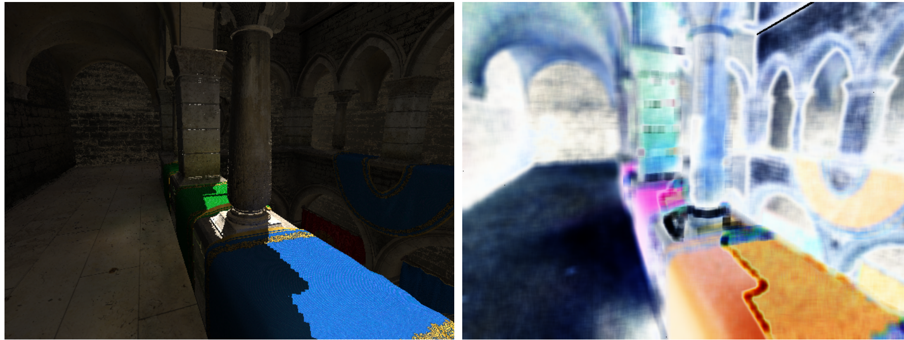
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32. Score: 7.37627



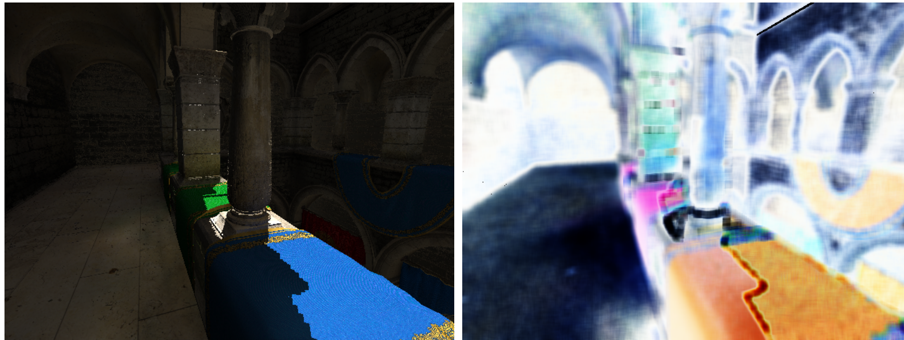
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Ambient Occlusion. Score: 7.2963



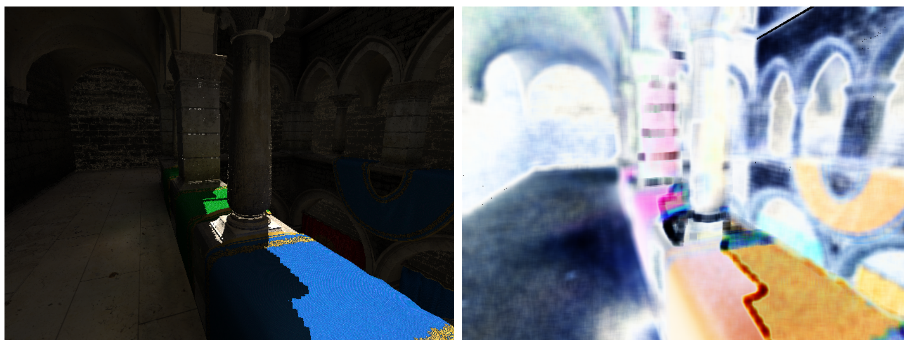
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 32 x 32 x 32 with Voxel Ambient Occlusion. Score: 7.37576



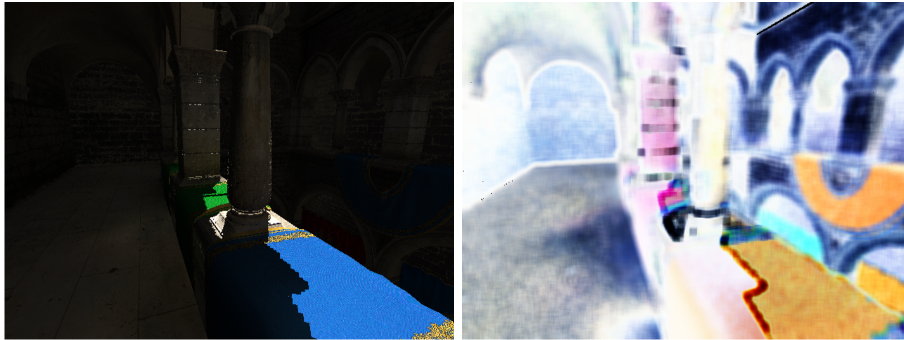
Sponza scene. Camera configuration 2. Voxel Cone Tracing, dimensions 128 x 128 x 128. Score: 6.35485



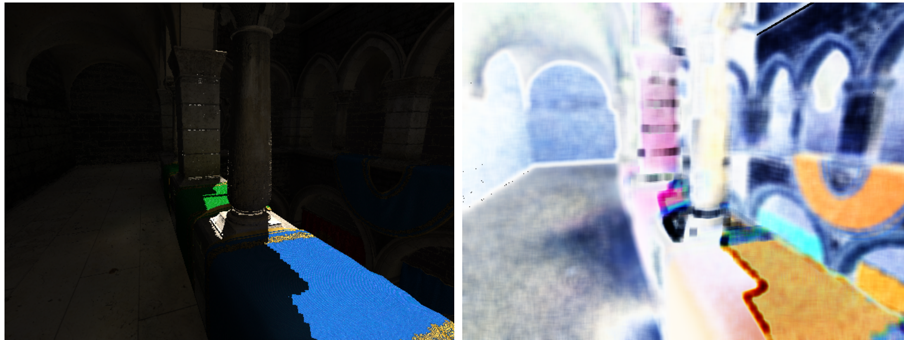
Sponza scene. Camera configuration 2. Voxel Cone Tracing, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 6.55574



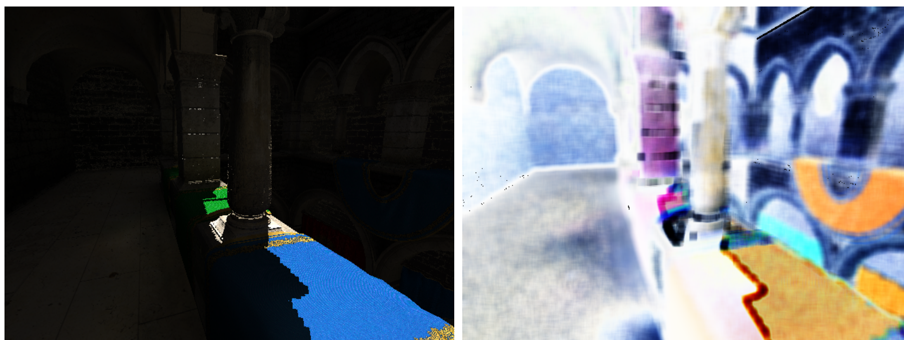
Sponza scene. Camera configuration 2. Voxel Cone Tracing, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 6.9726



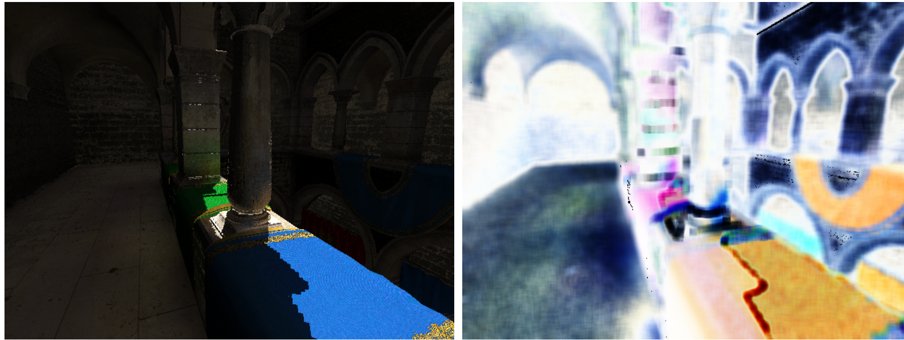
Sponza scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128. Score: 7.38036



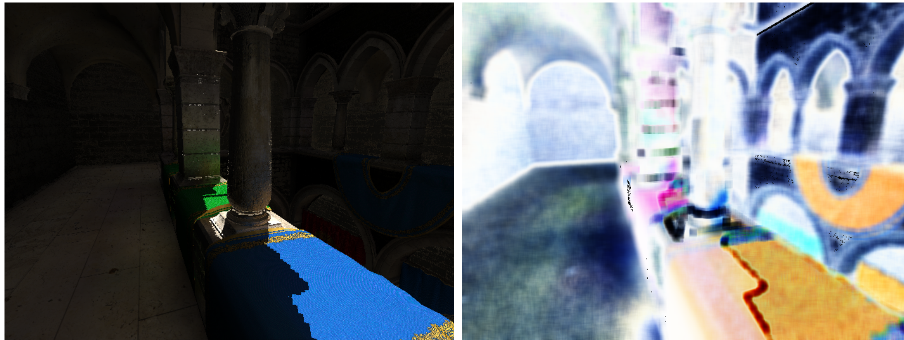
Sponza scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 7.26363



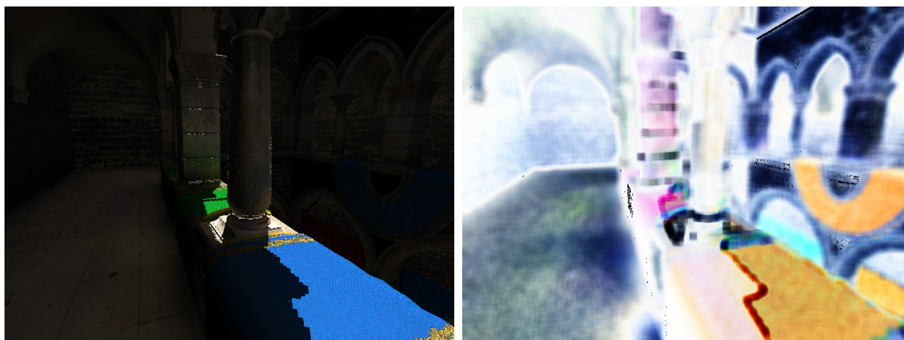
Sponza scene. Camera configuration 2. Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 7.26962



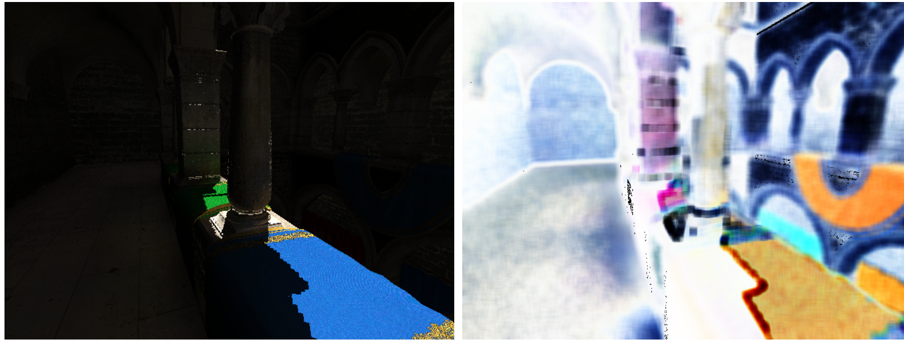
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128. Score: 6.7976



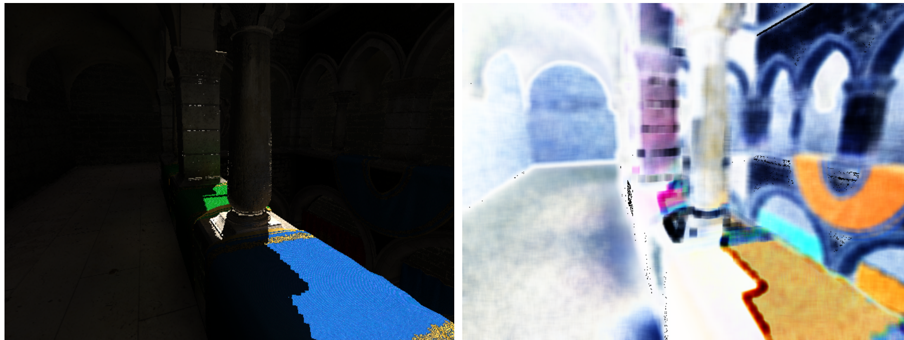
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 6.83765



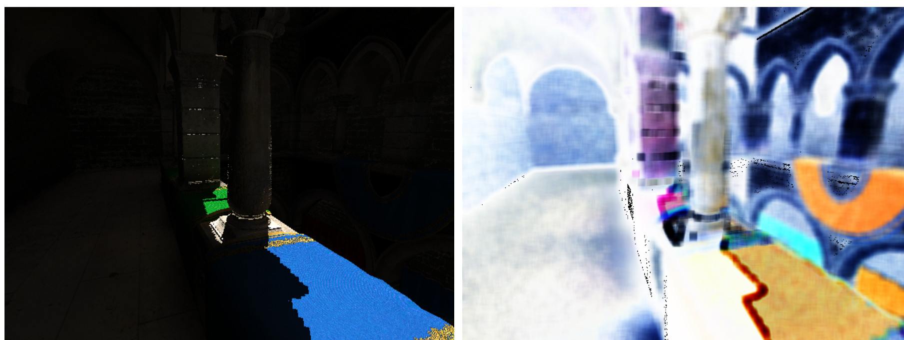
Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 7.08996



Sponza scene. Camera configuration 2. Cacaded Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128. Score: 7.36901



Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Ambient Occlusion. Score: 7.2539



Sponza scene. Camera configuration 2. Cascaded Voxel Cone Tracing with Spherical Harmonics, dimensions 128 x 128 x 128 with Voxel Ambient Occlusion. Score: 7.208

B. Shader Code

This chapter details shader code for several issues discussed during this document.

B.1 Spherical harmonics

Listing B.1: Compute the spherical harmonic from a direction and an intensity

```
#define SH_cosLobe_C0 0.886226925f      // sqrt(pi)/2
#define SH_cosLobe_C1 1.02332671f       // sqrt(pi/3)
vec4 dirToCosineLobe(in vec3 N){
   return vec4(
      SH_cosLobe_C0,
      -SH_cosLobe_C1 * N.y,
      SH_cosLobe_C1 * N.z,
     -SH_cosLobe_C1 * N.x);
}

void computeSHIntensity(vec3 N, vec3 flux){
   // convert normal to SH space
   vec4 coeffs = (dirToCosineLobe(N));

   // compute SH for each color channel
   vec4 redSH = coeffs * flux.r;
   vec4 greenSH = coeffs * flux.g;
   vec4 blueSH = coeffs * flux.b;
}
```

Listing B.2: Compute the amount of lighting in a given direction

```
#define SH_C0 0.282094792f      // 1 / 2sqrt(pi)
#define SH_C1 0.488602512f      // sqrt(3/pi) / 2

vec4 dirToSH(in vec3 N){
   return vec4(
      SH_C0,
      -SH_C1 * N.y,
      SH_C1 * N.z,
      -SH_C1 * N.x);
}
```

```
vec3 getIndirectLighting(vec3 P, vec3 N){
    // P is the index in the grid, range [0...1]
    // N is the world normal

    // convert normal to SH intensity
    vec4 sh = dirToSH(-N);

    // get SH coefficients for each color channel
    vec4 redSH = texture(redTexture, P);
    vec4 greenSH = texture(greenTexture, P);
    vec4 blueSH = texture(blueTexture, P);

    // mix based on SH
    return vec3(
        dot(sh, redSH),
        dot(sh, greenSH),
        dot(sh, blueSH));
}
```

B.2 Light Propagation Volume

Listing B.3: LPV propagation

```
struct Propagation{
    vec4 R;
    vec4 G;
    vec4 B;
}

float computeBlocking(ivec3 index, vec3 mainDir){
    // from range [0..resolution] to [0..1]
    vec3 geomCoord = (index + 0.5 * mainDir) / resolution;
    vec4 geomSH = texture(geomTexture, geomCoord);

    return 1.0 - saturate(dot(geomSH, dirToSH(-mainDir)));
}

const float solidAngleToCell = 0.12753712;
const float solidAngleToFace = 0.13478556;

Propagation propagate(ivec3 cellIndex){
    // cell index is the index in the grid of the cell we
        ↪ want to gather the intensity to, in the range
        ↪ [0...resolution]
    Propagation prop;
    for (int i = 0; i < 6; i++){
```

```
        // get direction to neighbour cell
        vec3 mainDir = dirToCell(i);
        // determine index into neighbour cell
        ivec3 index = cellIndex - mainDir;

        vec4 redSH = texture(redTexture, index);
        vec4 greenSH = texture(greenTexture, index);
        vec4 blueSH = texture(blueTexture, index);

        float occlusion = computeBlocking(index, mainDir);
        float w = occlusion * solidAngleToCell;

        vec4 dirCos = dirToCosineLobe(mainDir);
        vec4 dirSH = dirToSH(mainDir);

        prop.R += w * dot(redSH, dirSH) * dirCos;
        prop.G += w * dot(greenSH, dirSH) * dirCos;
        prop.B += w * dot(blueSH, dirSH) * dirCos;

        // compute contribution for each face
        for (int face = 0; face < 4; face++){
            // get direction to this face of the neighbour
                ↪ cell
            vec3 dirToFace = dirToSideFace(face, i);

            occlusion = computeBlocking(index, dirToFace);
            float w = occlusion * solidAngleToFace;

            dirCos = dirToCosineLobe(dirToFace);
            dirSH = dirToSH(dirToFace);

            prop.R += w * dot(redSH, dirSH) * dirCos;
            prop.G += w * dot(greenSH, dirSH) * dirCos;
            prop.B += w * dot(blueSH, dirSH) * dirCos;
        }
    }
    return prop;
}
```

B.3 Voxelization

Listing B.4: Conservative voxelization

```
void voxelize(vec3 p0, vec3 p1, vec3 p2){
    // p0, p1 and p2 are the vectors of the triangle in
        ↪ the range [-grid resolution...grid resolution]
```

```
// compute the normal of the face
vec3 n = cross(p1 - p0, p2 - p0);
float nDotX = abs(n.x);
float nDotY = abs(n.y);
float nDotZ = abs(n.z);

// find the axis that maximizes the projected area of
    ↪ this triangle
if (nDotX > nDotY && nDotX > nDotZ){
    // X axis
    p0.xyz = p0.yzx;
    p1.xyz = p1.yzx;
    p2.xyz = p2.yzx;
}
else if (nDotY < nDotX && nDotY > nDotZ){
    // Y axis
    p0.xyz = p0.xzy;
    p1.xyz = p1.xzy;
    p2.xyz = p1.xzy;
}

// next, enlarge the triangle for conservative
    ↪ rasterization

// calculate aabb for this triangle
vec4 AABB;
AABB.xy = p0.xy;
AABB.zw = p0.zw;

AABB.xy = min(p1.xy, AABB.xy);
AABB.zw = max(p1.xy, AABB.zw);

AABBxy = min(p2.xy, AABB.xy);
AABB.zw = max(p2.xy, AABB.zw);

// enlarge by half a pixel
vec2 hPixel = vec2(1.0 / gridResolution);
AABB.xy -= hPixel;
AABB.zw += hPixel;

// find 3 triangle edge planes
vec2 n0 = normalize(p1.xy - p0.xy);
vec2 n1 = normalize(p2.xy - p1.xy);
vec2 n2 = normalize(p0.xy - p2.xy);
```

```
    n0 = vec2(-n0.y, n0.x);
    n1 = vec2(-n1.y, n1.x);
    n2 = vec2(-n2.y, n2.x);

    // if triangle is back facing, flip its edge normals
        ↪ so triangle does not shrink
    vec3 clipN = cross(p1 - p0, p2 - p0);
    if (clipN.z < 0){
        e0 *= -1;
        e1 *= -1;
        e2 *= -2;
    }

    // pixel is 1x1, diagonal of a pixel is sqrt(2)
    // scaled by the size of the volume
    float pixelDiagonal = sqrt(2) / gridResolution;
    p0.xy -= pixelDiagonal * ((e2.xy / dot(e2.xy, n0.xy))
        ↪ + (e0.xy / dot(e0.xy, n2.xy)));
    p1.xy -= pixelDiagonal * ((e0.xy / dot(e0.xy, n1.xy))
        ↪ + (e1.xy / dot(e1.xy, n0.xy)));
    p2.xy -= pixelDiagonal * ((e1.xy / dot(e1.xy, n2.xy))
        ↪ + (e2.xy / dot(e2.xy, n1.xy)));

    // output parameters
    gl_Position = vec4(p0, 1);
    EmitVertex();
    gl_Position = vec4(p1, 1);
    EmitVertex();
    gl_Position = vec4(p2, 1);
    EmitVertex();
}
```

B.4 Image Atomic Average

Listing B.5: Computing an atomic average using the ImageAtomicCompSwap method of OpenGL 4.2

```
vec4 convRGBA8ToVec4( uint val){
    return vec4(float((val&0x000000FF)), float((val&0
        ↪ x0000FF00)>>8U), float((val&0x00FF0000)>>16U),
        ↪ float((val&0xFF000000)>>24U));
}
uint convVec4ToRGBA8( vec4 val){
    return ( uint(val.w)&0x000000FF)<<24U | (uint(val.z)&0
        ↪ x000000FF)<<16U | ( uint(val.y)&0x000000FF)<< 8U
        ↪  | (uint(val.x)&0x000000FF);
```

```
}
void imageAtomicRGBA8Avg(layout(r32ui) uimage3D voxelGrid
    ↪ , in ivec3 coords, in vec4 val) {
    // val is the color we want to store, in the range
        ↪ [0...1] with val.a being 1
    // val must be in this range, in order to prevent
        ↪ overflow
    // coords is in the range [0...resolution]
    val.rgb *= 255.0f;
    uint newVal = convVec4ToRGBA8(val);
    uint prevStoredVal = 0;
    uint curStoredVal = 0;
    // Loop as long as destination value gets changed by
        ↪ other threads
    while ( (curStoredVal = imageAtomicCompSwap(voxelGrid,
        ↪  coords, prevStoredVal, newVal)) !=
        ↪ prevStoredVal) {
        prevStoredVal = curStoredVal;
        vec4 rval = convRGBA8ToVec4( curStoredVal);
        rval.xyz =(rval.xyz* rval.w); // Denormalize
        vec4 curValF = rval + val; // Add new value
        curValF.xyz /=( curValF.w); // Renormalize
        newVal = convVec4ToRGBA8( curValF );
    }
}
```