FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Department of Simulation and Graphics

Realtime Global Illumination using Dynamic Radiance Volumes

Master Thesis

Author:

Andreas Reich

Examiner and Supervisor:

Prof. Thorsten Grosch

2nd Examiner:

Dr. Christian Rössl

Supervisor:

Johannes Jendersie

Magdeburg, 26.08.2015

Contents

Acknowledgements

I would like to especially thank my supervisor and good friend Johannes whose support was far beyond what I would expect from any supervisor. Special thanks to Anke who cheered me up when I got stuck badly, discussed contents even if she had other things to do and reviewed the thesis. I look forward to return you the favor!
Further, I would like to thank John for his extensive proofreading. Thanks to Prof. Grosch who made this thesis possible and laid out the basic direction. As well to Dr. Rössl who took the role of the second examiner without any hesitation.

Since this thesis also marks the end of my time as a student in Magdeburg, I want to thank all the people who helped me on my way through the bachelor and master. In this context my deep gratitude goes to my parents who supported me every single step of the way I have taken since I left our family home.
I was lucky to find many friends during my studies who made me feel home. It will be very hard to leave you behind!
Special thanks also to all the people who worked with me at Acagamics. While it may look like I was teaching others in the lectures I gave, you cannot imagine how many things I learned from you all.
Last but not least I want to thank all the extremely supportive professors and the university staff for making my education possible in the first place.

Abstract

Illumination and thus rendering is a computationally complex problem since correct lighting requires the interaction of every surface point with the entire scene. Accurate global illumination has become the standard in non-interactive applications, i.e. mainly in movies. During the last few years it has become more and more important in interactive applications like simulations and games as well. The visual importance, especially of the first indirect light bounce, is obvious for most scenes which would be widely unlit otherwise. Today, many games use a mixture of pre-computed and real-time techniques to achieve approximations of global illumination.

In this thesis, we present a new technique for completely dynamic real-time global illumination that works without any pre-computation. We compute single indirect bounce lighting using cascaded regular grids of light caches. Which caches need to be lit is determined at runtime. This allows fast processing and a very low memory footprint. Indirect light is obtained from a reflective shadow map, and saved into a spherical harmonics representation for each cache. It can then later be interpolated across several light caches. To compute accurate indirect shadows we use voxel cone tracing within a pre-filtered binary voxelization. Additionally, we propose hemispherical per-cache environment maps for a radiance representation that provides enough accuracy to enable indirect specular effects.

The work introduces the reader into the topic of global illumination and gives an overview over many related and similar approaches. The extensive evaluation section of this thesis shows that our technique has a very low memory footprint, works well with high screen resolutions and achieves competitive results both in terms of performance and quality. Improvements can be made especially in the area of performance of the indirect specular lighting and shadowing under higher quality configurations.

Index of Notation

The notations used in this paper are almost identical to those used in the book Physically Based Rendering [PH10]. Photometric quantities and relations are explained in more detail in Section 2.1.

Mathematical

x	Point in 3D space
$\overrightarrow{\mathrm{xy}}$	Normalized direction vector from x to y
$\mathbf{v}$	Direction vector in 3D space
$\mathrm{p}_x, \mathbf{v}_x$	x component of point / vector
$\mathbf{v} \cdot \mathbf{w}$	Dot product of vectors $\mathbf{v}$ and $\mathbf{w}$
$(\mathbf{v} \cdot \mathbf{w})^+$	Dot product of vectors $\mathbf{v}$ and $\mathbf{w}$ with negative values clamped to zero
$\mathbf{v} \times \mathbf{w}$	Cross product of vectors $\mathbf{v}$ and $\mathbf{w}$
$\|\|\mathbf{v}\|\|$	Euclidean length of vector $\mathbf{v}$
$\hat{\mathbf{v}}$	Normalized vector $\mathbf{v}$

Quantities & Functions

A	**Area**
ω	**Solid Angle**
ϕ	**Radiant Flux**, light power
I	**Radiant Intensity**, flux density per solid angle
E	**Irradiance**, flux density per area
L	**Radiance**, flux density per area per solid angle
ρ	**Reflectance**, ratio between incoming and outgoing flux
f_r	**BRDF**, function on the relation between irradiance and outgoing radiance

1 Introduction

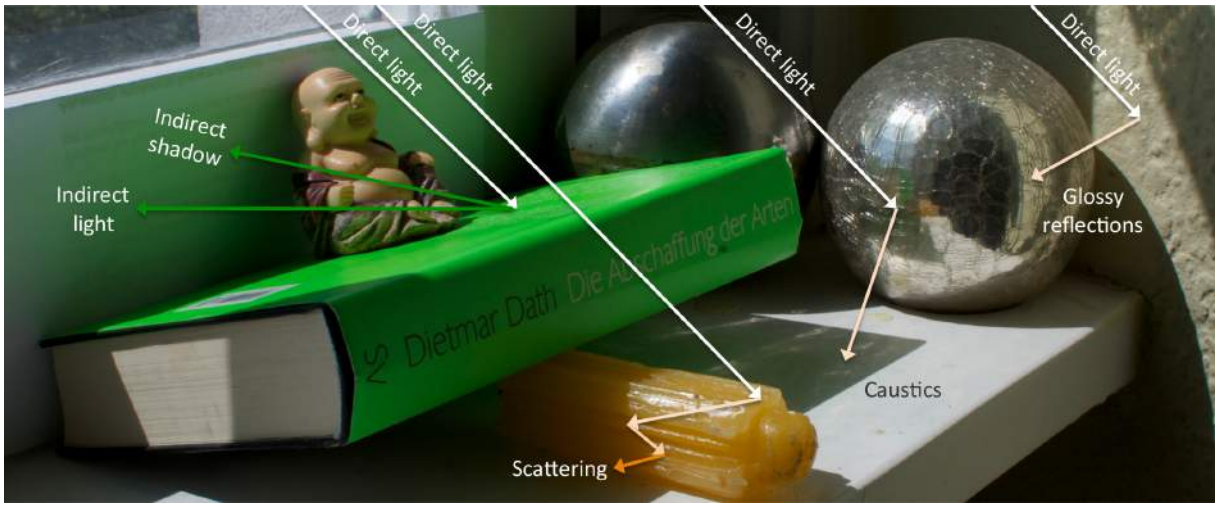


Figure 1.1: [RDGK12] Photography of a scene with various global illumination effects: Diffuse and specular bounces, caustics and scattering.

1.1 Introduction

Everything that human eyes can perceive is the result of light reaching the retina after interacting with matter. The way light interacts with our environment is very complex and always depends on a global context. There is a variety of natural phenomena like indirect light, (indirect) shadowing, (glossy) reflections and scattering which are impossible to simulate by local lighting models. The photography in Figure 1.1 demonstrates a few of them. Compared to the simulation of local light effects which only take an isolated surface point into consideration, the simulation of realistic global illumination effects as they occur in nature is extremely challenging both in terms of computational and algorithmic complexity.
Images which lack these global effects look synthetic and are often missing important cues which are needed to understand the interrelations of the depicted objects. Wherever 3D-scenes need to be displayed in an either believable or aesthetic manner, the simulation of global illumination becomes important. Some of these applications are movies, architectural visualizations, video games and professional training simulations. While the specific demands of these applications vary greatly, the underlying principles are the same as they are governed by the physical laws of light which are familiar to our visual system. Generally, these applications can be divided in two categories: Interactive and non-interactive. Where a non-interactive application like a movie or single images can afford computation times of several days, interactive applications need to compute at least

parts of the simulation just-in-time to provide the user with the expected feedback on his actions. A simulation is usually called interactive if images are rendered in less than 50ms (20 frames per second). However, frame rates need to be much higher to be comfortable for the user. Fast paced games usually profit from much better rendering times of more than 16ms (about 60 frames per second) [CCD06] and the vendors of the upcoming virtual reality displays recommend even shorter rendering times of about 11-8ms (90-120 frames per second) to reduce motion sickness [Pre14]. Naturally, real-time rendering that is performed on personal computers usually makes use of a set of approaches and algorithms, that differ from offline rendering which often runs on large computer clusters.

1.2 Motivation

Figure 1.2: Screenshot from the 2008 video game *Mirrors Edge*, featuring pre-computed indirect lighting.

In this work we want to introduce a new global illumination approach that is aimed for real-time rendering and does not need any pre-computation. To be able to display the aforementioned light effects in real-time, interactive applications often make use of various pre-computation steps under the assumption that specific parts of the scene do not change. In the extreme case this means that both scene and lighting conditions are assumed to be completely static which allows to compute a large variety of effects up-front. This has been common practice in many games for over a decade now, as can be seen for example in Mirrors Edge in Figure 1.2 where only the characters and very few dynamic objects are lit at runtime.
Obviously such techniques imply many restrictions on the design of virtual worlds and

may not be applicable at all. However, there are also solutions which less restrictions, for example to allow dynamic lights in a static scene or apply pre-computed lighting on dynamic objects (more on other techniques in Chapter 3). Of course there are various trade-offs between different approaches and so far there is no technique that is even close to simulating all global illumination effects in real-time in a reasonable quality (which can be considered to be an open problem for offline rendering as well).
A fully dynamic lighting approach not only frees a simulation of many technical restrictions, but also lowers the scene authoring iteration times as it is no longer necessary to wait for pre-computation processes to display the final result. Currently there are relatively few approaches that allow global illumination effects within completely dynamic scenes and lighting conditions. Of those, many come with a prominent artifacts like temporal incoherency (flickering). This is why many applications for which pre-computations are not applicable fall back to comparatively simple direct lighting and drop almost all important effects of global illumination.

1.3 Goals

The primary goal of this work is to explore the field of completely dynamic global illumination further and come up with a new solution which satisfies the conditions presented in this section.

We limit ourselves to what we think are the most important global illumination effects: Indirect diffuse lighting, indirect shadows and glossy reflections. Light can bounce many times in a scene before it reaches the viewer. In this thesis however, only the most important first indirect bounce will be handled.
As already mentioned, we strongly focus on avoiding pre-computation entirely and demand that all computations can be performed every frame. It should be possible to manipulate the scene's geometry and surface materials as well as lights and the camera freely at runtime.
The solution should run entirely on the graphics hardware to free the CPU for different work and avoid expensive transfers.

Our most important quality criterion is temporal coherency. Many dynamic global illumination approaches suffer from flickering when moving either light, objects or the camera. We believe that such artifacts are far more intolerable than physical inaccuracies of a lighting solution. Therefore, this work aims to avoid such artifacts as much as possible.
The work tries to accomplish appealing and credible lighting, physical correctness is secondary. Though, it should be based on physical laws to be able to make statements about its theoretical correctness.
The time that a single frame takes to render should be adequate to the computed detail at all times. This also means that we want to be able to trade quality for performance if necessary. We aim for real-time rendering performance on mid-range desktop hardware.
In contrast to pre-computation-based techniques and also some modern dynamic approaches we want to try to keep the memory consumption low.
Finally, the approach should be as invariant as possible to the chosen scene and its scale. That means that the technique should adapt well to various small and large scenes and still deliver convincing results.

1.4 Main Contribution

The main contribution of this thesis are:

- We introduce a new view-space based light cache selection in cascaded radiance volumes. Our approach is able to limit the lighting computations to the set of actually needed caches and keeps the overall memory consumption low, as we do not need to save any complex data in a grid. Additionally, we provide implementation details on how to speed up this process using shared memory.
- We present dynamic, scalable and comparatively accurate indirect shadowing for light caches using voxel cone tracing on groups of virtual lights.
- We propose the new idea of hemispherical specular environment maps for glossy reflections as an addition to classic spherical harmonic based radiance caches.

1.5 Thesis Outline

The next chapter will elaborate several theoretical and practical topics which are important to understand this thesis. Experienced readers may skip parts of this chapter. Since we are not able to introduce all necessary basics within the scope of this thesis, this chapter also recommends literature where newcomers can look up all relevant details.
Chapter 3 discusses various pieces of related work which are either similar, inspirational or fundamental to this thesis or solve similar problems.
Our own contribution, the dynamic radiance caching algorithm will be described in Chapter 4. Results are evaluated and discussed in Chapter 5.
Finally, Chapter 6 will summarize the thesis and give an outlook to possible future work.

2 Prerequisites

This chapter explains some of the underlying principles that are required to understand the remaining chapters of this thesis. Readers that are familiar with the basic concepts may skip the respective parts.

2.1 Theoretical Foundation

The following section explains several of the fundamentals of physically based rendering. Only a few basics which are helpful for a good understanding of this thesis will be discussed. For more detailed explanations the reader is referred to the book *Physically Based Rendering, From Theory to Implementation* by Pharr and Humphreys [PH10].

2.1.1 Physical Foundation

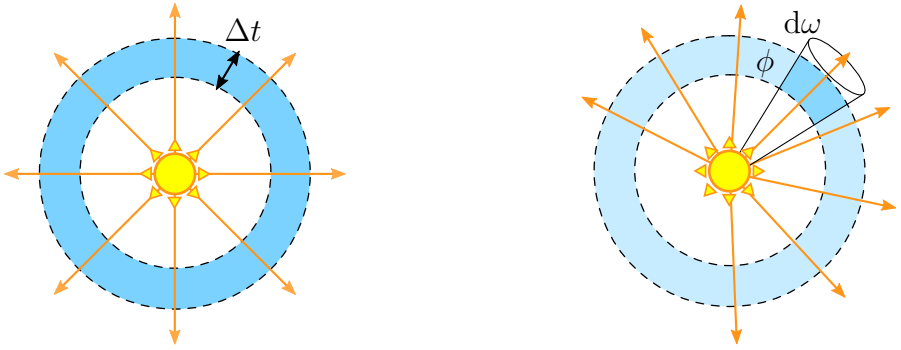
Generally, light is an electromagnetic wave which originates at a light source and is reflected or absorbed by the surfaces it hits. There are several physical phenomena like diffraction, polarization and interference which are usually ignored for rendering, as they often play a minor role for the appearance of a scene. Electromagnetic waves can also be represented by particles (photons) which have a certain wavelength that is responsible for the perceived color. We are only interested in the visible part of the spectrum which ranges from about $380\,nm$ to $780\,nm$. The number of photons registered by the eye is responsible for the perceived brightness.
As the human retina has only three receptors, each with a non-linear range of wavelength-support, it is usually sufficient to perform all calculations using only the three basic color stimuli red, green and blue [WS82] (RGB).

2.1.2 Radiometric Quantities

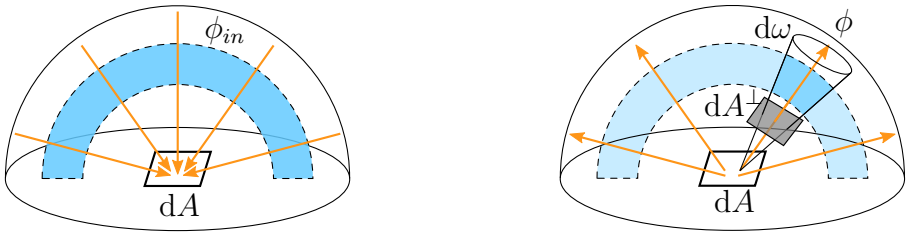
There are several basic quantities that describe the transport and perception of light. Several of them are measures over a *solid angle*, the 3D equivalent of a 2D arc length. The solid angle subtended by an object in a given point is computed by the projected area on the unit sphere around this point. Solid angles are measured in steradians sr. A solid angle representing the entire unit sphere has $4\pi\,sr$. As usual, we use the symbol ω both for the solid angle itself and the normalized directions of differential solid angles when integrating over (parts of) a sphere. For example, an integration of a direction dependent function $f(\mathbf{x})$ over all directions in a hemisphere is expressed as:

$$\int_{2\pi\, sr} f(\omega)\, \mathrm{d}\omega \tag{2.1}$$

$\mathrm{d}\omega$ denotes the differential steradian, while ω alone is a single direction on the unit hemisphere.



(a) Flux ϕ, light emittance per second. **(b)** Intensity $I = \frac{\mathrm{d}\phi}{\mathrm{d}\omega}$



(c) Irradiance $E = \frac{\mathrm{d}\phi_{in}}{\mathrm{d}A}$ **(d)** Radiance $L = \frac{\mathrm{d}\phi}{\mathrm{d}\omega \cdot \mathrm{d}A^{\perp}}$

Figure 2.1: Sketches visualizing different radiometric quantities.

Flux ϕ describes the energetic output of a light source (see Figure 2.1a). It measures how much energy a light source emits over time and is thus given in watts.

Intensity I gives how much flux ϕ per solid angle ω is emitted in a certain direction (see Figure 2.1b).

$$I = \frac{\mathrm{d}\phi}{\mathrm{d}\omega} \tag{2.2}$$

It is especially useful to describe in which direction light sources emit more or less light.

Irradiance E determines how much light (incoming flux ϕ_{in}) per area A a surface receives (see Figure 2.1c).

$$E = \frac{\mathrm{d}\phi_{in}}{\mathrm{d}A} \tag{2.3}$$

The opposite, how much light per area is leaving a surface, is also called radiant exitance or radiosity. However, as in many other works, we will use the term for both arriving and leaving flux per area.

The last and most important quantity is *radiance* L since it is the only "visible" measure as it corresponds to the seen brightness (see Figure 2.1d). It is defined as emitted flux ϕ per

solid angle ω per projected emitter area $A^{\perp}$ in respect to a detector.

$$L = \frac{\mathrm{d}\phi}{\mathrm{d}\omega \cdot \mathrm{d}A^{\perp}} \tag{2.4}$$

The projected differential area $\mathrm{d}A^{\perp}$ depends on the angle θ between emitter surface and detector direction:

$$\mathrm{d}A^{\perp} = \mathrm{d}A \cdot \cos\theta \tag{2.5}$$

2.1.3 Basic Relations and Common Types of Light Sources

A *point light* is an emitter without any extent, radiating equally in all directions with the intensity I. The irradiance E in respect to such a light source declines with the square of the distance d as the area "seen" by the source declines quadratic. Additionally, this area depends on the angle *theta* between the light direction and the surface normal. The resulting formula is known as photometric distance law:

$$E = \frac{I \cdot \cos\theta}{d^2} \tag{2.6}$$

A *Lambert emitter* is a hemispherical light source that appears equally bright from all directions. That means that the radiance L is independent of the angle under which the emitter is seen. This results in a cosine falloff of the intensity:

$$I(\theta) = (\cos\theta)^{+} \cdot I_0 \tag{2.7}$$

Another reoccurring light type is the *spot light*. Generally, a spot light is a point light with a limited region of intensities bigger than zero. There are many ways to define the attenuation curve of spot lights. Often images are used to model more complex patterns. In this thesis we define the intensity attenuation in direction d for a spot light that points in the direction l with an opening angle of α as following:

$$I(\mathrm{d}) = \left(\frac{(\mathrm{l} \cdot \mathrm{d}) - \cos\alpha}{1 - \cos\alpha}\right)^{+} \cdot I_0 \tag{2.8}$$

2.1.4 Bidirectional Reflectance Distribution Function

The *bidirectional reflectance distribution function* (BRDF) $f_r(\mathrm{x}, \omega_i, \omega_o)$ determines the ratio between the differential outgoing radiance $\mathrm{d}L$ in the direction ω_o and the differential irradiance $\mathrm{d}E$ from ω_i at the surface position x.

$$f_r(\mathrm{x}, \omega_i, \omega_o) = \frac{\mathrm{d}L}{\mathrm{d}E} \tag{2.9}$$

Generally, the values of the BRDF depend on the wavelength of the irradiance, but as mentioned earlier we use only RGB vectors instead. Physically plausible BRDFs are always positive, have an equal value if ω_i and ω_o are swapped (*Helmholtz reciprocity*) and are energy-preserving (i.e. sum of outgoing flux is smaller or equal to sum of ingoing

flux). The BRDF determines the color and reflectance properties of a surface. Thus, a given configuration is often also called *material*.

There are many different attempts to describe BRDFs by analytical models. The most basic one is the Lambert BRDF, also simply called *diffuse* material. Diffuse surfaces behave like Lambert emitters and distribute incoming radiance in all directions:

$$f_r(\mathrm{x}, \omega_i, \omega_o) = \frac{\rho_d}{\pi} \tag{2.10}$$

Where ρ_d is the diffuse *reflectance*. Reflectance ρ is generally defined as ratio between all outgoing and all incoming light:

$$\rho = \frac{\phi_o}{\phi_i} = \frac{\int\limits_{2\pi\, sr} L_o(\omega_o) \cdot \cos\theta_o \,\mathrm{d}\omega_o}{\int\limits_{2\pi\, sr} L_i(\omega_i) \cdot \cos\theta_i \,\mathrm{d}\omega_i} \tag{2.11}$$

It is often used in combination with a second BRDF to account for specular highlights. Note that it is possible to combine arbitrary BRDFs as long as the combined result is still physically plausible.

In this thesis we make mainly use of the *Blinn-Phong* BRDF [Bli77]. It is a very simple half-vector BRDF which delivers more plausible results than the more frequently used *Phong* BRDF [AMHH08, p. 251f]. A half-vector $\hat{\mathbf{h}}$ is the normalized sum of a direction to the light $\hat{\mathbf{d}}$ and a direction to the viewer $\hat{\mathbf{v}}$:

$$\hat{\mathbf{h}} = \frac{\hat{\mathbf{d}} + \hat{\mathbf{v}}}{||\hat{\mathbf{d}} + \hat{\mathbf{v}}||} \tag{2.12}$$

Since the original formulation of this BRDF is not energy preserving, we use the normalizated form as derived by Sloan and Hoffman [AMHH08, p. 257]. For a given surface normal $\hat{\mathbf{n}}$ and a half-vector $\hat{\mathbf{h}}$ it is defined as:

$$f_r(\hat{\mathbf{n}}, \hat{\mathbf{h}}) = \rho_s \cdot \frac{\gamma + 8}{8\pi} \cdot ((\hat{\mathbf{n}} \cdot \hat{\mathbf{h}})^+)^\gamma \tag{2.13}$$

Where ρ_s is the specular reflectance (also called specular color) and γ the specular exponent which steers the sharpness of the highlight. Both may vary across a surface.
Note that this does not account for energy preservation of a combination with another BRDF. For the typical use together with the Lambert BRDF, care has to be taken that $\rho_s + \rho_d$ is still not bigger than zero.

2.1.5 Rendering Equation

The well-known rendering equation by Kajiya [Kaj86] is a general description of the radiance in a given point x in a direction ω_o:

$$L_o(\mathrm{x}, \omega_o) = L_e(\mathrm{x}, \omega_o) + \int\limits_{2\pi\, sr} f_r(\mathrm{x}, \omega_i, \omega_o) \cdot L_i(\mathrm{x}, \omega_i) \cdot \cos\theta_i \,\mathrm{d}\omega_i \tag{2.14}$$

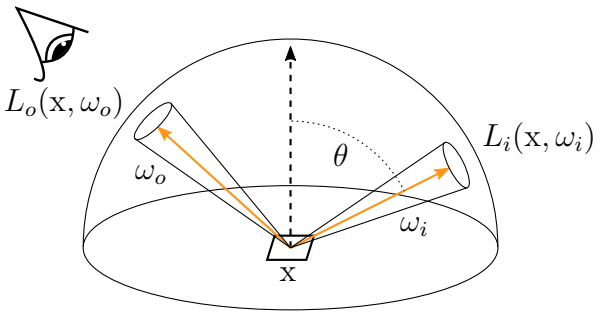


Figure 2.2: Schematic sketch of the rendering equation. The seen/outgoing radiance L_o depends on the incoming radiance L_i from all directions on the hemisphere.

Where L_e is the self-emission and L_i the incoming radiance for a given direction. f_r denotes the BRDF as described in the previous section. For all directions ω_i on the hemisphere, the integral sums up how much light is reflected to a observer in the direction ω_o. Figure 2.2 visualizes the concept. Note that the computation of every L_i again requires the evaluation of the entire equation at all points that reflect/emit light in this direction to the point x. This means that in theory the radiance at any point in any direction depends on all other points. Approaches that try to solve or approximate this recursive property of the rendering equation are thus performing *global illumination*, as opposed to *local illumination* where L_i only includes directly visible light sources.

One of the easiest ways of solving the rendering equation is *path tracing* which was presented in the same work as the equation. It solves the integral via a Monte Carlo integration over paths that originate at the viewer, instead of starting from the light sources: For each radiance sample (= screen pixel), multiple rays are shot into the scene. At each intersection the emissivity of the surface and the direct influence of one or more lights are evaluated (next event estimation) which involves additional "shadow rays" to check which lights can reach the surface directly. The ray is then reflected recursively, according to the BRDF of the hit surface. For surfaces with an overall reflectivity below one (practically all surfaces in reality), there is a chance that the ray will be absorbed which ends the path. Depending on the scene, it might take very long until a rendering process using path tracing converges.

There are several phenomena which are ignored by this basic form of the rendering equation. For example it is assumed that light travels through vacuum, i.e. no scattering or absorption within participating media occurs.

2.2 Modern GPU Pipeline and Capabilities

This section introduces several concepts of modern graphics hardware. These are not only necessary to understand implementation details, but also some of the decisions about the design of the presented algorithms. Since we rely on the open graphics application programming interface (API) OpenGL 4.5, we make use of terms and names as they are defined in the OpenGL (core) specification [Khr14a]. Note that these are sometimes different from those used by the hardware vendors and other APIs.

Throughout this section we point out several properties of the underlying hardware processes which are not specified by OpenGL. We try to avoid stating contemporary facts about specific pieces of hardware. As of this writing, the mentioned properties are at least true for the recent desktop GPUs of the three big GPU vendors Intel, AMD and Nvidia. There are several resources available on the inner workings of the hardware or at least on how to use it more efficiently [Int15, AMD15b]. For Nvidia hardware, we recommend the CUDA programming guide [Nvi15] which maps in many cases surprisingly well to other APIs like OpenGL. While not entirely up to date, Giesen's article *A trip through the Graphics Pipeline 2011* [Gie11] gives an excellent overview of the inner workings of today's graphic processors.

2.2.1 Rendering Pipeline Overview

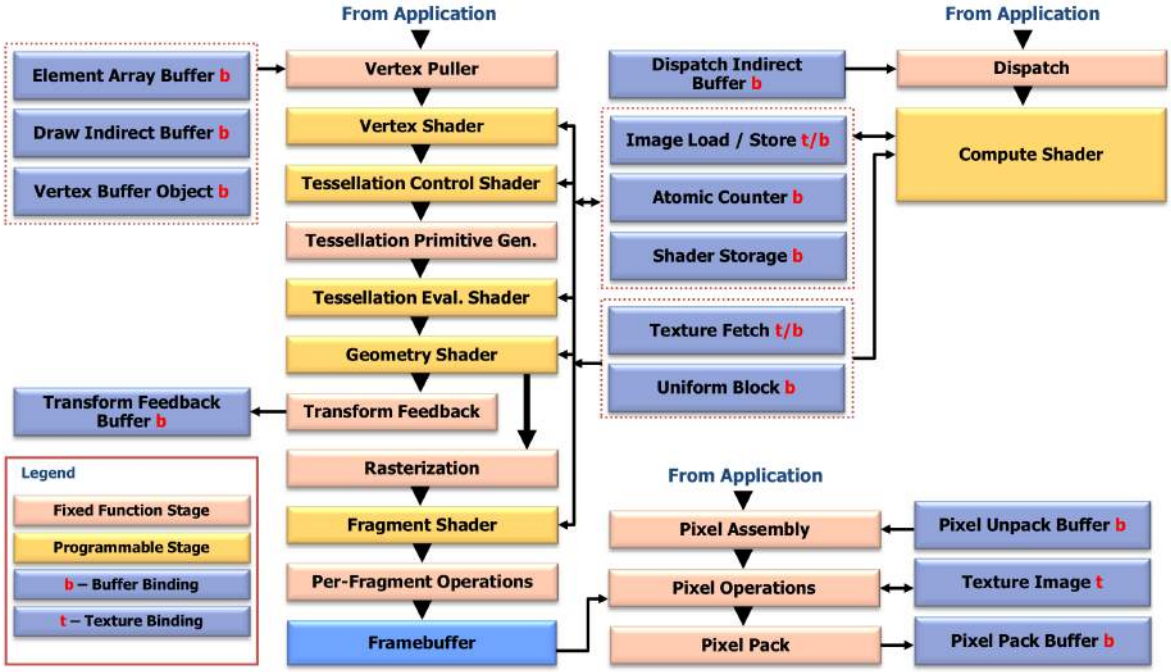


Figure 2.3: [Khr14a] OpenGL 4.5 pipeline overview.

Figure 2.3 gives a coarse overview over the modern OpenGL pipeline. The blue boxes represent different types of memory, or rather ways to access memory. As memory accesses play an important role in GPU programming, Subsection 2.2.2 will go into more detail. With the exception of compute shaders, all major programmable stages in the figure are depicted on the vertical starting with "Vertex Puller" and ending at "Framebuffer". The underlying processing is what is usually called the rendering pipeline which is traversed by the GPU on each draw call issued by the CPU. Compute shader are outside of this pipeline and explained separately in Subsection 2.2.3. Within the rendering pipeline there are also several *shader* stages: Vertex shader, two shaders for tessellation, geometry and fragment shader. Generally, shaders are small programs that are issued in parallel. They take data from the previous pipeline step and output data to the next, usually under the use of globally available resources. Shaders in OpenGL are written in a specific language called GLSL which is compiled by the graphics driver to run on the GPU. All

shaders in the pipeline need to provide a specific set of outputs and have access to several guaranteed inputs. However, it is possible to transfer almost arbitrary additional data between the stages. How this data is processed depends on the concrete stage and its current configuration.

Vertex Processing

The vertex shader fetches data from previously specified vertex buffers. It is invoked for each vertex. A single invocation has only access to the vertex it is assigned to. Each invocation must output at least a position which is later used in the rasterization stage. The vertex shading stage is mostly used for all sort of vertex transformations.

Afterwards, the optional tessellation stages are executed. These are skipped here since we do not use the hardware tessellation features in this thesis.
The optional geometry shading stage is invoked afterwards per primitive (= triangles, lines or points). It is able to generate new geometry on-the-fly. Before the advance of compute shaders, their ability to stream vertex data back to the memory (Transform Feedback) was very important for many algorithms. Nowadays, their use is very limited as many of their former tasks can be performed more efficiently by compute shaders. In this thesis we use geometry shader only to implement a real-time voxelization algorithm (see Section 4.3.4).

Pixel Processing

The rasterization stages generates pixels from the processed geometry and invokes the fragment shader for each of them. Depending on various multisampling parameters, this shader might be invoked for fragments of pixels, thus its name. The task of the fragment shader is to output one or more colors which will be written (or blended into) one or more framebuffers (also called render-targets). As long as not configured otherwise, the fragment shader is provided with perspective correct interpolated output from the last active stage preceding the rasterizer. If a depth buffer is used, the rasterizer will perform depth tests to fulfill a previously defined condition, usually to assure that nearer triangles are not overdrawn by more distant ones. For fragment shaders that optionally discard fragments or change a fragment's depth, such tests happen after the shading which might inflict a major performance penalty.

2.2.2 Memory Resources

In OpenGL there are basically only two types of memory resources: Textures and buffers. However, there are many ways in which these two can be accessed (see Figure 2.3). In general it is possible to access a resource that was created as a buffer like a texture but not the other way round. Conceptually, a resource is bound to a certain usage type and can then be accessed by shaders or, depending on the usage, may be used to influence scheduling within the graphics pipeline. Textures are generally defined as a collection of images, which consist of image elements, called texels. Contrary, buffers have no such

limitations and can contain virtually any data. Both types of resources lie in memory that is directly accessible by the graphics card. Whether those accesses are cached, how fast they are, how large underlying resources are allowed to be and where the memory lies physically depends very much on a resource's current usage and the hardware itself. The different types of access merely tell the driver how a resource is intended to be used and how it will be addressed. However, for all global resources a very high memory throughput coupled with rather high latencies can be expected. In practice this means that for each memory access, there should be as many arithmetical instructions as possible to hide the latency.

Transfers from and to CPU accessible memory are usually slow and can introduce huge stalls between CPU and GPU which can normally work independently [HM12]. Therefore, practically oriented algorithm design usually tries to avoid memory transfers wherever possible.

Textures

Most of the time, textures are accessed read-only through sampler objects (*Texture Fetch* box in Figure 2.3) and normalized floating point texture coordinates. Sampler objects offer different filter functionalities which are implemented in hardware and make use of texture caches on the devices of all major vendors. Contrary, *Image Load / Store* allows random read/write access to specific texels and enables atomic operations. However, these accesses might be uncached and rather slow in comparison.

There is wide range of different data formats that can be stored in texels. This includes various compressed, integer, floating point and fixed point formats. As a general limitation, all formats have between one and four color channels (though a texel may not necessarily store color).

There are three distinct kinds of textures: 2D textures, 3D (volume) textures and cube-maps. 3D and cube-map textures are composed of multiple layers. Cube-maps have a layer for each side of a cube and may be addressed using a direction. Volume textures are composed of depth layers, but accesses may perform filtering in all three dimensions. Generally, it is possible to use a layer of an arbitrary texture type as rendertarget to update its contents with the results of a rendering process. However, there are a few texture formats that might not be supported to be used as render-target.

All texture types support so called *mipmaps*. Mipmapping denotes the process of repeatedly creating pre-filtered, half-sized versions of the original texture which are called mipmaps. Mipmaps are foremost used for filtering when a texture is minimized, i.e. when many texels cover a single pixel. The original paper [Wil83] that presented mipmaps already proposed to use them for other purposes like filtering of highlights with varying solid angle as well. Similarly, they will be later used to filter outgoing radiance for varying BRDF parameterization (see Subsection 4.3.3).

Buffers

To allow access from arbitrary shaders, a buffer might either be used as *Uniform Buffer Object* (UBO) for read-only access or as *Shader Storage Buffer Object* (SSBO) which

allows random writes and even a limited set of atomic operations. The values in an UBO are often called constants, as they are constant during the execution of a shader. Both buffer access types come with special layouting rules and different size limitations.
Another important use of buffer resources is to provide vertex input in the form of vertex and element array buffers. Element array buffers determine how vertices form basic primitives like triangles. They are also known as index buffers.
Using *Draw/Dispatch Indirect Buffer* bindings it is possible to specify the parameters for draw/dispatch calls via a GPU resource (which might have been used as a SSBO earlier).

2.2.3 Compute Shader

The compute shader is the newest addition to the family of shaders in OpenGL. It provides an easy interface for general purpose computations on the GPU (GPGPU), i.e. operations which are not bound to a rendering process. GPGPU became increasingly popular with the advance of programming interfaces specifically designed to perform any parallel computation on the GPU such as CUDA and OpenCL. The main advantage of using compute shaders within OpenGL, over using other APIs, is that all OpenGL resources are directly available. This makes compute shaders an excellent choice for algorithms that are part of the rendering process, but do not fit into the classic concept of the rendering pipeline with its fixed function vertex fetching and rasterization.

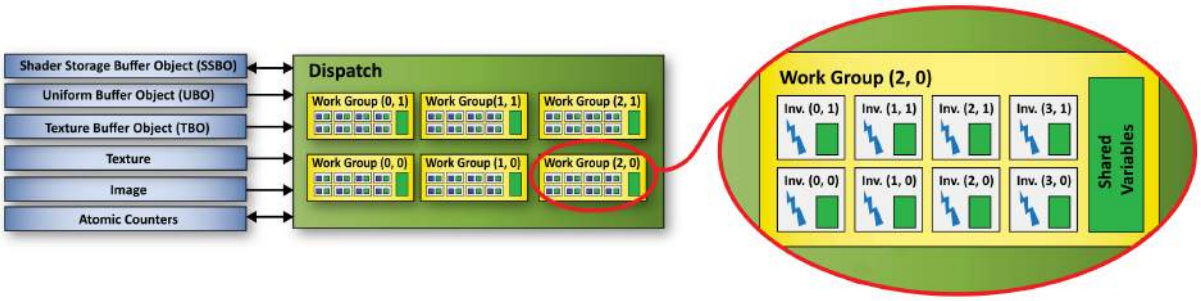


Figure 2.4: [Khr14b] OpenGL 4.5 compute shader programming model scheme.

Figure 2.4 visualizes the basic programming model of compute shaders. A *dispatch* is organized into a number of *work groups*. The number of work groups can be determined at runtime. Each work group consists of a fixed number of threads which is given by the shader at its compilation time. All threads within a dispatch execute the same code and differ only in their unique thread IDs. Threads within a single work group can be synced to each other (via a barrier) and have access to a certain amount of shared memory. The size of the shared memory is limited by hardware restrictions. Access to shared memory is much faster than to global resources, but is still slower than to the registers of a thread. As with other memory (access) types, there are several properties that, while partially hardware specific and not specified by OpenGL, are important to keep in mind for efficient use.

The ratio between the number of threads actually running simultaneously and the maximum number of threads the hardware can execute in parallel is called occupancy [Nvi15, AMD15a]. Occupancy depends on the chosen group size, the number of registers needed

and the amount of shared memory used. Low occupancy means that wide parts of the available computational resources remain unused. While issues with low occupancies may occur in any shader stage, the compute shader is more prone to such errors as the programmer has much more influence on the scheduling than on the shaders of the rendering pipeline.

2.3 Realtime Rendering

This section gives an overview over a few important aspects of realtime rendering. However, it is assumed that the reader is already familiar with the most basic concepts like (camera-)transformations, (triangle) geometry setup and normal mapping. Information about these topics can be found in the book *Real-Time Rendering* by Akenine-Möller et al. [AMHH08].

2.3.1 Deferred Shading

In classic *forward rendering* the rendering and shading of objects is coupled. To be able to compute direct lighting for multiple lights, shading procedures either involve loops over all light sources or it is necessary to render all objects additively multiple times computing a fixed number of lights each time (multipass lighting).

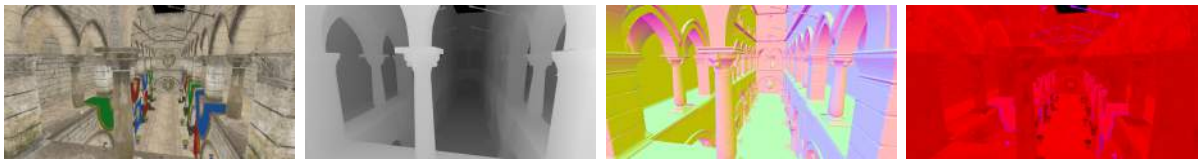


Figure 2.5: Example for a G-buffer with (left to right) diffuse reflectance, depth, normals and specular BRDF properties.

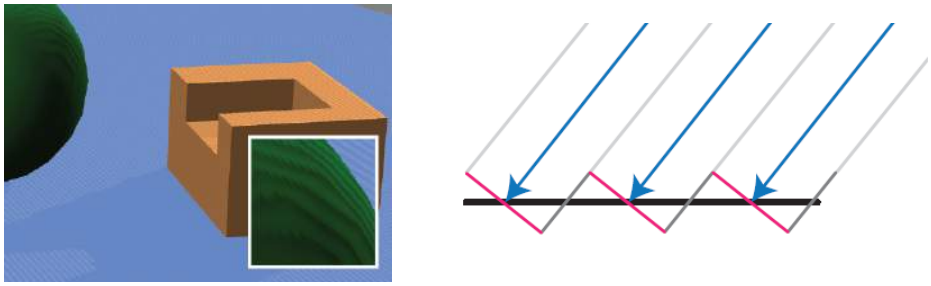
Deferred shading decouples object rendering and pixel shading by saving all information needed for shading in the so called *G-buffer*. This buffer saves a depth (which can be used to reconstruct the world position), normal and material properties for each screen pixel. Figure 2.5 gives an example for a possible configuration. Afterwards this information is used to compute dynamic illumination in one or more screen-space passes without re-rendering the scene geometry itself. The approach scales very well with many lights and can reduce the general complexity of a renderer considerably. Overdraw occurs only during the G-buffer creation which makes expensive BRDF calculations more viable. The main drawbacks of deferred shading include difficulties with multisampling-antialiasing and transparent objects which cannot be handled by this technique at all.

2.3.2 Shadow Mapping

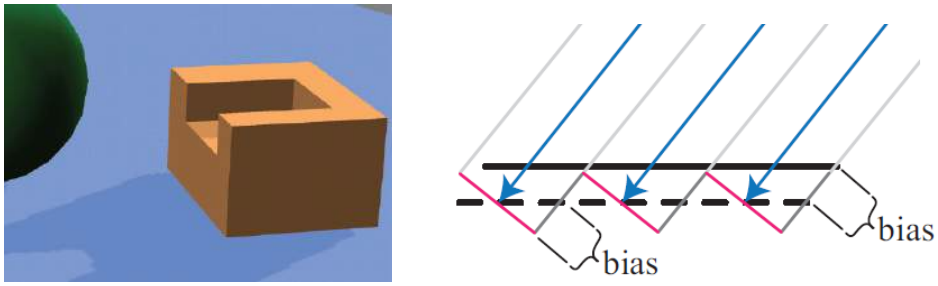
Depending on the definition, direct lighting may already no longer be classified as local illumination since it requires to determine the direct visibility of all light sources. Visibility checks in general are a "global problem" since it is necessary to take the entire

scene between light emitter and receiver into account. Especially for non-dimensionless light sources like area or environment lights (light from the entire sky) direct visibility determination can be a difficult problem.

The most popular technique for direct shadows from dimensionless light sources is *shadow mapping* [Wil78] since it works very well on today's graphics hardware. A shadow map is a texture containing depth values rendered with the view of a corresponding light source. When shading an object, the distance to the light source is compared with the depth value stored in the shadow map for the respective direction from the light. If the stored value is smaller, the pixel lies in the shadow, otherwise not.



(a) Self-shadow aliasing caused by texels that are both below and above the surface.



(b) Bias fixes above issue, but now the object seems to float.

Figure 2.6: [AMHH08, p.351] Shadow map samples surface, shown by blue arrows. The red line shows the distance that the shadow map saves.

There are various extensions to shadow mapping that try to improve the sampling quality or simulate non-dimensionless light sources. Shadow mapping is prone to artifacts like edge aliasing (due to the limited map resolution) and self-shadow aliasing, also known as "shadow acne". Surfaces are sometimes incorrectly considered shadowing themselves since point samples from the shadow map are used to represent an area's depth, see Figure 2.6a. The problem can be overcome by adding a bias to the depth values. This effectively moves the surface away from the light which again leads to another artifact known as "Peter Panning": For a too large bias, objects are disconnected from their shadows which leads to a floating appearance, see Figure 2.6b.
To alleviate this issue, Holbert [Hol11] introduced a normal dependent offsetting approach. Contrary to the classic constant offset, the offset is computed based on the angle between surface normal and light direction. This allows much lower depth bias values without self-shadow aliasing.

In our implementation we use shadow maps combined with normal offset mapping to determine the direct light visibility.

2.3.3 Gamma Correction & HDR Rendering

Displays generally cannot output arbitrary radiance intensities. In which range possible values are, depends on the concrete display hardware and its configuration. The input format of most displays is eight bits per channel which offers only a *low dynamic range* (LDR). However, the LDR contents are not directly interpreted as a radiance but gamma corrected using a transfer function. Typically the inputs are assumed to be encoded in the *sRGB* color space.

All this has several implications on how rendering needs to be performed. While input textures are typically encoded in sRGB as well, all rendering operations have to be performed in linear color space. Note however, that it is not possible to convert a 24bit sRGB value to a linear RGB value without either information loss or more memory per color. Therefore all operation result have to be saved in *high dynamic range* (HDR) formats. Most common for this task are floating point render-targets. The final HDR result needs to be converted back to sRGB which can be done automatically by modern GPUs. Additionally, tone mapping may be applied to map a wide range of illumination levels onto the limited color space.

All images in this thesis were correctly computed in linear space. Since indirect lighting is usually much dimmer than direct light, there is a wide range of intensities we want to display. To account for this, we apply the logarithmic global Drago tonemapping operator [DMAC03] before gamma correction where not noted otherwise. Contrary to many other works we do not scale indirect lighting separately at any point.

2.4 Spherical Harmonics

Spherical harmonics (SH) are a set of orthonormal basis functions, analogous to the Fourier transform's basis functions, but defined on the surface of a sphere. They are often used in computer graphics to approximate spherical functions like visibility, lighting or reflectance. Similarly, they will later be used in Subsection 4.3.2 to describe the total irradiance for a given normal.
More detailed explanations can be found for example in [Gre03, Slo08] on which this summary is based.

2.4.1 Definition

The SH functions in general are defined on imaginary numbers, but for use in computer graphics usually only real-number SH are relevant.

$y_l^m(\theta, \varphi)$ represents a SH function. The angles θ and φ give a point on the unit hemisphere. To convert from spherical to Cartesian coordinates we use the following conversion:

$$(\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta) \rightarrow (x, y, z) \tag{2.15}$$

The index l represents the SH *band*. Each band consists of polynomials of the degree l (for zero it is a constant function, for 1 it is linear, etc.). There are $2l + 1$ functions for a given band. Accordingly, the higher the band index l, the higher is the

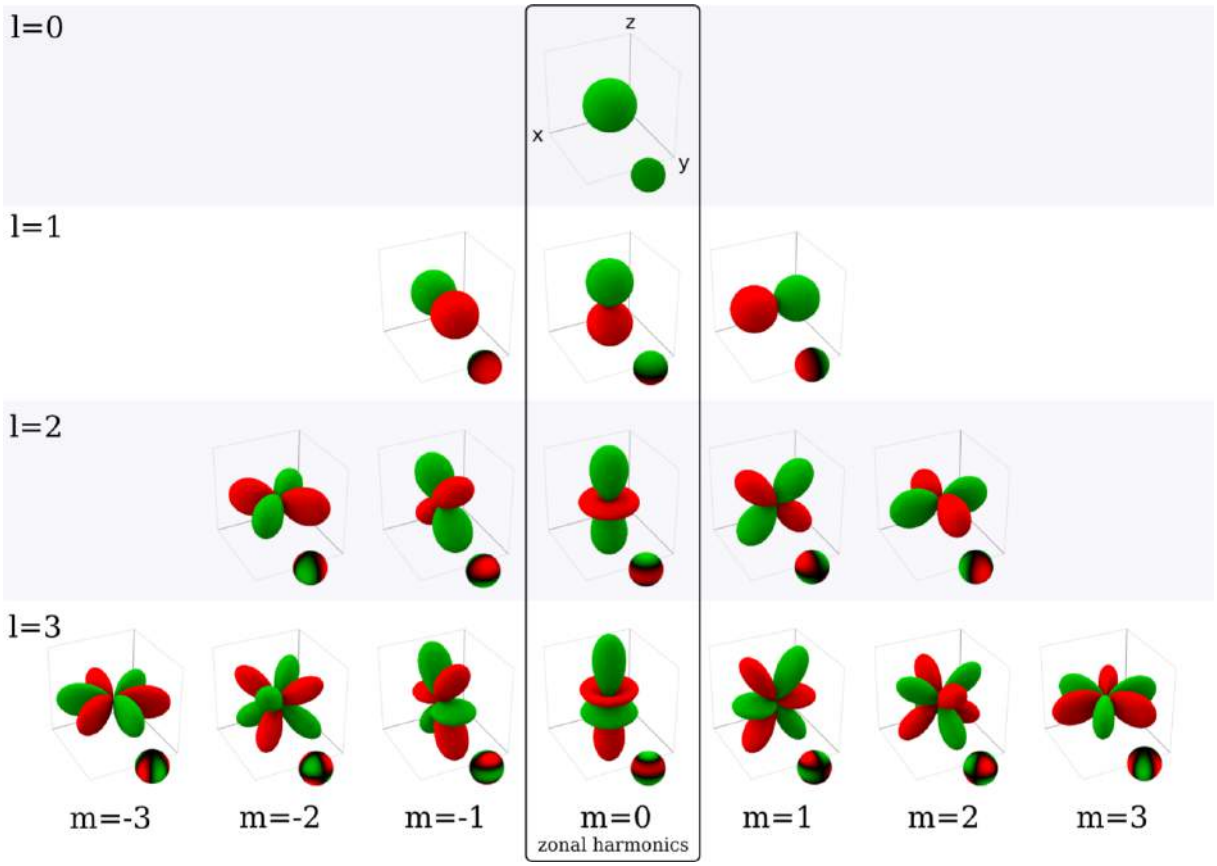


Figure 2.7: [Gue07] The first four SH bands plotted as unsigned spherical functions by distance from the origin unit sphere. Green are positive values, red are negative ones.

frequency of the information that is encoded in this band. Figure 2.7 visualizes the first four bands.

$y_l^m(\theta, \varphi)$ is defined as:

$$y_l^m(\theta,\varphi) = \begin{cases} \sqrt{2} K_l^m \cos(m\varphi) P_l^m(\cos\theta) & m > 0 \\ \sqrt{2} K_l^m \sin(-m\varphi) P_l^{-m}(\cos\theta) & m < 0 \\ K_l^0 P_l^0(\cos\theta) & m = 0 \end{cases} \tag{2.16}$$

Where $P_l^m(x)$ are the associated *Legendre* polynomials and K_l^m are normalization constants defined as:

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}} \tag{2.17}$$

The associated Legendre polynomials are recursively defined as following:

$$\begin{aligned} P_0^0(x) &= 1 \\ P_m^m(x) &= (1-2m) P_{m-1}^{m-1} \\ P_{m+1}^m(x) &= x(2m+1) P_m^m \\ P_l^m(x) &= \frac{x(2l-1) P_{l-1}^m(x) - (l+m-1) P_{l-2}^m}{l-m} \end{aligned} \tag{2.18}$$

2.4.2 Projection, Reconstruction and Convolution

Each spherical function $f(\theta, \varphi)$ can be represented with an infinite series of SH functions, each weighted by a SH coefficient. To calculate a SH coefficient for a given band of a spherical function f (i.e. to project f), the product of f and the SH function y needs to be integrated:

$$c_l^m = \int_{4\pi\, sr} f(\omega) y_l^m(\omega)\, \mathrm{d}\omega \tag{2.19}$$

By truncating the SH representation to n bands, a function f can be approximated:

$$\widetilde{f}(s) = \sum_{l=0}^{n-l} \sum_{m=-l}^{l} c_l^m y_l^m(s) \tag{2.20}$$

Since SH are orthonormal basis functions, the integral of the product of two functions $\widetilde{f}$ and $\widetilde{g}$, is the same as the dot product of their coefficients f_l^m and g_l^m:

$$\int_{4\pi\, sr} \widetilde{f}(\omega) \cdot \widetilde{g}(\omega)\, \mathrm{d}\omega = \sum_{l=0}^{n-l} \sum_{m=-l}^{l} f_l^m g_l^m \tag{2.21}$$

This property is extremely useful as it allows to collapse integrations over the entire sphere into a single dot product which might be (depending on how many bands are used) rather cheap to evaluate.

2.4.3 Rotation and Zonal Harmonics

SH functions are rotationally invariant: For a function f and its by R rotated copy g, the following holds for their corresponding SH projections $\widetilde{f}$ and $\widetilde{g}$:

$$\widetilde{g}(s) = \widetilde{f}(R(s)) \tag{2.22}$$

This means that it is possible to rotate the SH representation of a function, without losing precision (other than rounding errors).

Later on this property will be very useful combined with *Zonal Harmonics*. Zonal Harmonics are SH projections of functions that have rotational symmetry around an axis. If the axis is Z (in respect to Equation 2.15), then only the c_0^m coefficients of the SH projection will be non-zero.
Rotation of such Zonal Harmonics z_l to a new direction d is generally much easier than arbitrary SH. The resulting, rotated SH coefficients are obtained as:

$$c_l^m = \sqrt{\frac{4\pi}{2l+1}} z_l y_l^m(d) \tag{2.23}$$

3 Related Work

This chapter gives an overview of related realtime global illumination techniques. Ritschel et al. [RDGK12] already gave a great overview over this field for all approaches before 2012. Therefore, we will focus on newer methods and those which are either fundamental, similar or inspirational to this thesis instead of trying to cover the whole field.

3.1 Many-Lights Methods

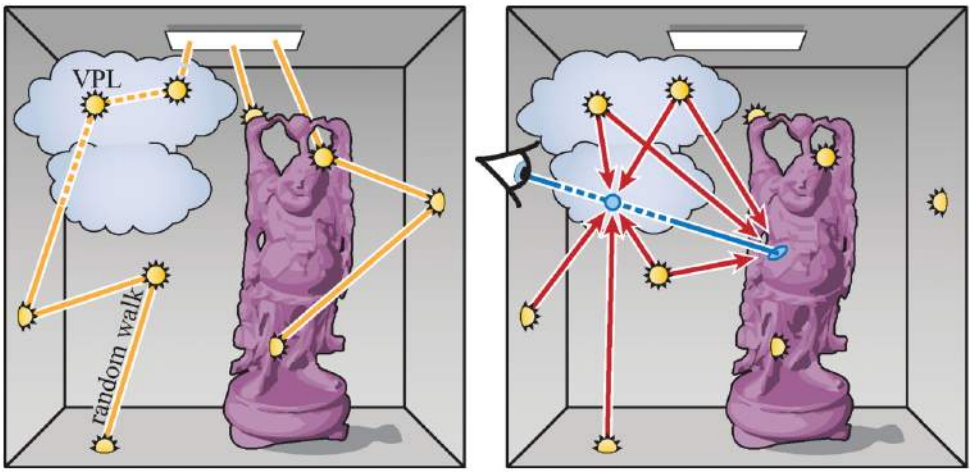


Figure 3.1: [DKH+14] Two passes of many-light algorithms: Left distribution of virtual lights, right direct lighting using virtual lights.

Many-light methods reduce the global illumination to the smaller problem of calculating the direct lighting of many virtual light sources. The basic principle was first described in Keller's work on Instant Radiosity [Kel97] (IR). All many-light approaches perform two passes (see Figure 3.1): First, virtual light sources are distributed in the scene. Then, each virtual light source performs direct lighting.
The main advantages of the many-light approach in general are its relatively easy unified framework and its good scalability in performance and quality. Global illumination using M virtual lights can generally be expressed as following:

$$L_o(\mathrm{x}, \omega_o) = \sum_{i=1}^{M} f_r(\mathrm{x}, \overrightarrow{\mathrm{x}\mathrm{x}_i}, \omega_o) \cdot G(\mathrm{x}, \mathrm{x}_i) \cdot \hat{V}(\mathrm{x}, \mathrm{x}_i) \cdot I_i(\overrightarrow{\mathrm{x}_i\mathrm{x}}) \tag{3.1}$$

Where $G(\mathrm{x}, \mathrm{x}_i)$ is the geometry term that describes the light transfer from the virtual light in x_i to the surface in point x. It depends on the type of the virtual light source. The visibility $\hat{V}(\mathrm{x}, \mathrm{x}_i)$ of a virtual light is either zero or one for dimension-less lights. I_i gives the

intensity distribution of the ith virtual light which is evaluated in the direction $\overrightarrow{\mathrm{x}_i\mathrm{x}}$ from virtual light to the surface point. It is defined by the product between BRDF in the point x_i and the incoming flux x_i originating from the direction ω_i.

$$I_i(\overrightarrow{\mathrm{x}_i\mathrm{x}}) = f_r(\mathrm{x}_i, \omega_i, \overrightarrow{\mathrm{x}_i\mathrm{x}}) \cdot \phi_i \tag{3.2}$$

The original instant radiocity approach distributes the light by tracing photon rays from the light source that are bounced inside the scene. Like more recent techniques, IR uses virtual point light sources (VPL).
Dachsbacher et al. [DKH+14] recently did a detailed survey on many-lights for both offline and real-time rendering. We focus here mainly on important real-time approaches.

3.1.1 Reflective Shadow Mapping

One of the most important real-time global illumination algorithms is reflective shadow mapping by Dachsbacher et al. [DS05] on which many other approaches rely, including the one presented in this thesis. It handles the virtual light distribution with the rasterization of an augmented shadow map, the reflective shadow map (RSM). The RSM does not only contain depth as normal shadow maps, but also additional layers containing reflected (diffuse) flux and the normal of the seen surface. Each texel of this shadow map is then used as virtual point light. The original paper shades each pixel with a given number of sampled VPLs without taking indirect shadows into account. Also, the technique handles only a single indirect bounce of diffuse lighting since the VPLs are strictly speaking hemispherical Lambert emitters.

While RSM offers an easy way to generate virtual lights in real-time, it does basically not alter the way how those lights are applied and how their visibility is determined. There are a few techniques that try to reduce the number of virtual lights and create few virtual area lights (VAL) by clustering. Both Dong et al. [DGR+09] and Prutkin et al. [PKD12] suggest to cluster VPLs using k-means. The former performs the clustering in world-space, the later directly in image space of the RSM. Both approaches suffer from temporal coherence problems since the size and position of the clusters can change rapidly.

3.1.2 VPL Bias and Compensation

Using virtual point lights, the geometry term is defined as following:

$$G(\mathrm{x}, \mathrm{x}_i)_{VPL} = (\hat{\mathbf{n}}_i \cdot \overrightarrow{\mathrm{x}_i\mathrm{x}})^+ \cdot \frac{(\hat{\mathbf{n}} \cdot \overrightarrow{\mathrm{x}\mathrm{x}_i})^+}{||\mathrm{x} - \mathrm{x}_i||^2} \tag{3.3}$$

Where $\hat{\mathbf{n}}$ and $\hat{\mathbf{n}}_i$ are the surface normals in point x and x_i respectively. Note that the geometry term is a combination of the photometric distance law and the definition of intensity for Lambert emitter as described earlier in Subsection 2.1.3.

The inverse-squared distance between shading point and light source introduces a singularity, as it reaches zero in the limit. Therefore, the contribution of a VPL converges to infinity at its position. The result are bright splotches, mostly visibility in creases, as seen left in Figure 3.2.

Figure 3.2: Left: Small bright splotches due to inverse squared distance in geometry term. Right: Using virtual area lights as proposed in [LB13].

Typically these artifacts are avoided by clamping the geometry term to a user defined maximum which obviously introduces a bias. There are several bias compensation techniques: Kollig and Keller [KK04] shoot rays to nearby surfaces and evaluate the lighting there to transport it back to the original surface. However, this technique may degenerate to path-tracing and is thus not suitable for interactive applications. Other techniques try to compensate the bias approximately. E.g. the bias compensation by Novák et al. [NED11] stores the bounded transport and computes the residual transport via a screen-space post-processing step. Although it misses invisible surfaces, it is both efficient and plausible in its results.
There are also several approaches that use different types of virtual light sources instead of VPLs. Hašan et al. [HKWB09] introduce a new light type, the virtual spherical light (VSL). Instead of a performing point-to-point evaluations, this type of light is evaluated by an integration over the solid angle subtended by the light source. Their approach is biased but does not cause bright splotches and preserves the overall energy by redistributing.
Lensing et al. [LB13] use approximated disc shaped lights. Since they use reflective shadow maps, the area of these virtual area lights A_i can easily be estimated by the distance of the VAL to the light source and the solid angle a RSM texel subtends. They show that if all surfaces are assumed to be diffuse, the lighting can be solved analytically by a simple change in the geometry term:

$$G(\mathrm{x}, \mathrm{x}_i)_{VAL} = (\hat{\mathbf{n}} \cdot \overrightarrow{\mathrm{xx}_i})^+ \cdot \frac{(\hat{\mathbf{n}}_i \cdot \overrightarrow{\mathrm{x}_i\mathrm{x}})^+}{||\mathrm{x} - \mathrm{x}_i||^2 + A_i} \tag{3.4}$$

Using this geometry term, splotches at specular surfaces with low roughness can still occur. However, most singularities are no longer visible (see Figure 3.2 right). We use this technique in our approach since it is extremely fast, easy to implement and more plausible than a bounded geometry term.

Bias compensation in participating media are considerably more complex but not handled in this thesis.

3.1.3 Shadows for Many-Lights

Computing visibility for a large number of virtual lights is a difficult problem for real-time renderers since classic shadow computations like shadow mapping are far to expensive for this task. There are a few methods that try to optimize shadow mapping for rendering large amount of lights:
Imperfect shadow maps by Ritschel et al. [RGK+08] leverage the fact that exact secondary visibility is not needed. First, points are placed at random locations on the scene geometry. Shadow maps are then computed using these points instead of triangles which is much faster. Holes in the shadow maps are filled with a push-pull heuristic. The succeeding paper [REH+11] makes the technique (both VPL placement and shadowing) view adaptive and allows larger scenes.
Olsson et al. [OSK+14] store shadow maps on a large virtual texture. Each frame it is decided which lights actually need to compute shadow maps, how high their resolution should be and which objects act as occluders. All of these operations are performed on the GPU.

Other methods like Micro-rendering by Ritschel et al. [REG+09] or ManyLoDs by Hollänger et al. [HREB11] use hierarchical data structures to be able to render objects at the exactly needed detail. This allows very fast approximations of visibility. These techniques have also several other implications, for example on how shading performed, which is why they are often separately categorized as *point-based GI*.

3.2 Caching Methods

Many recent real-time global illumination methods, including our own, compute indirect lighting only for special *light caches* and interpolate the results of those across many pixels. Which quantities a light cache saves varies from technique to technique as well as the strategy that is responsible for creating light caches either at runtime or as a pre-computation step. Our approach relies on dynamic cache placement, its implications and trade-offs will be discussed in Subsection 4.2.1. Because caches are often placed sparsely, most of the following techniques are prone to light bleeding artifacts, i.e. light "seeping" incorrectly through blockers.

3.2.1 (Irr-)Radiance Caching

The basic idea of light caching was first introduced by Ward et al. [WRC88] in the form of Irradiance Caching (IC) to speed up path-tracing. Each time a ray intersects a diffuse surface it first checks if there are nearby caches from which the irradiance can be interpolated. If not, the irradiance is computed and a new cache is created at this place. There are many extensions to this method, most notably Radiance Caching by Krivanek et al. [KGPB05] (RC) which supports glossy surfaces by saving incoming radiance encoded in a high order spherical harmonics representation.

Pre-convoled Radiance Caching by Scherzer et al. [SNRS12] is able to perform radiance caching with a single indirect bounce at interactive frame-rates. As in the original radiance caching, caches lie on surfaces. For each cache a hemisphere of all incoming radiance values

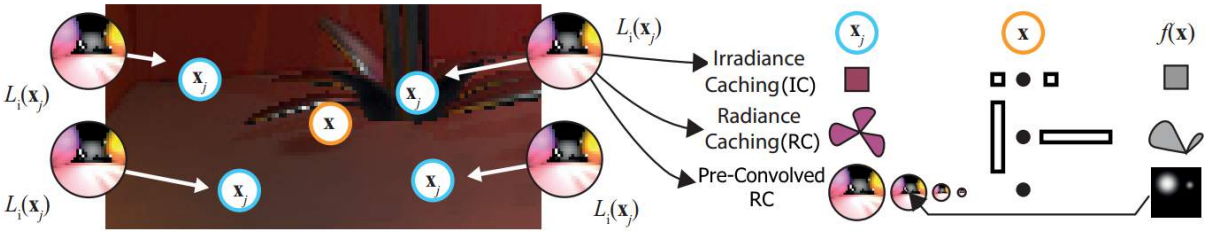


Figure 3.3: [SNRS12] Storage and evaluation of Irradiance (IC), Radiance (RC) and Pre-Convolved Radiance Caching. For all techniques cache items (blue circles) are interpolated for image pixels (orange circle). How cache items are evaluated differs (right): IC stores an irradiance value which is interpolated (top row). RC (middle row) stores radiance in an SH representation and convolves it with the likewise stored BRDF. Pre-convolved RC (bottom row) store mipmaps of incoming radiance which can be queried using the BRDF.

is rendered, as depicted in Figure 3.3. To be able to quickly evaluate arbitrary specular exponents (the paper makes use of the Phong BRDF), the result is pre-convolved using mipmapping. Caches are placed statically and apply their results to the screen using splatting, i.e. rendering them directly to the deferred shading buffer. Clustured Pre-Convolved Radiance Caching, an extension to the previous method by Rehfeld et al. [RZD14], places the radiance caches according to screen-space clusters which makes the technique more scalable but less temporal coherent. Additionally it uses cone tracing (which will be explained in Section 3.5) in a prefiltered voxel grid to gather radiance from the caches instead of splatting them to the screen.

3.2.2 (Irr-)Radiance Volumes

Figure 3.4: [GSHG98] Visualization of an irradiance volume. Each sphere visualizes a light cache colored by the local irradiance for each direction.

Irradiance volumes as proposed by Greger et al. [GSHG98] are very popular to light dynamic objects in otherwise static environments. An irradiance volume is a regular

grid in which each grid cell saves precomputed irradiance values for all possible surface normals. The original work uses hemispherical grid mappings for this task. However, other representations like spherical harmonics are more common in praxis. Each surface can now look-up and interpolate precomputed irradiance values from the nearest grid cells to perform diffuse lighting. Note however that this method can not handle light bounced off or occluded by dynamic objects.

The advantage of saving irradiance for all possible normals over saving incoming radiance for each direction, is that an irradiance signal has a much lower frequency than radiance. Only a single irradiance value is needed for diffuse lighting of a surface (i.e. a given normal) which is equivalent to a hemispherical integration over the incoming radiance. Nonetheless, if arbitrary BRDFs are to be supported, the total incoming irradiance for a given normal is not sufficient and integration over radiance is necessary. Note however, that for a spherical harmonics representations, integration over a lobe given in SH has the same costs as evaluating the SH coefficients for a certain direction (as earlier described in Subsection 2.4.2).

Real-Time

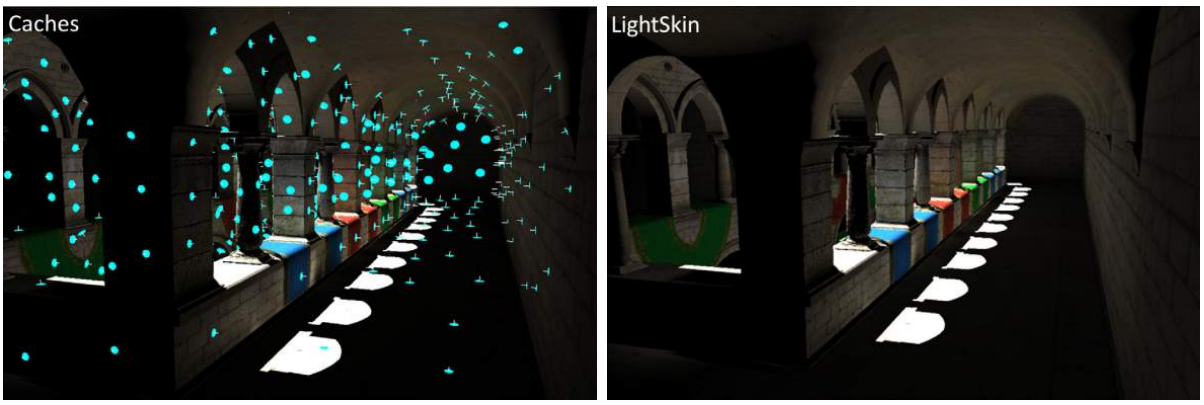
Njasure et al. [NPG05] compute radiance volumes at runtime with interactive frame rates by rendering a cubemap at each grid point to gather radiance which is afterwards encoded in a low number of spherical harmonics coefficients.

Radiance Hints by Papaioannou [Pap11] retrieve radiance from a RSM and approximate multiple diffuse light-bounces between the light caches. Like the RSM algorithm on which the technique is based, occlusion of VPLs is not handled. For the secondary bounces however, visibility of the radiance hints (= light caches) is approximated by the minimum and maximum distance of the used RSM samples.
The recent approach of Vardis et al. [VPG14] improves upon Radiance Hints by using chrominance compression and a occupancy grid to reduce cache updates. They use less SH bands for the chroma part of the radiance to allow higher order bands for the more import luminance portion. Which radiance caches contribute to the scene's surfaces is determined by down-sampling a real-time binary voxelization. Contrary to Radiance Hints, caches in mid-air will not be updated. The visibility of virtual light sources is determined by sparse sampling in the binary voxelization along the connecting rays.

Radiance volume approaches in general are rather robust against temporal coherence issues. The cache placement and interpolation is predefined and thus the only source of incoherence is the lighting process itself.

3.3 LightSkin

LightSkin [LB13] by Lensing et al. uses precomputed light cache positions which are placed directly on surfaces (see Figure 3.5). This has several advantages over radiance volume approaches. Each cache needs to evaluate the lighting only for a single given normal and set of material properties. Neighbor-relationships are precomputed which



(a) Cache distribution. **(b)** Lighting results.

Figure 3.5: [Len14] LightSkin global illumination approach.

allows for efficient real-time interpolation of both indirect diffuse and glossy reflections. A smooth indirect shadow approximation is achieved by pre-computing a size measure for the light caches which makes it possible to use them as coarse blockers. The combination of clever cache placement during the pre-computation and the sophisticated interpolation at runtime makes it possible to achieve a relatively high lighting quality even for low amounts of caches, without any temporal artifacts.

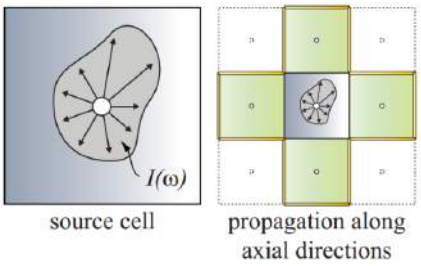
While an expensive pre-computation step is needed to place the light caches, compute their neighbors and determine their properties, the approach is not limited to static scenes as caches can be moved with animated objects. Depending on the number of caches, only smooth lighting can be performed and the technique does not scale well with large scenes since the number of caches for distant objects can not be reduced trivially. Neighborship-relations are either saved per vertex or on a texture. Both methods can increase the memory footprint of a scene considerably.

Furthermore, Lensing describes in his PhD thesis [Len14] how to use LightSkin for subsurface scattering effects and proposes applications in mixed reality scenarios.

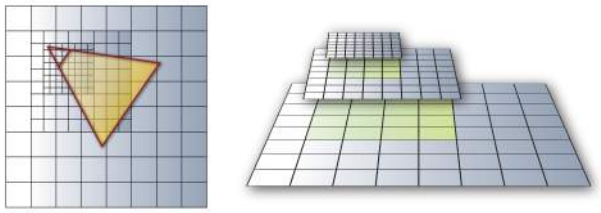
3.4 Discrete Ordinate Methods

Discrete ordinate methods compute light transport by exchanging energy between a cellular discretization of the scene. Usually it is possible to update the lighting iteratively across many frames. As long as there are no high frequency changes, slow adaption can exploit frame coherencies to benefit the overall performance drastically.

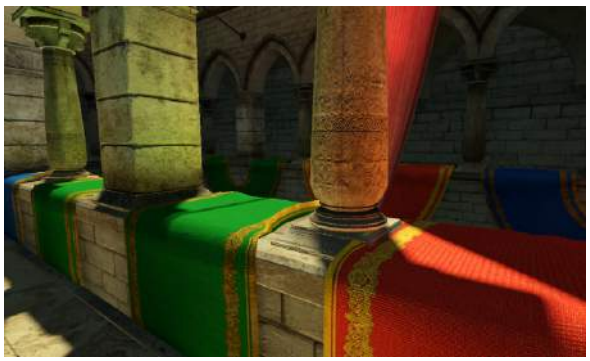
A popular real-time representative of these methods is Cascaded Light Propagation Volumes by Kaplanyan and Dachsbacher [KD10]. Using a RSM, light is injected into a light propagation volume (LPV). The LPV is a regular grid wherein each cell contains a light intensity distribution, represented by a few low order spherical harmonic coefficients. A second grid stores a volumetric representation used for fuzzy occlusion. Considering these blockers, the light is then propagated across the LPV in each frame (see Figure 3.6a). For better scalability with larger scenes, multiple nested LPVs are used, as visualized in Figure 3.6b. After the propagation, the LPVs can easily be looked-up to light the scene.



(a) Light propagation principle.



(b) Cascading of multiple LPVs.



(c) Low frequency indirect lighting.

(d) Lit participating media.

Figure 3.6: [KD10] Cascaded LPV, principles and results.

The approach does not rely on any pre-computation and is able to capture diffuse and very low frequency specular lighting in real-time (see Figure 3.6c). However, it is very prone to light bleeding artifacts and, depending on the LPV resolution, light may only be transported for short distances.
The original approach obtained an approximate blocker volume from the already existing information in the RMS and the camera view. Rasterized Voxel-based GI by Doghramachi [Dog13] presents an alternative real-time voxelization approach with several implementation improvements.

3.5 Voxel-Volume Methods

Due to the increasing processing power of GPUs and the development of real-time voxelization algorithms, methods using ray or cone tracing within a voxelized scene representation have gained popularity in the last few years. Voxelized representations have several advantages over classic triangle-based scenes: The level of detail can be controlled trivially, random access using world coordinates can easily be achieved, intersection tests are simple to perform etc.. The simplest data structures contain only a single binary value per voxel which determines whether the cell is solid or not (binary voxelization). However, other informations like flux, normals or continuous opacity are sometimes encoded as well.

3.5.1 Voxelization via Rasterization

The GPUs rasterization capabilities can be used for extremly fast voxelization. There are boundary voxelizations that encode only the surfaces of an object and solid voxelizations which also capture the interiors of a model as well. Solid voxelization is generally more difficult to achieve since more write operations are needed and the interior of an object might not always be clearly defined.

A general problem with GPU rasterization is the lack of conservativity. For voxelization it is usually needed that all voxels that are touched by a triangle are marked. This is not even the case for standard 2D rasterization. To overcome this issue [PF05][Chapter 42] by Hasselgren et al. propose to enlarge triangles and remove superfluous pixels by discarding them in the fragment shader. Very recent GPUs expose new rasterization modes to accomplish conservative rasterization more quickly in hardware [Cho15].

One of the first algorithms using the GPUs rasterization pipeline was invented by Fang and Chen [FC00] which needed to draw the entire scene once per volume slice. A later approach by Eisemann and Décoret [ED06] achieves binary voxelization in a single pass. Since it writes to one or more 2D targets, it achieves a limited bit depth and can not write directly to a volume texture. The algorithm was later extended to support solid geometry as well [ED08]. [DCB+04] presented a similar approach that handles geometry that lies parallel to the viewing direction with less errors.

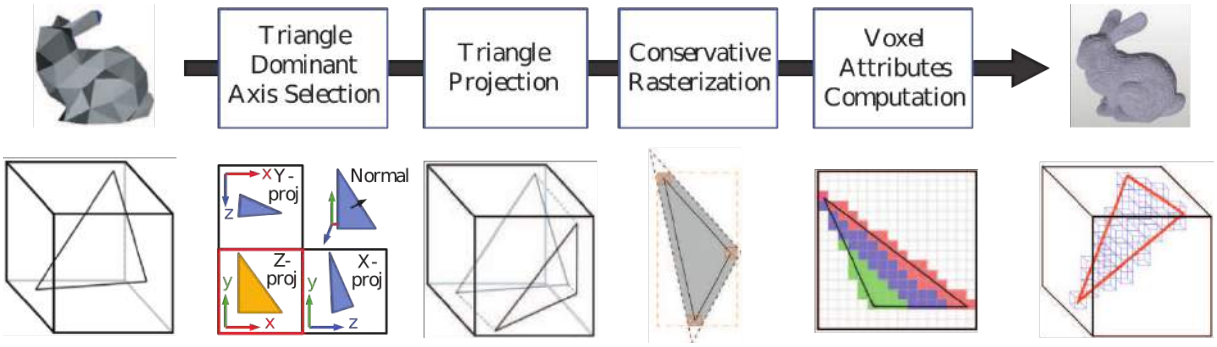


Figure 3.7: [CG12] Illustration of single pass voxelization pipeline.

Many of older methods relied on graphics hardware which was not able to perform random write accesses. Newer methods can leverage these possibilities and are able to perform high quality boundary voxelization in a single pass. In this thesis we rely on a conservative single pass voxelization technique by Crassin and Green [CG12]. Figure 3.7 illustrates how it works. Contrary to [ED06] it is able to write into a volume texture instead of writing bits in a set of 2D targets. The geometry shader decides on the dominant direction of each triangle and uses it as projection axis to maximize the output of the rasterization stage. Conservative rasterization is achieved with the earlier mentioned technique by Hasselgren et al. [PF05][Chapter 42]. Each pixel writes up to three values along the projection axis into the volume texture to mark all voxels that are touched by the source triangle. If necessary additional attributes can be computed in the fragment shader.

3.5.2 Voxel Cone/Ray Tracing

Voxel-Based GI by Thiedemann et al. [THGM11] traces multiple rays at each pixel's world position through a binary voxelization. Whenever a ray hits a surface, it performs a shadow mapping test and, if succeeded, adds light (radiance) to its originating pixel using flux from a reflective shadow map. Since tracing rays through the voxel-volume is a very costly operation, indirect lighting is only possible over short distances.

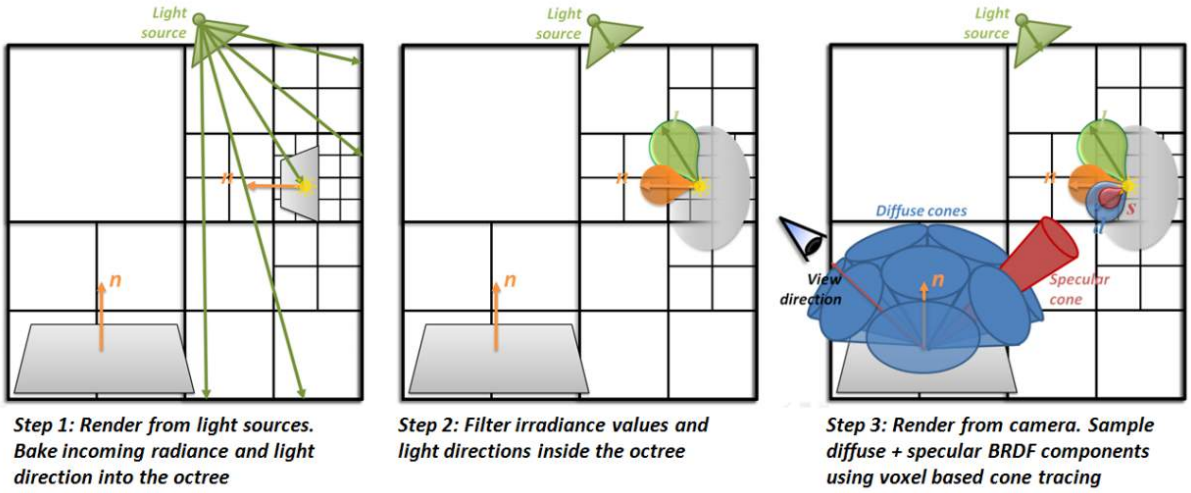


Figure 3.8: [CNS+11] Illustration of Voxel Cone Tracing.

Voxel Cone Tracing by Crassin et al. [CNS+11] overcomes this issue by tracing cones instead of rays through a mip-mapped voxel volume. In addition to the occlusion value, each voxel also saves flux, which was previously injected by an RSM, and a normal. Figure 3.8 gives a short overview over the approach. In each visible point a few cones covering the local hemisphere are traced to achieve diffuse lighting. During the tracing of a cone, the step size and the sampled volume-mipmap-level increases, so that the samples cover approximately the entire cone. This technique allows Voxel Cone Tracing to trace very long cones at relatively low costs. Since occlusion is fuzzy in all but the lowest mipmap-level, lighting is applied continuously with each sample depending on its opacity. The tracing stops when the cone is completely blocked using an emission-absorption model. Additionally, each voxel saves its flux anisotropically for more accurate lighting in higher mip-levels. Using an extra, thinner cone for the specular highlight the approach achieves relatively accurate reflections within the same framework. The main draw-back of Voxel Cone Tracing is its high memory footprint. Because of this the original work makes use of a sparse voxel octree. However, for practical considerations the technique has been used in production with cascaded volumes instead [McL14].

Layered Reflective Shadow Maps by Sugihara et al. [SRS14] works very similar but decouples occlusion from lighting data. Visibility determination is handled via cone tracing as before, while the actual lighting is performed by lookups in a pre-filtered Layered Reflective Shadow Map. The RSM is split up into multiple layers to be able to create pre-filterable textures without large depth or normal discontinuities. Doing so avoids most of the memory and initialization/update overhead associated with the large voxel data structure, since each voxel encodes only binary visibility information. While the approach has a significantly lower memory footprint than the original Voxel Cone Tracing, it suffers

from temporal coherency issues for highly gloss surfaces and does not scale well with the number of light sources.

3.6 Summary

There are only few techniques that fulfill the goals of this thesis (Section 1.3). Most notably Voxel Cone Tracing [THGM11] (VCT) and Cascaded Light Propagation Volumes [KD10] (LPV) are able to perform global illumination without any pre-computations. Both have already been used in production environments. VCT achieves very high quality rendering at to the price of large complicated memory structures (sparse voxel octree on GPU) and, compared to LPV, at much lower frame-rates. LPV on the other hand is often not able to catch indirect lighting over longer distances and is very prone to light bleeding artifacts. Another completely dynamic technique is the radiance volume by Vardis et al. [VPG14]. Like LPV it is able to capture diffuse lighting but has less limitations on light transport distance.

All three techniques rely on reflective shadow mapping which is fundamental to many real-time lighting approaches. On the other hand, direct usage of RSMs often suffers from temporal artifacts as seen in many previous works. Caching light at specific locations leverages the low frequency of most indirect lighting situations and can reduce shading costs tremendously. Voxelized scene representations are used in many recent works since real-time voxelization is by now cheaply available.

This thesis is mostly inspired by LightSkin [LB13] which is able to robustly compute both diffuse and indirect lighting at a relatively high quality. The final algorithm presented in the next chapter is similar to the recent radiance volume method of Vardis et al. [VPG14] of which we were not aware until shortly before the end of our project. However, there are several key differences that will be elaborated in Section 5.6.

4 Main

4.1 Overview

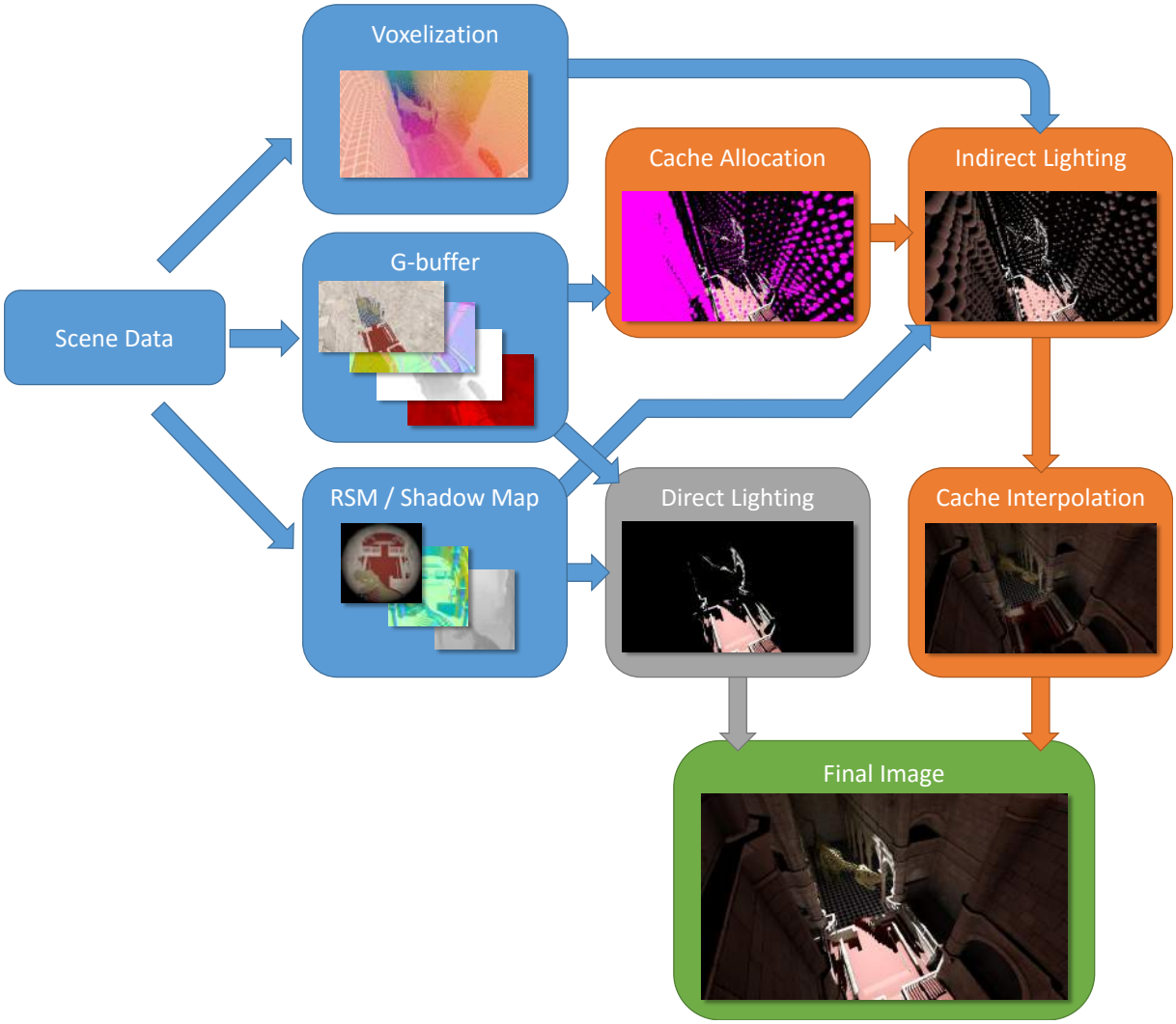


Figure 4.1: Overview over our basic rendering pipeline. The three steps of our technique are the orange boxes.

Inspired by Lensing's LightSkin approach [LB13], we want to compute indirect lighting only at specific locations and interpolate the results over multiple pixels, as opposed to techniques like reflective shadow maps or voxel cone tracing. Like other previous works, we call these locations *Light Caches*.
Our technique consists of three basic steps which are executed in every frame: Cache allocation, indirect lighting and cache interpolation. Figure 4.1 gives an overview of the

pipeline steps and their dependencies. The cache allocation pass assigns cells on a regular grid to cache memory, thus creating the caches that are about to be used in this frame. Using a reflective shadow map, indirect lighting is then computed for all caches. Finally each pixel interpolates an indirect lighting value using all neighboring caches within the grid.

Each of the following major sections will explain one of these stages and their interdependencies. The last major section of this chapter will elaborate several implementation details, some of them crucial to the overall performance.
Many parts of our approach are based on several well-known techniques of which most have already been discussed in Chapter 2 and Chapter 3. Where necessary, details of the respective algorithms will be elaborated to address specifics implied by the overall technique.

4.2 Cache Allocation

Since the goal of this work is a fully dynamic solution that works without any precomputations, it is necessary to place all caches at runtime, as opposed to LightSkin [LB13] where caches are placed in a computational intense preprocessing step. Instead, we use a grid based approach similar to Vardis et al. [VPG14]. Before we discuss the details of our solution, several general properties and implications of such an approach are elaborated.

4.2.1 General Properties of Dynamic Cache Placement

Compared to a pre-computation based approach, there are several intrinsic advantages which can be expected of dynamic light cache placement. Since we try to achieve only a single bounce of indirect light, caches are only used by locations that lie in the current view frustum. It should be possible to decrease the number of caches for distant objects, thus keeping the screen-space cache density low even for high viewing ranges.
While it can be very beneficial to place caches freely, only depending on view space information may introduce several temporal artifacts if caches disappear or move. Additionally, it is necessary to guarantee that target pixels have easy access to a certain number of caches to be able to interpolate smoothly between them. Note that the interpolation pass can have temporal coherence problems on its own, as the assignment of a cache to its surrounding world positions needs to be stable, no matter how densely they are sampled, i.e. how many pixels a given world space area covers.

Precomputed caches can naturally hold almost arbitrary data. Dynamic caches on the other hand need to acquire their data every frame, which is not only costly performance-wise but may also cancel out certain types of data as it may not be accessible. For example LightSkin [LB13] needs a rather complex area metric for each cache that cannot be computed in real-time. Naturally, such constraints also affect the following lighting and interpolation stages tremendously.

We evaluated several approaches before we came up with our final algorithm. A summary of these attempts can be found in Section A..

4.2.2 Cache Address Volume

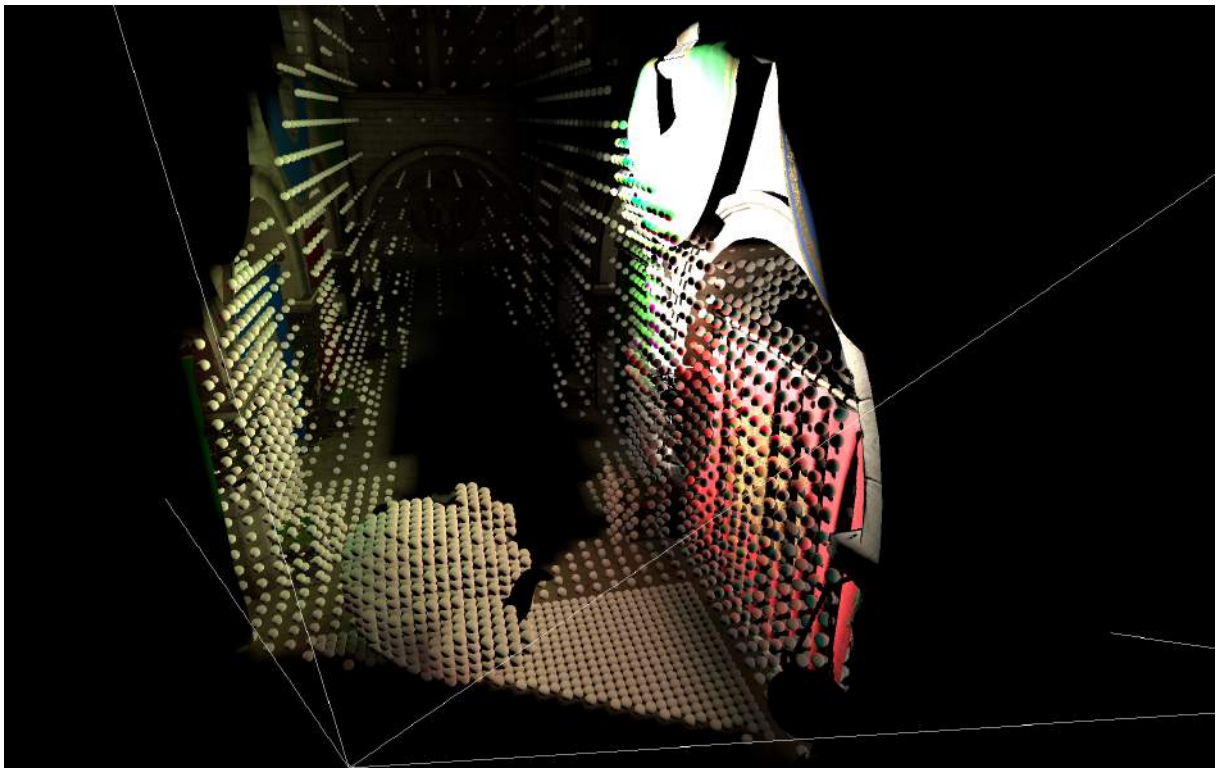


Figure 4.2: Visualization of active caches for a view hinted by the white lines. Note that there is a "shadow" behind the object in the middle where no caches are active since the camera cannot see behind it.

To cancel out temporal artifacts caused by moving light caches, we decided to place caches only at the nodes of a regular world space grid. The main disadvantage of this method is that caches are not placed on surfaces and thus cannot have a specific orientation. Since only a small fraction of all grid cells is actually needed at any time (i.e. lies near to geometry), their data is stored in a separate *cache buffer* and only addresses to this buffer are saved within the grid. Accordingly, we call this grid the *cache address volume (CAV)*. This approach reduces not only the memory consumption drastically, it makes it also much easier to perform the lighting for each cache, since those reside consecutively in a special memory block instead of being scattered within a volume as in Vardis et al. [VPG14].

During the cache allocation pass the CAV is cleared and then refilled depending on the current view. This is done by a full-screen pass, using the per-pixel world-positions from the previously rendered scene. Each pixel allocates caches at the eight nearest CAV cells by reserving addresses and writing them into the CAV. If however an affected cell already contains a valid cache address, it is skipped. As pixels do not introduce any data besides the cache allocation itself, no additional writes are needed. Since neighboring pixels often access the same grid cells, there is room for some effective optimizations using shared memory which are explained in Subsection 4.5.3. Figure 4.2 shows a resulting cache distribution already with CAV cascading which is explained in the next section.

Cascading

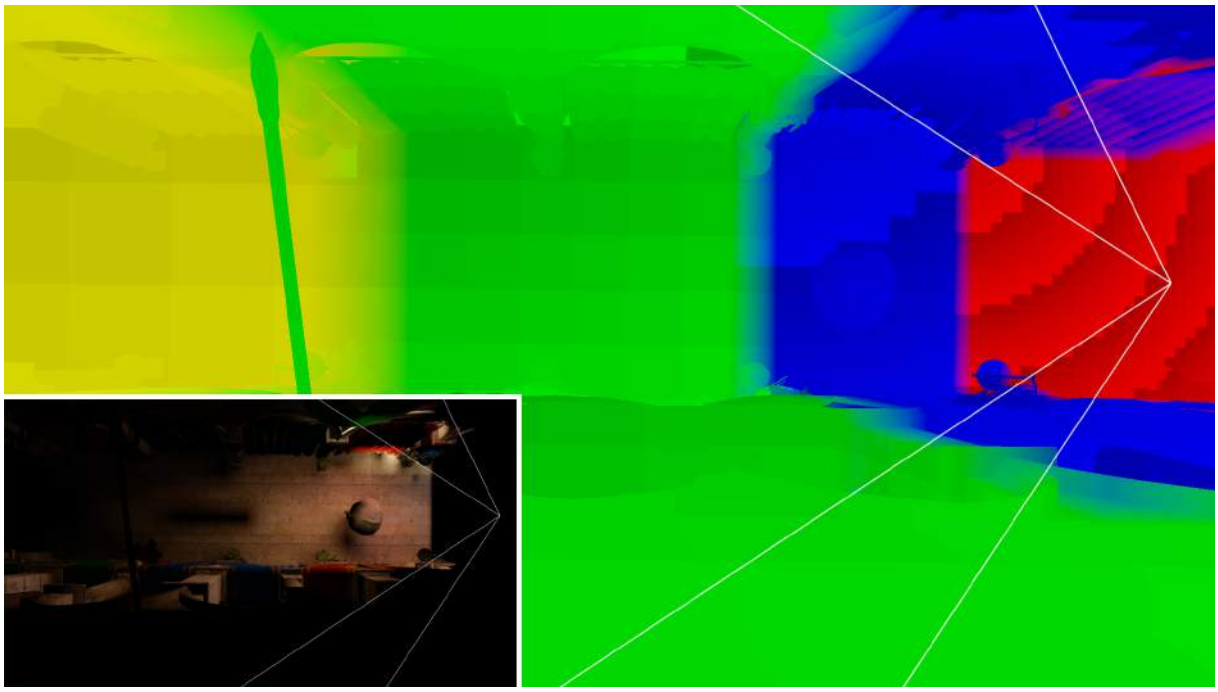


Figure 4.3: Visualization of CAV cascades. Again, the white lines to the right show the active view frustum. Areas with the same brightness within a cascade (single colored area) indicate the local cache size.

Similar to Cascaded Light Propagation Volumes [KD10] a set of nested CAV cascades is centered on the camera. Figure 4.3 visualizes the cascading system. This introduces a distance-dependent level of detail and keeps the number of caches at a reasonable level, even for large scenes. The cascade positions are snapped to multiples of their grid size to ensure consistent lighting results. For simplicity, all CAV cascades are cubic and have the same resolution, while their grid size is doubled for each consecutive cascade.

Caches are only allocated in the smallest CAV cascade which encloses a given pixel world-position. To be able to allocate all eight nearest CAV cells in a single cascade, special "decision boundaries" which are smaller than the actual cascade are used for cascade selection. As the snapped movement introduces temporal incoherency, we move the decision boundaries smoothly with the camera and shrink them further by another grid cell size to ensure that they are fully enclosed by their respective cascades.
Additionally, there are transition areas between two successive cascades. Within these areas, caches are allocated in both cascades to be able to avoid hard edges during the interpolation stage later on. The size of the transition areas can be chosen freely. For our demos we use transition areas as wide as the cell size of the next bigger cascade. This delivers sufficiently smooth transitions for reasonable performance costs in our scenes.

Contrary to Light Propagation Volumes, the cascades are not offset depending on the camera direction. Instead they are strictly centered on the camera position as this helps to keep the level of detail stable for view rotations.
While this means that a very large portion of each cascade is never used at all, the costs

are still rather low since there are no computations involved for empty cells. Also, the memory footprint is still rather small: A highly detailed setup consists of four cascades, each with a resolution of 64^3. With a four byte address in each cell, this makes only four megabytes for all CAV cascades together.

4.3 Indirect Lighting

The indirect lighting of the caches relies on reflective shadow maps (see Subsection 3.1.1). In principle each cache iterates over the entire RSM to retrieve the encoded lighting information. This section explains how this information is processed and how the results are saved for later interpolation.

4.3.1 RSM Preparation

We render the RSM in a relatively high resolution (usually 1024^2) and sample it down for the upcoming indirect lighting steps (usually from 16^2 up to 128^2). In the process depth values and normals are averaged, while flux values are summed to preserve energy. For geometry with many depth discontinuities, this improves the temporal coherence under movement of both objects or lights. This approach may result in incorrect virtual lights, floating in midair with normals vectors that are not present in the original RSM samples. However, we have not encountered any visible artifacts resulting from this filtering technique in our tests.
The direct lighting process uses the original depth from the RSM as shadow map. In the following sections the term RSM denotes the already down-sampled texture set.

4.3.2 Diffuse Lighting

We separate diffuse and specular lighting, making the latter an optional feature that may be deactivated to save resources if necessary. Diffuse lighting is comparatively easy to represent and compute since it is (for a given position and reflectivity) a low frequency function over a surface normal, independent of the current view direction.

Recall the definition for lighting with M virtual lights (see Section 3.1):

$$L(\mathrm{x}, \omega_o) = \sum_{i=1}^{M} f_r(\mathrm{x}, \overrightarrow{\mathrm{xx}_i}, \omega_o) \cdot G(\mathrm{x}, \mathrm{x}_i) \cdot \hat{V}(\mathrm{x}, \mathrm{x}_i) \cdot I_i(\overrightarrow{\mathrm{x}_i\mathrm{x}}) \tag{4.1}$$

We use the geometry term G for disc shaped area lights by Lensing et al. [LB13] as mentioned in Subsection 3.1.2. The visibility function $\hat{V}$ is ignored for now, it is handled later on in Subsection 4.3.4. Since we are only interested in diffuse lighting, the BRDF f_r and the VAL's intensity curves I_i are rather simple:

$$f_r(\mathrm{x}, \overrightarrow{\mathrm{xx}_i}, \omega_o) = \frac{\rho_d}{\pi} \tag{4.2}$$

$$I_i(\overrightarrow{\mathrm{x}_i\mathrm{x}}) = \frac{\rho_i}{\pi}\phi_i \tag{4.3}$$

The final expression for diffuse indirect lighting is defined as follows:

$$L_d(\mathrm{x}) = \frac{\rho_d}{\pi} \sum_{i=1}^{M} (\hat{\mathbf{n}} \cdot \overrightarrow{\mathrm{xx}_i})^+ \cdot \underbrace{\frac{(\hat{\mathbf{n}}_i \cdot \overrightarrow{\mathrm{x}_i\mathrm{x}})^+}{||\mathrm{x} - \mathrm{x}_i||^2 + A_i} \cdot \frac{\rho_i}{\pi} \phi_i}_{L_i} \tag{4.4}$$

The diffuse reflectivity ρ_d at the point x is factored out - it can be applied easily during the cache interpolation stage. Like the position of a virtual area light, its area A_i can be computed from the depth values saved in the RSM. Lensing et al. [LB13] use the following surface independent estimation for a virtual light i at the distance z_i from the source light (which is stored in the RSM):

$$A_i = z_i^2 \cdot \frac{A_{near}}{z_{near}^2 M} \tag{4.5}$$

We assume a quadratic RSM with a total of M pixels with a near plane area of A_{near} at a distance of z_{near}.

Our cache grid effectively quantifies the range of positions x in which the sum over all virtual lights is evaluated. Since caches are not placed on surfaces, the normal vector $\hat{\mathbf{n}}$ is not known in advance. Therefore, we save the incoming radiance L_i as a function over the sphere in each cache to integrate it later on with a cosine lobe, i.e. weighting every direction with the dot product between the surface normal and the direction. As in many previous works, we choose a spherical harmonics representation with either two or three bands to approximate this function. The SH representation allow iterative computation and easy evaluation.

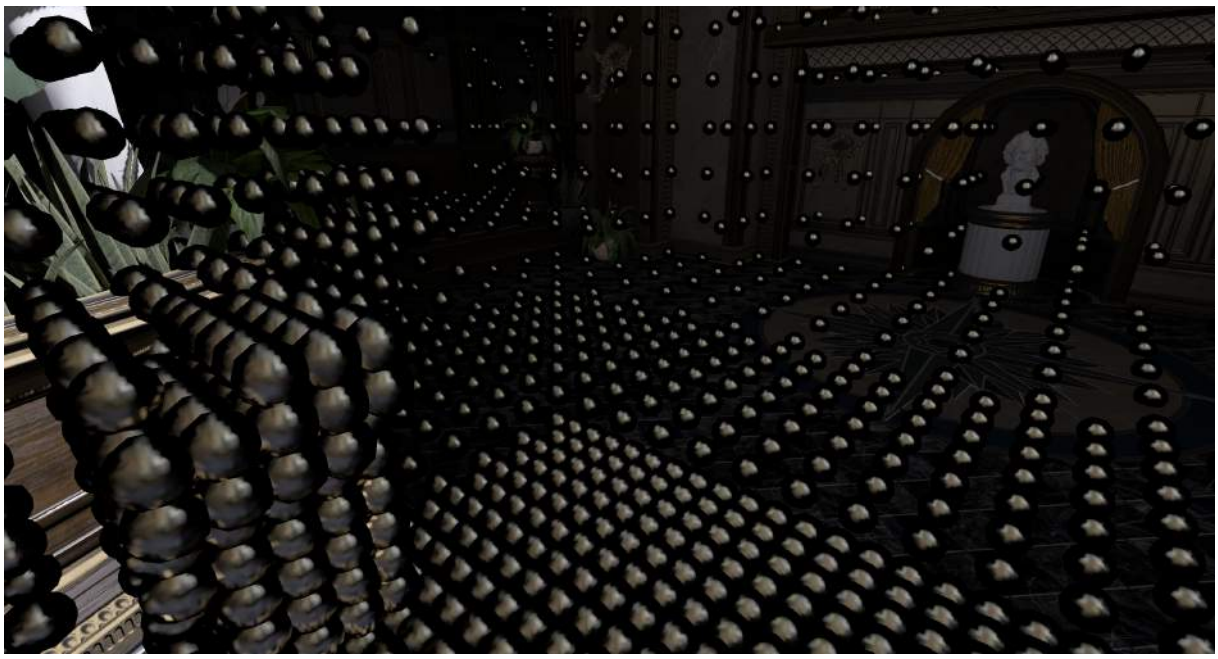
To perform the diffuse cache lighting, the SH coefficients for all virtual lights are accumulated and saved into the cache buffer for later evaluation. The impact on performance and quality of varying number of SH bands will be discussed in Subsection 5.3.3 and Subsection 5.4.3.

4.3.3 Specular Lighting

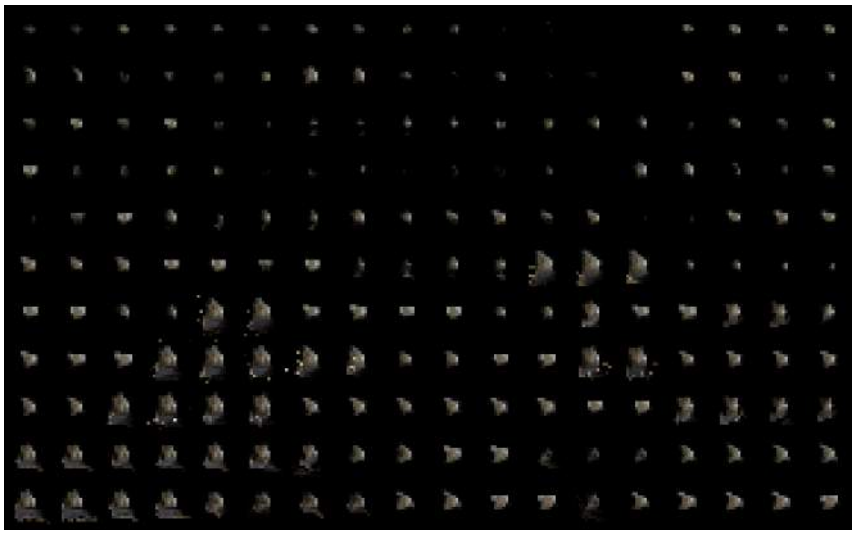
As mentioned in Subsection 2.1.4 we make use of a normalized version of the Blinn-Phong BRDF. Using virtual point lights from the reflective shadow map, the specular part of the visible radiance is defined as follows:

$$L_s(\mathrm{x}) = \rho_s \cdot \frac{\gamma + 8}{8\pi} \cdot \sum_{i=1}^{M} (\hat{\mathbf{n}} \cdot \overrightarrow{\mathrm{xx}_i})^+ \cdot L_i \cdot ((\hat{\mathbf{n}} \cdot \hat{\mathbf{h}}_i)^+)^\gamma \tag{4.6}$$

Where $\hat{\mathbf{h}}_i$ is the half vector to the ith light and γ the specular exponent. The incoming radiance L_i from a light was defined in Equation 4.4.
For moderately high γ the outgoing specular radiance is, contrary to outgoing diffuse radiance, a high frequency function over $\hat{\mathbf{n}}$. Using the same representation as in the previous section on diffuse lighting would require to evaluate much more SH bands (e.g. as in radiance caching [KGPB05]) which are difficult to compute and prone to ringing artifacts.



(a) Cache visualization displaying indirect specular reflections.



(b) Cut-out of the texture atlas.

Figure 4.4: Visualizations of the specular environment map, using 16^2 pixels per cache.

If it is assumed that a cache lies directly on a *visible* surface, then the space of possible normal vectors is limited to the hemisphere pointing towards the viewer. We need to record outgoing radiance (again postponing the factors outside of the sum) for every direction on the hemisphere for each possible γ. As the outgoing radiance for lower exponents can be estimated by integrating results for higher γ values, only an upper bound γ_{max} is handled for now. For very large γ_{max}, the $((\hat{\mathbf{n}} \cdot \hat{\mathbf{h}}_i)^+)^{\gamma_{max}}$ term is only bigger than zero if $\hat{\mathbf{n}} \cong \hat{\mathbf{h}}_i$. This means that in the limit each given virtual point light contributes only to a single hemisphere direction.
We discretize the hemisphere into a limited number of directions. Using the large γ_{max} assumption, we apply light only to the nearest discrete direction. Similar to the work of McGuire et al. [MEW+13] on plausible Blinn-Phong cubemaps, we require that the solid angle of the given Blinn-Phong lobe does not exceed the angle between two data

points. This allows to compute a plausible value for γ_{max}, given a number of D uniformly distributed directions.

$$\int_{2\pi sr} (\cos\theta)^{\gamma_{max}} \, \mathrm{d}\omega = \frac{2\pi}{D} \tag{4.7}$$

$$\gamma_{max} = D - 1 \tag{4.8}$$

To be able to efficiently address values per direction on a texture, the hemisphere is projected onto a square. First, all directions are expressed in the local view space, which is a transformation that maps the direction to the camera to $(0, 0, 1)$. For the projection itself it is important that it introduces as little distortion as possible since otherwise the directions are no longer distributed equally. For this purpose we use uniform concentric maps by Shirly and Chiu [SC97]. We call the resulting maps *specular environment maps* and store them in a large texture atlas. The atlas can easily be addressed by the cache address which was already used for the cache buffer. Depending on the quality setting, each cache typically uses a resolution between 8x8 and 32x32 pixels on this atlas. After all caches are lit, mipmaps of the Specular Environment Map atlas are created. Similar to classical reflection maps [AMHH08, p. 308] they will later be used to estimate the specular lighting for Blinn-Phong exponents below γ_{max}.
Figure 4.4 shows visualizations of the specular environment map. We found that the method has some weak points that will be later discussed in Subsection 5.3.6 and Subsection 5.4.5.

4.3.4 Indirect Shadows

So far, indirect visibility was ignored. In most cases however, approximate shadows are extremely important to the overall image quality, as seen in Figure 4.5. This chapter presents a voxel cone tracing based approach with several adjustments and new simplifications that fit well in our framework.

Shadow Cone Tracing

To estimate shadowing effectively, we trace cones from the caches to virtual lights within a binary voxelization of the scene. The general approach is very similar to the cone tracing step in Layered Reflective Shadow Maps from Sugihara et al. [SRS14]. By averaging the occlusion percentage, mipmaps are created from the initial binary data (where the occlusion percentage is either zero or one).

Higher mip-levels tend to have too low occlusion values since we voxelize only surfaces. Instead, a solid voxelization would be required. However, such solid voxelizations are not only computational more intense, they also require a clear cut definition of the outer and inner volume which is not given by most triangles meshes.

Cone Trace LOD

Contrary to both Sugihara's layered reflective shadow maps and Crassin's voxel cone tracing we do not use the cone tracing to look up a pre-filtered lighting value. Instead, the

Figure 4.5: A scene using our indirect diffuse lighting without (top) and with (bottom) indirect shadows.

tracing is only used to determine if a given virtual light is visible from the position of a cache. We found, that tracing a cone to every virtual light is, depending on the RSM resolution, usually too expensive. However, most virtual lights lie very close in space and do not need to be tested separately. To test multiple VALs at once we need an estimate of their spatial distribution. Inspired by Variance Shadow Maps from Donnelly and Lauritzen [DL06], we save not only depth but also squared depth in the reflective shadow map and pre-filter the combined depth texture by creating mipmaps. As in Variance Shadow Maps, the squared depth will be used to retrieve the variance of the depth value. We introduce the *indirect shadow LOD* as a global constant parameter that determines for which mip-level cones are traced to check the visibility of the underlying group of virtual

lights (0: for each VAL, 1: every four, 2: every 16, ...). All other lighting computations will still work on the lowest mip-level but use the previously computed shadowing value which stays the same across multiple lights.

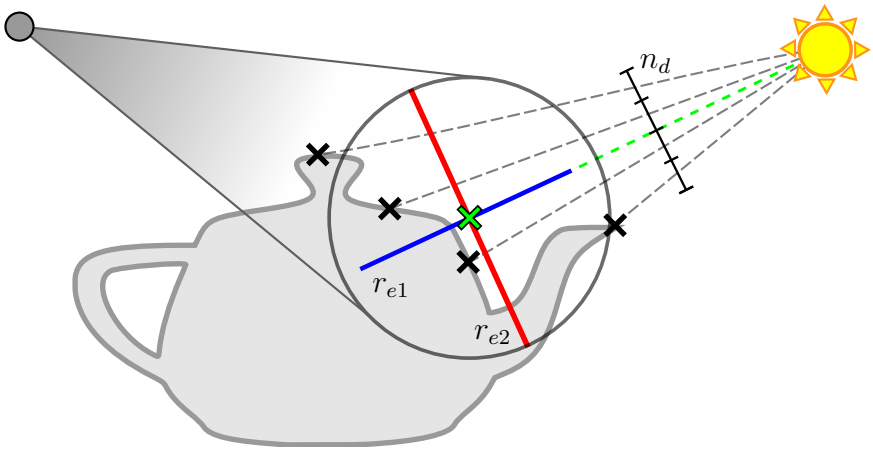


Figure 4.6: Shadow cone tracing from a cache (grey circle left) to a group of virtual lights (black crosses). The opening angle of a shadow cone depends on the sphere around the virtual lights which is the maximum of the projected sample width (red) and the depth variance (blue).

The pre-filtered depth texture is now used to estimate the angle of a cone that encloses a group of virtual lights. To do so, a sphere around the concerning virtual lights needs to be estimated. Figure 4.6 illustrates the algorithm. The center of the sphere (green cross in Figure 4.6) is determined directly by the shadow sample position and its depth which is the average of all underlying virtual lights. For the radius the maximum of two separate estimations r_{e1} and r_{e2} is used.
The first estimate r_{e1} is based on an distribution estimate of light depth values: If it is assumed that the depth values are normally distributed, approximately 95% of all values lie within the range of two times their standard deviation (blue line in Figure 4.6).

$$E(d) = \frac{1}{N}\sum_{i=1}^{N} d_i \tag{4.9}$$

$$E(d^2) = \frac{1}{N}\sum_{i=1}^{N} d_i^2 \tag{4.10}$$

$$\sigma^2 = E(d^2) - E(d) \tag{4.11}$$

$$r_{e1} = \sqrt{E(d^2) - E(d)} \tag{4.12}$$

The average depth $E(d)$ and averaged squared depth $E(d^2)$ of the virtual lights are available through the pre-filtered depth texture.
While the first estimate works well if the virtual lights have very different distances to their origin-light, it underestimates if they are parallel to the near plane of the origin-light. Thus, the second estimate (depicted red in Figure 4.6) is the size of the sample area, projected to the center of the sphere. The sample area is determined by the relative sample size, given by the width w_{near} and depth z_{near} of the near clip plane, the RSM

resolution R and the chosen indirect shadow LOD l.

$$r_{e2} = E(d) \cdot \frac{w_{near}}{z_{near} \cdot R \cdot 2^{-l}} \tag{4.13}$$

Cone Trace Sampling Steps

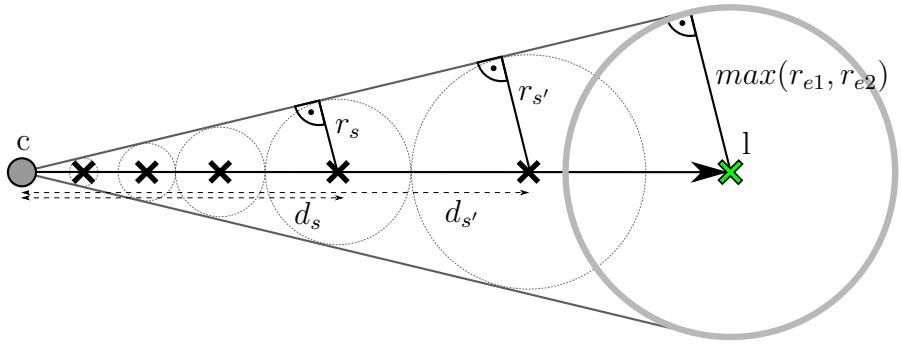


Figure 4.7: Sampling a shadow cone from a cache to a light(-group). The first two enclosing circles do not touch each other because the sample distance would then below the minimum.

The ray from cache to the sphere is sampled in increasing steps. For each sample, we need to determine the mip-level in the voxel-volume of which a single voxel is approximately as large as the cone segment that is represented by the sample. This is estimated by the radius r_s of the inner circle of the current sample, as depicted in Figure 4.7. It can be computed by easy trigonometry:

$$r_s = d_s \cdot \sin\alpha = d_s \cdot \frac{max(r_{e1}, r_{e2})}{||\mathrm{c} - \mathrm{l}||} \tag{4.14}$$

d_s is the distance of the given sample to the cache. c and l are the positions of the cache and light. The cone angle α does not need to be evaluated directly. If all values are given in voxel volume coordinates, the sampled mip-level is simply $log_2(r_s)$.
The distance d_s of the next sample is chosen such that two inner circles touch each other:

$$d_{s'} = d_s + r_s + r_{s'} = d_s + r_s + d_{s'} \cdot \sin\alpha \tag{4.15}$$

$$d_{s'} = \frac{d_s + r_s}{1 - \sin\alpha} = \frac{d_s + r_s}{1 - \frac{r_s}{d_s}} \tag{4.16}$$

To avoid needless oversampling for thin cones, step sizes smaller than one voxel are not allowed (as illustrated by the first two samples in Figure 4.7).

Using a trilinear texture filter, the opacity values α_s from the voxel volume are interpolated within and across mip-levels. The total visibility V is accumulated front to back:

$$V = 1 - \sum_{s=1} \alpha_s \prod_{j=0}^{s-1} (1 - \alpha_j) \tag{4.17}$$

The sampling stops if either the light group position l is reached or the visibility approaches zero.

4.4 Cache Interpolation

After all caches have been populated with lighting data, their information needs to be interpolated across the screen. As in the cache allocation pass, each pixel uses its world position to retrieve the buffer locations of its eight nearest caches in one or two cascades (for transitions).

For the diffuse lighting, the irradiance is evaluated with the per-pixel normal in each cache. To do so, the radiance SH needs to be integrated over the cosine weighted hemisphere. This requires to express the cosine lobe in SH representation and perform the dot product with the irradiance SH coefficients. Since this weight function is radially symmetric to the normal vector, rotated Zonal Harmonic coefficients can be used to derive the expression easily. First, Zonal Harmonic coefficients z_l are computed for a fixed normal direction, pointing to $(0, 0, 1)$.

$$z_l = \int_{2\pi sr} L_i \cdot (\omega \cdot (0,0,1))^+ \cdot y_l^0(\omega)\,\mathrm{d}\omega \tag{4.18}$$

$$= \int_{2\pi sr} L_i \cdot \cos\theta \cdot y_l^0(\omega)\,\mathrm{d}\omega \tag{4.19}$$

Note that by integrating only over the upper hemisphere, the cosine does not need to be explicitly clamped. Due to the rotational invariance of Spherical Harmonics, the resulting Zonal Harmonics coefficients can be rotated to the normal direction (see Subsection 2.4.3) without data loss. The resulting expressions for the first three SH bands can be found in Subsection B.2.

The result is trilinearly interpolated depending on the relative position of the pixel to its eight nearest caches. To retrieve the final visible radiance, the interpolated radiance value is then multiplied with the diffuse reflectance divided by π (see Subsection 2.1.4).

For the specular lighting we need to determine the relative coordinate and, depending on the Blinn-Phong exponent, at which mip-level the specular environment map of each cache should be sampled. Identical to the computation of the specular environment map, finding the coordinate involves the projection of the normal vector using the local view space (see Subsection 4.3.3). Since the direction to the camera varies only little from cache to cache, we use the local view space at the pixel-position for all caches. This introduces a slight inaccuracy but circumvents the possibility that a per-pixel normal vector might lie outside of a cache's local view hemisphere.
The mip-level is computed similarly to γ_{max} as before, only that this time the mip-level m is searched for.

$$\int_{2\pi\ sr} (\cos\theta)^\gamma\,\mathrm{d}\omega = \frac{2\pi}{R \cdot 2^{-m}} \tag{4.20}$$

$$\frac{2\pi}{\gamma+1} = \frac{2\pi}{R \cdot 2^{-m}} \tag{4.21}$$

$$m = \log_2\left(\frac{2 \cdot R^2}{\gamma+1}\right) \cdot \frac{1}{2} \tag{4.22}$$

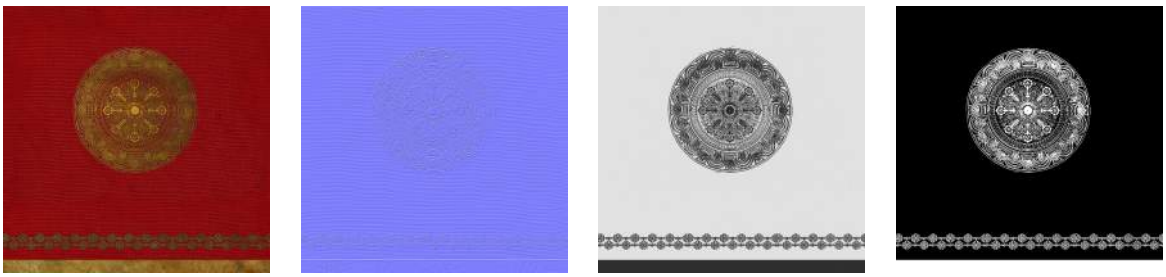
Where R is the resolution of a single specular environment map. As before with the diffuse lighting, the results of all caches are trilinearly interpolated and multiplied with the specular reflectivity of the corresponding pixel.

Pixels in transition areas (see Section 4.2.2) compute two radiance values, one for both CAV cascade. The results are then interpolated linearly.

4.5 Implementation Details

In this section we provide several details of our implementation which is freely available on GitHub (https://github.com/Wumpf/DynamicRadianceVolume). The implementation runs entirely on the GPU and makes use of several OpenGL 4.5 features.

4.5.1 Material Setup and Direct Lighting



(a) Color **(b)** Normalmap **(c)** Roughness **(d)** Metallic

Figure 4.8: Texture setup demonstrated on curtain from sponza scene.

Our material setup resembles what is usually called "metallic setup" in physically based shading (see Figure 4.8). To parameterize the Blinn Phong BRDF, all surfaces have a reflectance texture (sRGB), tangent space normal-maps and two material property textures which are combined into a single texture during loading if not already in the desired format. The first channel contains a roughness value r from which we derive the Blinn-Phong specular exponent γ and the second determines how metallic a surface is: Using this metallic value m, the reflectance c is split up into diffuse ρ_d and specular reflectance ρ_s. The interpretation of all these values happens at runtime using simple heuristics:

$$\gamma = \frac{2}{r^4 + 0.0005} \tag{4.23}$$

$$\rho_d = c \cdot (1 - m) + 0.02 \cdot m \tag{4.24}$$

$$\rho_s = 0.04 \cdot (1 - m) + c \cdot m \tag{4.25}$$

The exponent computation as well as the base diffuse reflectivity for metallic objects are based on cosmetic considerations. A specular reflectivity of 0.04 is the average for most dielectrics [Hof14].

Direct lighting is achieved using a deferred renderer with a floating point depth buffer and three layers: A sRGB texture containing reflectance, a two channel 16 bit signed integer

layer for angle representation of the normal vector and a 8bit two channel texture for the aforementioned roughness and metallic values.

4.5.2 RSM Format

Like our G-Buffer, the RSM consists of three textures and a floating point depth texture. As before we store normals compressed by storing their two angles in signed 16 bit integer textures. We tried to use the low precision floating point format *R11G11B10F* for storing the flux. This however led to severe artifacts if the RSM is downsampled from a texture which is more than three times larger. Instead, a three channel 16bit float format is used. Since we need to downsample depth and squared depth, we added a separate 2 channel 16bit float texture for this purpose. The full float depth texture is only used as depth buffer during rendering and for direct lighting.

4.5.3 Cache Allocation

For simpler addressability all CAV cascades are stored in the same 32bit integer volume texture. Since each cascade is cubic and has the same resolution, they can simply be aligned along the x-coordinate.

The task of the cache allocation pass is to find unique addresses for all needed caches and write them into the CAV. For this, a compute shader with a thread for each screen pixel is dispatched. Each pixel is only allowed to increase the global cache counter if its destination CAV cell has not already been written. This can easily be achieved using the compare and swap atomic operation on the CAV cell and an atomic-add on the global cache counter. The following simplified GLSL code describes how this process works:

```
void AllocateCache(uint cascade, uvec3 cachePosition)
{
  uvec3 CAVcoord = cachePosition;
  CAVcoord.x += cascade * AddressVolumeResolution;

  // Try lock.
  uint oldAddressValue = imageAtomicCompSwap(VoxelAddressVolume, CAVcoord,
                                             uint(0), uint(0xFFFFFFFF));
  // Continue if lock was successful.
  if(oldAddressValue == 0)
  {
    // Increment total cache count and retrieve buffer address.
    uint lightCacheIndex = atomicAdd(TotalLightCacheCount, 1);

    // Compute cache world position and save in global cache buffer.
    LightCacheBuffer[lightCacheIndex].Position = cachePosition *
                                      AddressVolumeCascades[cascade].WorldVoxelSize +
                                      AddressVolumeCascades[cascade].Min;

    // Store address. No need for atomic now, since the cell is kept locked.
    // Light cache index 0 is reserved for "empty".
    imageStore(VoxelAddressVolume, unpackedAddressCoord, lightCacheIndex + 1);
  }
}
```

Note that this is a very expensive operation (due to the atomics on global memory) which usually executed redundantly since neighboring pixels mark the same caches most of the time. To reduce the number of such function calls we first tried to implement a cache in shared memory. However, it turned out that the newly introduced atomic operations on shared memory, used to implement such a cache, were even more expensive which limited the efficiency considerably.
Our final version works with shared memory and a simple heuristic that does not rely on atomic operations:

```
shared uint cacheList[THREAD_GROUP_SIZE][THREAD_GROUP_SIZE];

// ...

// Retrieve integer that represents the cache coordinate and save it to the shared ↩
    memory.
int ownCacheCoord = GetCache1DCoord(pixelWorldPosition, addressVolumeCascade);
cacheList[gl_LocalInvocationID.x][gl_LocalInvocationID.y] = ownCacheCoord;

// Wait for all other threads in this group.
barrier();

// Allocate only if there is no thread with a lower number index on any axis
// with the same cache coordinate.
uvec2 lookUpThread = max(uvec2(0), uvec2(gl_LocalInvocationID.xy) - uvec2(1));
if((cacheList[gl_LocalInvocationID.x][lookUpThread.y] != ownCacheCoord &&
    cacheList[lookUpThread.x][gl_LocalInvocationID.y] != ownCacheCoord &&
    cacheList[lookUpThread.x][lookUpThread.y] != ownCacheCoord) ||
   lookUpThread == gl_LocalInvocationID.xy))
{
  // Unpack cache coord and allocate the nearest 8 caches.
  AllocateCaches(ownCacheCoord);
}
```

Again, the code was simplified for better readability. First, every thread writes the packed coordinate of one of its destination caches into shared memory. Remember that there are actually eight that influence a single pixel. To simplify the process we consider only the one with the lowest coordinate on all axes. After a barrier-sync we check if none of the threads to the left, bottom or left-bottom wrote the same coordinate to the cache list. The most bottom left thread always calls the allocate function - otherwise no cache would be allocated at all if all threads of a group wrote the same address into the shared memory. To handle transitions, we use a second cache list in memory where we apply the same scheme. As an additional optimization, threads perform the lookup in the opposite direction (top/right), so that it is unlikely that the same thread needs to allocate both the "normal" and the "transition-zone" cache.
We achieved the best performance with a thread group size of 16×16.

For a resolution of 2560×1440 and a view with about 16 k caches, our methods is roughly six times faster than the primitive approach.

4.5.4 Voxelization & Shadow Cones

We use the GPU voxelization as presented by Crassin and Green [CG12]. This allows us to recompute the voxelization every frame for a relatively low cost. Since we save only a 8bit occupancy value, our overall memory consumption stays low.

In order to avoid temporal incoherence that arises when objects are moving through the coarse binary voxelization, we add the difference between the current voxelization to the last one gradually. Depending on the (manually set) adaption rate, very fast objects may leave only faint traces in the voxel-grid. In consequence they will cast no or only dim shadows. The perceptually best adaption rate depends on the relative voxel cell size. Coarser volumes need slower adaption, otherwise large shadow features may appear suddenly.

All shadow cones start with an offset which is as large as two voxel cells to reduce the likelihood of accidental self-shadowing.

4.5.5 Cache Lighting

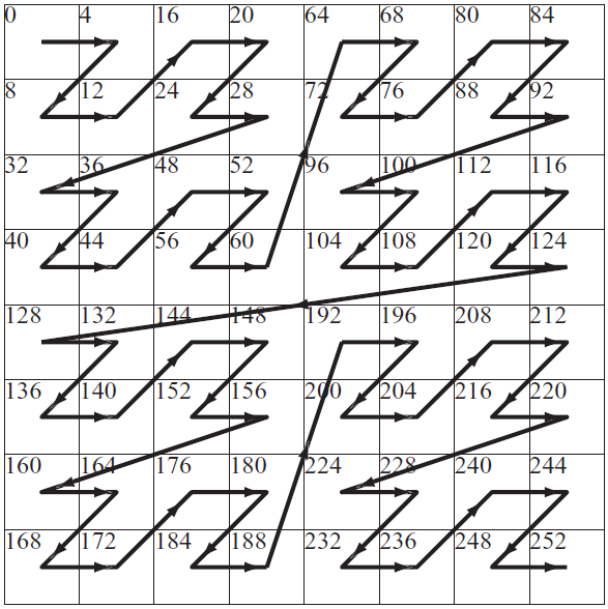


Figure 4.9: [AMHH08, p.848] The Morton sequence pattern.

Our shadow LOD algorithm requires that a shadowing value is valid for multiple neighboring virtual lights. This means that all VALs that share the same shadowing need to be bulk-processed. To achieve this without having to re-implement the iteration on the RSM for every shadow LOD, we use the *Morton sequence* [Mor66] which is visualized in Figure 4.9. Computing the two-dimensional coordinate from a running integer involves several costly integer operations. However, in some cases the lighting process even speed up in our implementation, most likely since the texture resides in the video ram in exactly this or a similar order [AMHH08, p.848]. Note that this more or less forces us to use quadric power of two RSMs since other sizes are more difficult to traverse and introduce exceptions to our shadow sampling scheme.

Since all shader invocations need to load all virtual lights from the RSM, we can easily optimize the texture accesses by first storing lights in the shared memory. In our tests uncompressing the data (decoding normals, computing position from depth etc.) proved to be beneficial although this takes more shared memory. The following code fragment illustrates the process:

```
shared LightInfo RSMCache[THREADS_PER_GROUP];

// ...

for(uint rsmIndex = gl_LocalInvocationID.x; rsmIndex < totalNumRSMPixels; rsmIndex += ↩
    THREADS_PER_GROUP)
{
  // Each thread loads a reflective shadow map texel = VAL.
  LightInfo cacheEntry;

  // Unpack rsmIndex to a actual position.
  ivec2 rsmSamplePos = ivec2(Morton_2D_Decode_16bit(rsmIndex));

  // Sample flux.
  cacheEntry.Flux = texelFetch(RSM_Flux, rsmSamplePos).rgb;
  // Sample depth and compute VAL area.
  float sourceLightToVAL = texelFetch(RSM_DepthLinSq, rsmSamplePos).r;
  cacheEntry.DiscArea = sourceLightToVAL * sourceLightToVAL * ValAreaFactor;
  // Compute world position.
  cacheEntry.Position = ComputeVALPosition(rsmSamplePos, sourceLightToVAL);
  // Sample and unpack normal.
  cacheEntry.Normal = UnpackNormal16I(texelFetch(RSM_Normal, rsmSamplePos).xy);

  // Write into cache.
  barrier(); // Wait for other threads to load lights.
  RSMCache[gl_LocalInvocationID.x] = cacheEntry;
  barrier(); // Make sure all other threads have written their lights.

  // Perform lighting for all cached lights.
  float visibility = 1.0;
  for(int i=0; i<LIGHTING_THREADS_PER_GROUP; ++i)
  {
    if(i % IndirectShadowComputationSampleInterval == 0)
    {
      shadowing = ComputeConeTraceShadow(/* ... */);
    }

    PerformLighting(RSMCache[i].Flux, visibility, /* ... */);
  }
}

// Write lighting results to cache buffer.
// ...
```

Each compute shader thread of a group loads exactly one light into the shared memory, waits for all other threads of its group and then starts to traverse the lights cached in the shared memory. This is repeated until all virtual lights have been processed.

Specular Lighting

For the specular environment map atlas we use the low precision floating point format *R11G11B10F*. There are two different ways to handle writes to this texture.
One way is simply to write values as they are computed. However, since multiple VALs might address the same pixel in the specular environment map, a read has to be performed first to add the radiance instead of overwriting existing information.
Because of this we experimented with a cached write: Each thread holds its portion of the texture in its registers and does not write it to the global texture atlas until all VALs were handled. Since even for low resolutions this creates a very high register pressure, it is cheaper to compress this in-memory map. As it is expensive to de-/compress from/to *R11G11B10F* in software, we use a custom 32 bit shared exponent format for this task. Due to the different highly lossy compressions, both methods yield visually different results. Subsection 5.3.6 discusses which technique performs better.

5 Evaluation

This chapter will evaluate the proposed algorithms both from a performance and a quality perspective. In the last section we try to compare our technique to related works. However, as we were not able to implement comparable implementations within the limits of this project, we can only provide comparisons on an abstract level.

5.1 Test Setup & Scenes

All tests were rendered using a Nvidia GTX670 desktop graphics card. The test machine contains an Intel i7-3770 CPU and uses Windows 8.1 64bit as OS.

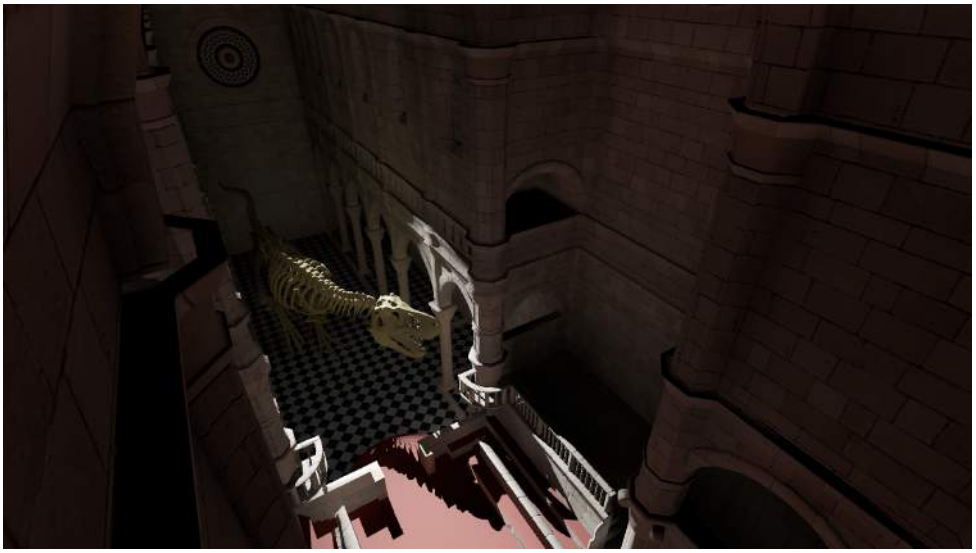
Figure 5.1 gives an overview over the main test scenes used in this chapter (and throughout the thesis). All three scenes have different detail densities and feature different mixtures of material properties.

We use McGuire's [McG11] version of the well known Crytek Sponza scene which was modeled by Frank Meinl. For a better fit into our physically based pipeline we used the freely provided texture set by Alexandre Pestana [Pes15]. Additionally, we placed a continuously rotating Utah teapot with a custom metal-like texture in the scene to showcase specular reflections on a curved surface with varying metallic and roughness values. The setup consists of 278k triangles and a total of 153k vertices. Its intensely contrasted curtain colors are a popular showcase for global illumination effects. We slightly modified the materials to be able to feature glossy reflections.

The Sibenik cathedral scene is from McGuire's database [McG11] as well. To fill the empty space, we added a dinosaur skeleton from the "Natural History Museum" scene modeled by Alvara Luna Bautista and Joel Anderson (available at http://www.3drender.com/challenges/). With only 131k triangles and 98k vertices, this scene is our least complex regular test scenario. While the scene itself is very simple, the dinosaur adds some complexity. Materials are mostly diffuse in this scene.

The Republique scene was extracted from a Unity3D tech demo for the game "République Remastered" by Camouflaj. The entire demo is freely available on the Unity Assetstore [Cam15]. While with 285k triangles and 353k vertices this scene has a raw amount of geometry-data similar to the Sponza scene, it is still the most complex, due its dense vegetation, which comes with many thin features. This leads to comparatively difficult shadowing.

(a) Sponza



(b) Sibenik

(c) Republique

Figure 5.1: Overview over our three main test scenes.

5.2 Parameter Overview

How our approach performs depends on several parameters. To make this chapter more readable and make comparisons easier, we introduce a standard configuration on all parameters. If not noted otherwise we fall back to these default settings. The defaults are annotated in parenthesis in the following parameter overview:

- General
  - Number of active caches
    - Cache address volume resolution (32^3)
    - Cascading settings (4 cascades, last cascade always covers the entire scene)
    - Camera
  - Reflective shadow map resolution (rendering 1024^2, indirect light 64^2)
  - Screen resolution (1920×1080)
  - Number of SH bands for diffuse lighting (2 bands)
  - Number of lights (a single spot light)
- Shadowing
  - Shadow LOD (2)
  - Voxel resolution (128^3)
- Specular
  - Specular environment map resolution (16^2)
  - Direct write or cached write (direct write)

This chapter tries to give insights on the effects of these parameters. We limit our observation to meaningful ranges and do not experiment with arbitrary combinations.

5.3 Performance

The performance and scalability of our approach is examined in this section. To be able to evaluate how specific parts of our algorithm perform, we use multi-buffered OpenGL timer queries to check how long a series of calls take to finish on the GPU (`GL_TIME_ELAPSED`) without stalling it. The sum of timings from multiple commands may exceed the total time a frame takes since the GPU might work on multiple commands at a time. However, in our tests this overlap seems to play a minor role since most tasks are either very computationally intense or depend on each other, which prohibits parallel command execution. In general such measurements can slow down the rendering process. For our scenarios though, we did not observe any drops in the framerate, independently measured with activated timer queries.
Representative images to most of the parameter studies will be presented in the next section.

5.3.1 General Breakdown

First, we want to provide a general overview over how long the different steps of our rendering pipeline usually take. For this, we measure all timings along 30 seconds camera flights through our test scenes using the default settings mentioned in Section 5.1. These paths are shown by the accompanying videos. The first set of graphs in Figure 5.2 shows timings without and the second in Figure 5.3 with indirect specular reflections. The stacked area plots are sorted in the order in which the respective passes are issued, starting with the G-buffer creation (light green) at the bottom and ending with the cache interpolation on top (beige). The black line at the top of all plots shows the total duration for each recorded frame. Note that there is a small white gap below this line which represents the duration of all overheads that were not measured explicitly like tonemapping, several uniform buffer updates and output to the backbuffer.

As can easily be seen, the indirect specular lighting increases the costs of the indirect lighting pass (dark yellow) considerably while the additional mipmapping of the specular environment map is very cheap. Since the light does not move in these tests, the rendering of the reflective shadow map (dark grey) is constant over time. The duration of the voxelization pass and its blending + mipmapping is constant as well since there are no changes in resolution or the geometrical complexity of the scene. More surprisingly, there are also almost no changes over time in the cache allocation pass (bright yellow) even when the cache lighting pass (dark yellow) is significantly slower, which hints a higher number of light caches. Since the main influence to this and the other two cache related passes is the number of active caches, we will investigate this further in the next section on cache count. The strong changes in the duration of the G-buffer creation pass (especially in Sponza) stems from varying overdraw depending on the current view orientation.

In general, the performance in the three scenes does not differ much. Only the significantly more complex Republique scene performs a bit slower, which especially is observable without indirect specular.

5.3.2 Cache Count

Retrieving the current cache count from the GPU is a very costly operation unless extensive multi-buffering is used to avoid stalls (i.e. keeping the count in a different buffer every frame to ensure that reads are only performed on an unused one). Therefore, we measured the cache count separately over time and combined the results in the graphs seen in Figure 5.4.

Surprisingly, there seems to be only a slight correlation between the duration of the cache lighting pass and the number of caches. The duration of the allocate pass is rather stable and only the apply cache pass shows a slight and very unsteady correlation to the cache count. The steadiness of the cache allocation pass most likely comes from the fact that on average every thread group performs a single cache allocation attempt regardless of the cache count. Whether an attempt is redundant might not be as important during runtime. The cache apply pass on the other hand has a strict control flow, i.e. it executes the same code irrespective of the number of caches. This leaves only the changing number of cache misses as reasonable explanation for its behavior.

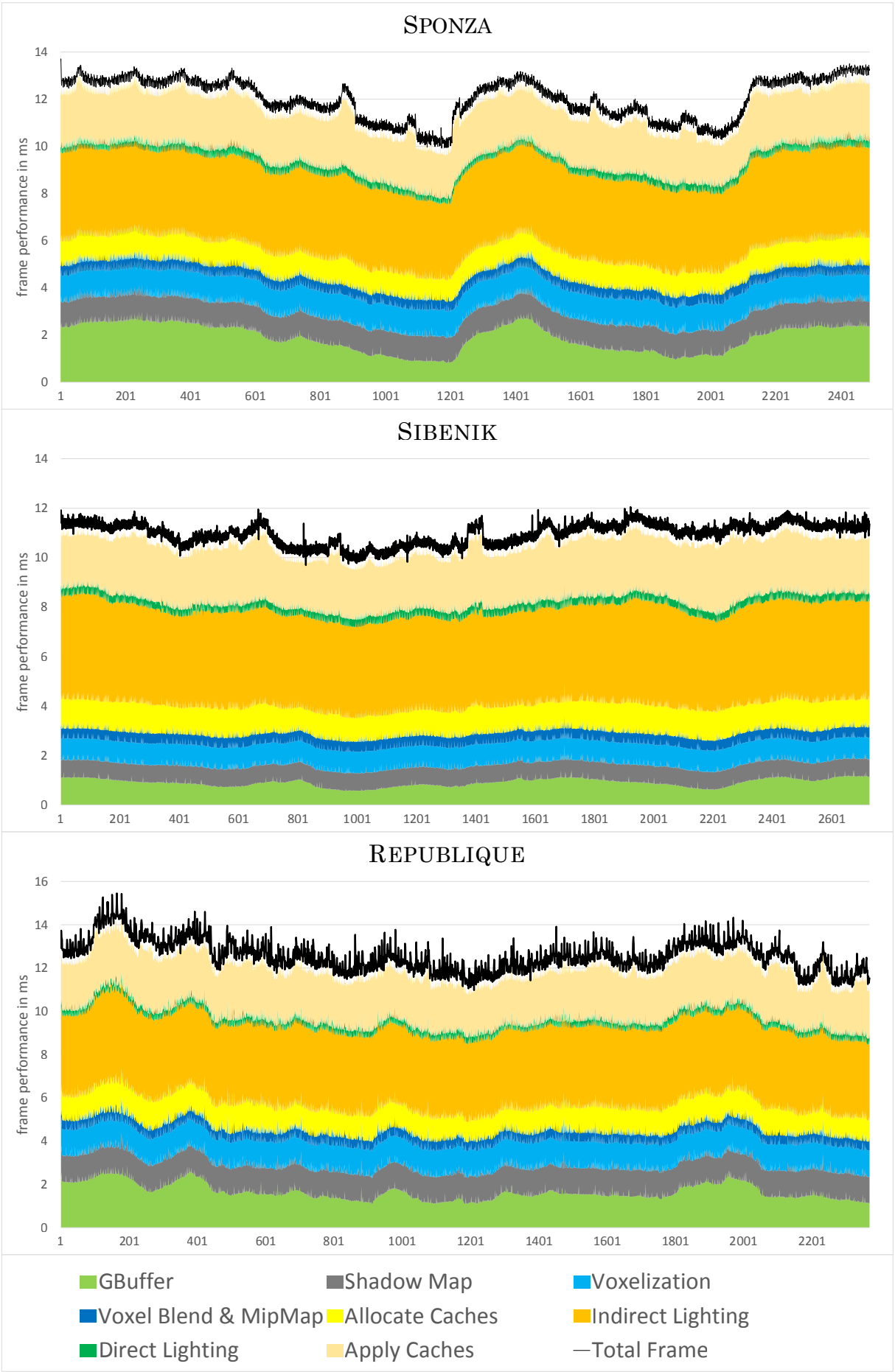


Figure 5.2: Timing breakdown with a moving camera (for 30s) for indirect diffuse lighting. x-axis: frame index, y-axis: stacked duration.

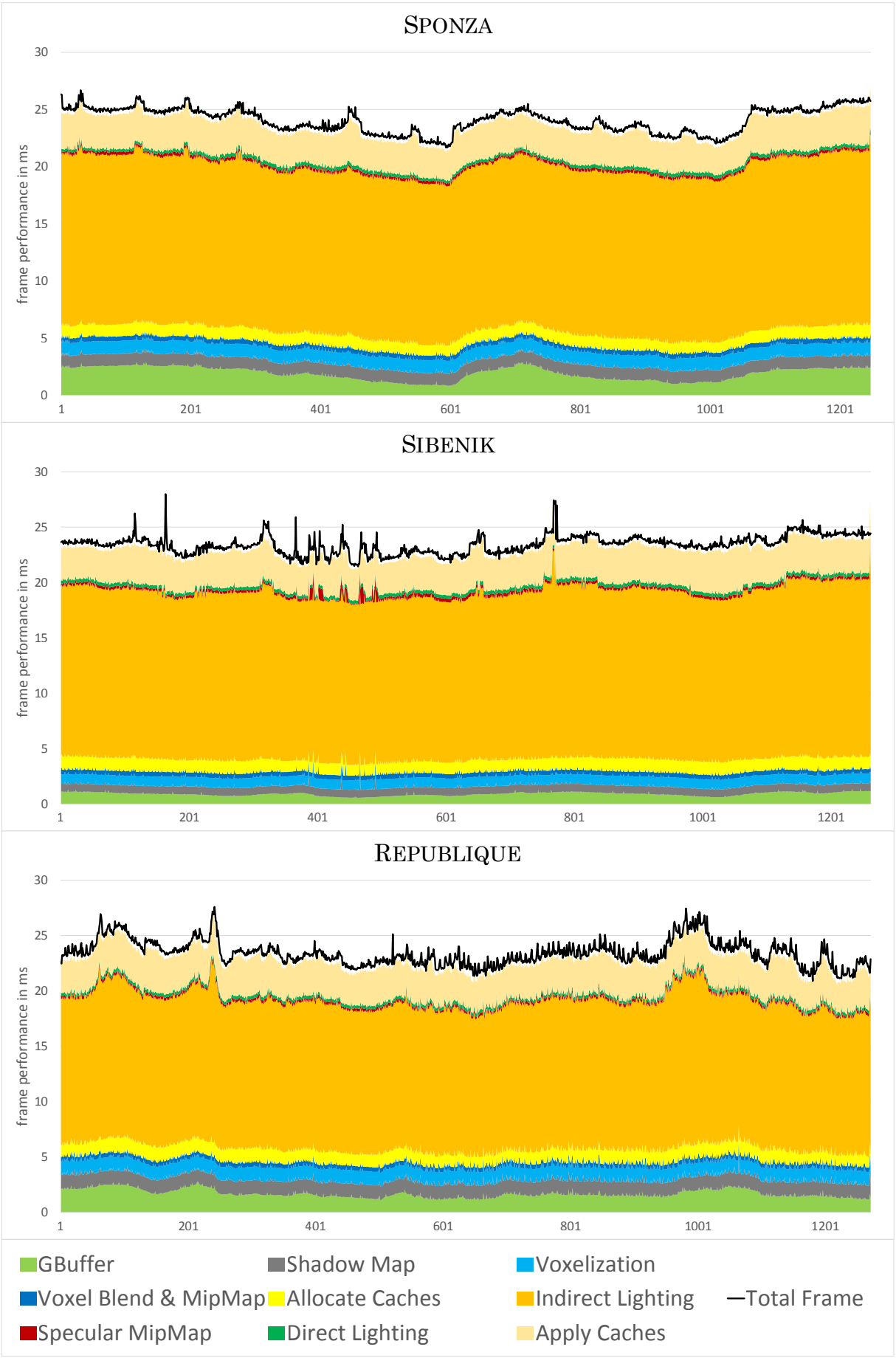


Figure 5.3: Timings with a moving camera (for 30s) for indirect diffuse & *specular* lighting. x-axis: frame index, y-axis: stacked duration.

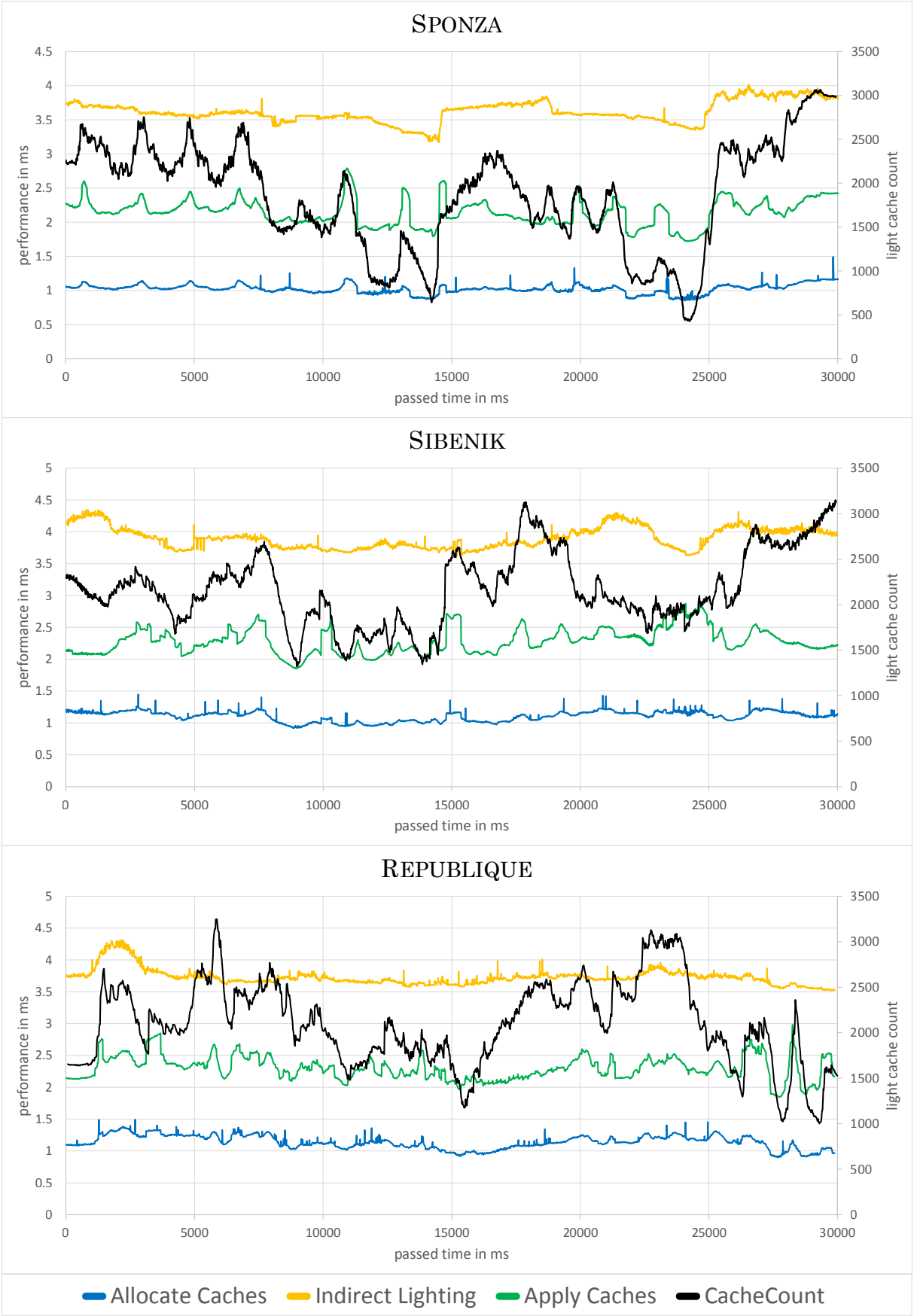


Figure 5.4: Visualization of the influence of cache count (black) to all relevant passes over time (30s), using the same camera flights and settings as in Figure 5.2.

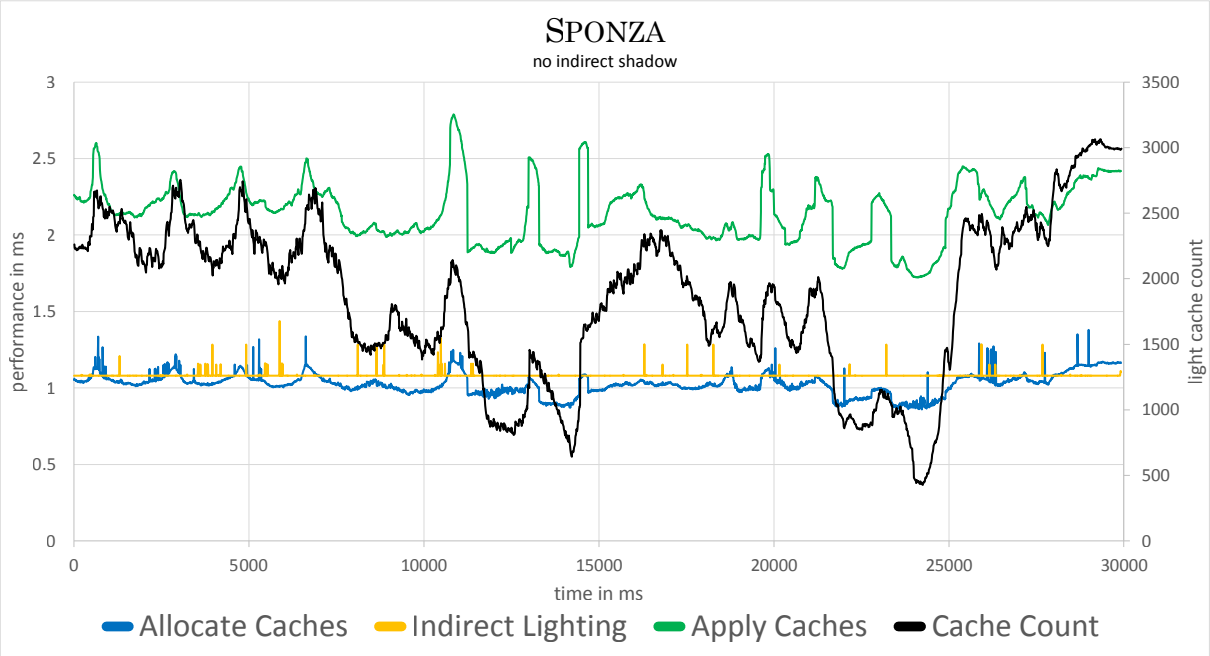


Figure 5.5: Cache count (black) in comparison to indirect lighting passes with inactive indirect shadowing (camera path and other settings as in Figure 5.2).

Clearly, the main bottleneck of the cache lighting pass in the preceding tests is not the raw number of caches to be lit, but another factor that changed along the tested camera paths. To reduce the number of influences we ran the same experiment in the SPONZA scene again with disabled indirect shadow.
The results, seen in Figure 5.5, show an even more extreme picture: Now the duration of the caching pass is almost completely constant, while correlations of the cache count and the performance of the other two parts are much more apparent. Naturally we suspected a bug in the implementation of the light caching pass that is expected to a have linear performance characteristic in respect to the number of active light caches. However, more experiments with higher cache counts show that there is in fact such a linear correlation. This time we chose a fixed view point in the SPONZA scene, using the default non-specular settings and measured the performance (and cache count) for different CAV resolutions. The results are plotted in Figure 5.6.

The implementation of the cache lighting pass uses 512 threads per group since, on average, it performed best with this count. As the shared memory optimization (see Subsection 4.5.5) enforces that all threads of a group remain active, this is also the lowest theoretical granularity to which the performance of this pass can react. Obviously though, due to the characteristics of our graphics card (number of multiprocessors, minimum/-maximum parallel thread groups in flight etc.), the measured granularity is much coarser, especially for a low number of thread groups. More experiments with different thread group sizes would be necessary for a deeper insight into this matter.
This experiment also shows that the allocate pass is more sensible to large cache counts than the apply pass.

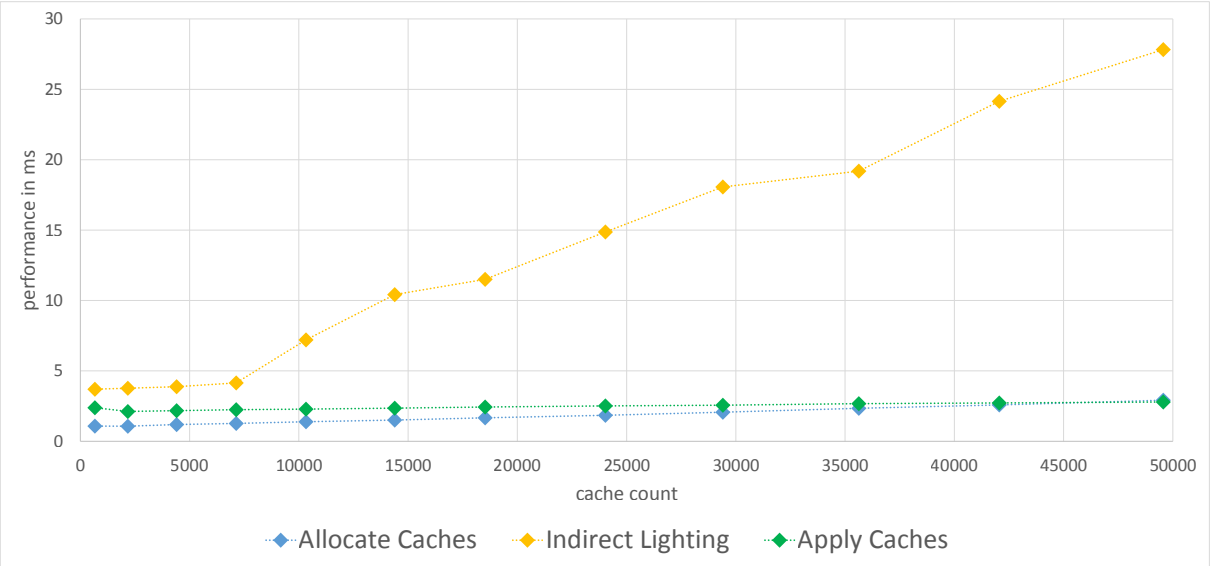


Figure 5.6: Performance for different cache counts. Recorded in Sponza with varying CAV resolutions, starting with 16^3, ending with 192^3. Otherwise default settings without indirect specular.

5.3.3 SH Band Count

For diffuse lighting we use either two or three spherical harmonics bands. Using more or a generic count of bands was considered too complex to implement for the large associated performance hit that is expected to bring only slight improvement in lighting. In fact, already switching from two to three bands can result in a much costlier cache apply pass, as can be seen in Table 5.1. The increased band count means higher reg-

	CAV 32^3 (2177)		CAV 64^3 (7128)		CAV 128^4 (24033)	
	Ind. Lighting	Apply	Ind. Lighting	Apply	Ind. Lighting	Apply
2 bands	3.75	2.12	4.16	2.25	14.8	2.51
3 bands	3.98	3.83	4.58	4.14	16.7	4.74

Table 5.1: Performance of indirect lighting and cache apply passes in ms for different numbers of spherical harmonics bands and CAV resolutions (resulting cache counts in parenthesis) in the Sponza scene.

ister pressure and more arithmetic for the indirect lighting pass as well. However, as the numbers show, the impact is more limited here. Since the main bottleneck is the visibility calculation, a few more or less arithmetic instructions do not have a large influence. Other passes are obviously not affected by the precision of our spherical harmonics representation.

5.3.4 RSM Resolution

The reflective shadow map resolution has a major impact on the duration of the indirect lighting pass which is practically always the slowest step in our rendering process.

RSM resolution	shadow/specular		
	off/off	on/off	on/on
16^2	0.220	1.423	3.276
32^2	0.419	2.480	6.138
64^2	1.200	3.875	16.260
128^2	4.311	9.710	50.160
256^2	16.764	38.419	155.660
512^2	66.587	154.771	400.265

Table 5.2: Performance of the cache lighting pass in ms for different reflective shadow map resolutions in the Republique scene. All tests the used same number of shadow cones.

Table 5.2 shows its influence on the performance in the Republique scene with a fixed view point that activates 3045 caches. Increasing the size of the reflective shadow map also increases the number of shadow cones per cache. Since we are only interested in the effects of the RSM resolution we canceled out that factor by increasing the shadow LODs so that the number of cones stays at 256.
Resolutions beyond 128^2 are practically not usable from a real-time performance standpoint, regardless of indirect shadows and specular effects being activated.

5.3.5 Indirect Shadow

Comparing the statistics on cache count influence with (see Figure 5.4) and without shadowing (see Figure 5.5) we can already deduce that indirect shadowing has a rather high performance impact depending on the visible scene portion which determines the average length of the shadow cones. The shadow lod is more important though, since, together with the RSM resolution, it determines how many shadow cones are needed for each cache.

RSM res. / shadow LOD	16^2	32^2	64^2	128^2	256^2
0	2.16	8.89	36.91	150.09	538.45
1	0.62	2.48	10.06	44.21	231.43
2	0.33	0.97	3.87	15.73	67.37
3	0.29	0.71	2.45	9.70	39.12
4	0.29	0.67	2.19	8.35	33.29
5		0.66	2.15	8.11	32.00
6			2.15	8.08	31.79
7				8.08	31.81
8					31.79

Table 5.3: Performance of the cache lighting pass in ms for different shadow LODs and RSM resolutions in the Republique scene.

Table 5.3 shows measurements of the duration of the cache lighting pass depending on shadow LOD and reflective shadow map resolution. The last entry in each column was rendered with a single large shadow cone. Each entry above used a quadrupled number of shadow cones.

Another important parameter is the resolution of the voxel volume. For simplicity, the application only supports cubic volumes that are aligned to the entire scene. This results in rather poor voxel space usage as all our scenes have a larger extent on a single dominant axis.

shadow LOD \ Voxel res.	32^3	64^3	128^3	256^3	512^3
0	22.22	29.61	36.91	112.07	301.33
1	6.89	8.60	10.06	24.23	68.54
2	3.26	3.56	3.87	6.03	16.69
3	2.34	2.41	2.45	2.64	3.29
4	2.18	2.19	2.19	2.21	2.24
5	2.15	2.15	2.15	2.16	2.16
6	2.15	2.15	2.15	2.15	2.15

Table 5.4: Performance of the cache lighting pass in ms for different shadow LODs and voxel volume resolutions in the REPUBLIQUE scene. RSM resolution was fixed to 64^2.

Table 5.4 shows how the resolution of the voxel volume influences the duration of the cache lighting pass. For unrealistic high shadow LODs there is almost no difference in performance for different voxel resolutions. This can easily be explained by the fact that a high shadow LOD results in large cones that take only few samples in high order mip-levels, thus using a low resolution version of the voxelization.
We deliberately did not include measurements on how long the voxelization takes, as this was discussed in the works from which our voxelization-implementation is derived.

5.3.6 Indirect Specular

So far we have ignored the effect of indirect specular lighting, except in the breakdown overview in Figure 5.3. Indirect specular lighting in general has a severe impact on the performance.
As noted in Section 4.5.5 there are two ways to handle writes to the specular environment map: Either by writing directly into it or by caching the map in the shader's registers. Table 5.5 shows timings of the indirect lighting pass in the SPONZA scene with varying RSM and specular environment map resolutions for both direct and cached write. The RSM resolution determines how often writes to the specular environment map occur. Using cached writes, specular environment maps with a resolution of 8^2 and lower are clearly faster on our machine. For 16^2 the differences are not very significant and 64^2 leads to a crash on our machine since the driver was not able to compile the shader (which indicates a driver bug since there was no controlled error output).

spec. map res. \ RSM res.	16^2	32^2	64^2	128^2	256^2
4^2	1.04	3.05	11.14	43.20	170.39
8^2	1.16	3.48	12.83	49.67	195.66
16^2	1.30	3.62	12.95	49.86	195.87
32^2	1.72	4.07	13.33	49.96	195.24
64^2	3.42	5.78	15.06	51.76	196.83

(a) Direct write.

spec. map res. \ RSM res.	16^2	32^2	64^2	128^2	256^2
4^2	0.74	1.74	5.89	22.50	88.66
8^2	0.90	2.12	7.09	26.92	105.93
16^2	3.75	3.05	9.32	34.43	134.22
32^2	8.48	14.34	37.82	131.00	501.43
64^2	*application crash*				

(b) Register cached write.

Table 5.5: Performance of the cache lighting pass in ms for different specular environment and RSM resolutions in the Sponza scene. The colors indicate where direct writing to the map or caching in the shader's registers is better.

For direct writes, the performance is very stable independently of the specular environment map resolution. However, this parameter can also raise memory requirements drastically (see also Section 5.5).

5.3.7 Resolution Dependence

Finally, we want to test the effect of the screen resolution on our global illumination algorithm. Both the cache allocation and interpolation (= apply) passes are executed for every screen pixel. As expected, our tests in Figure 5.7 show an almost linear correlation to the number of screen pixels. The duration of the allocate pass increases especially slow in respect to the resolution, compared to the allocate and apply passes. For this test we used the Sibenik scene and increased the CAV resolution to 64^3 for a higher cache density.

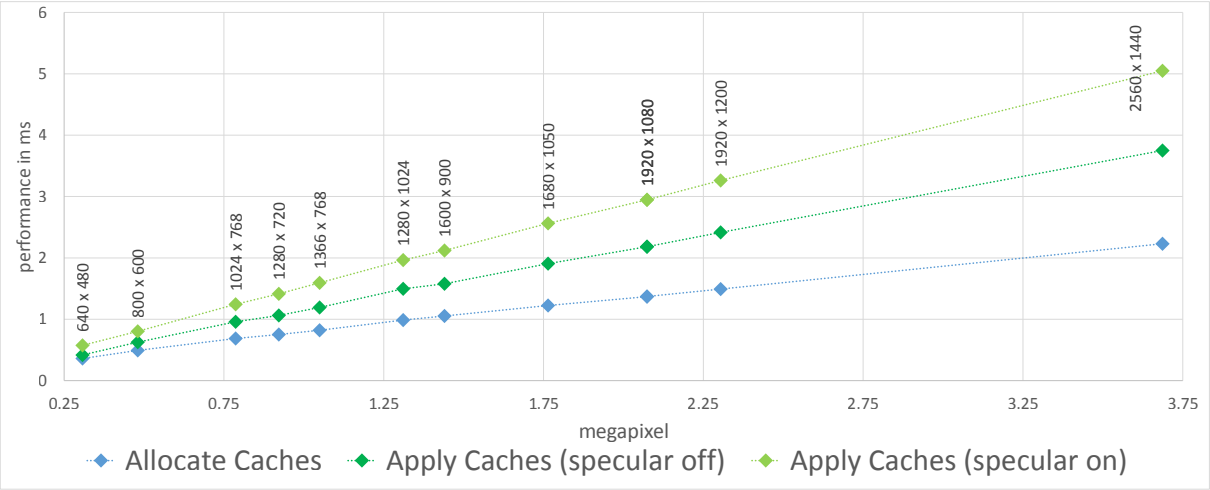


Figure 5.7: Performance of cache allocation and interpolation in ms at different screen resolutions in SIBENIK scene. Used CAV with 64^3, resulting in about 7300 caches.

5.4 Quality

While the previous section examined the effect of several parameters on how well the approach performs in terms of computation time, this section will try to evaluate its general visual quality.

5.4.1 Sources of Error

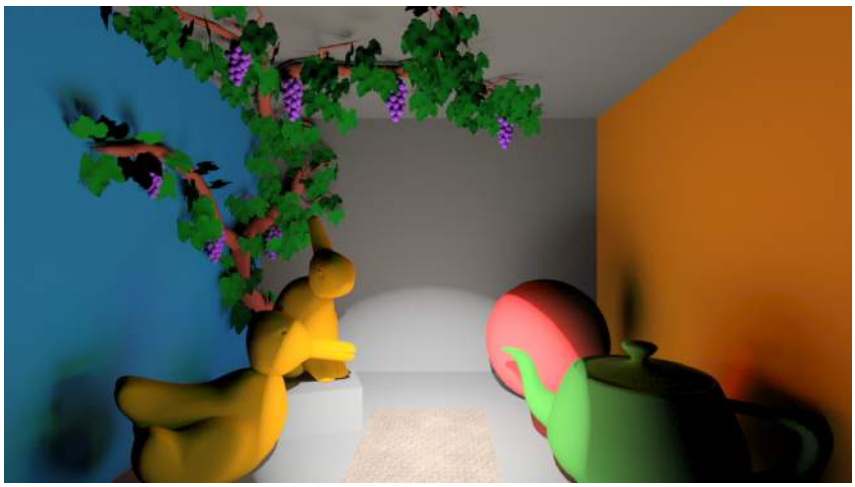
There are several sources of error in regard to the ground truth that is defined by the rendering equation. As the approach does not claim to be able to compute several phenomena at all, we limit these observation to the officially supported types of light-paths. In addition to both specular and diffuse direct lighting, our approach is able to compute specular and diffuse lighting after a diffuse light bounce. As in the original reflective shadow mapping, our approach is not able to handle indirect specular light bounces which are necessary to compute caustics.

The reflective shadow map itself is the first source of error. Problems arise from its limited resolution that determines how well the scene is sampled. Low RSM lead to flickering during movement and irregular lighting, even with the bias compensation techniques which were discussed in Subsection 3.1.2. Also, small features may be missed entirely which leads to either too low or too high brightness in different areas. Our approach to use down-sampled high resolution RSMs reduces flickering but creates virtual lights at implausible in-between positions (as already mentioned in Subsection 4.3.1).
The discretized placement of light caches and their later interpolation has a similar effect on the overall accuracy, however in a more temporally stable fashion. It assumes that there are no features of interest in-between two caches and that the transition can be expressed by simple linear interpolation.
This is especially hazardous in combination with the RSM visibility approximation which is performed by cone-casts in a pre-filtered volume: The higher the distance between the caches, the more likely is it that a cone starts within a solid object even when using a

cache resolution dependent offset. Voxel cone-casts assume that sampling the density information in arbitrary volume mipmap-levels expresses an isotropic visibility function for any cone going through this sample. This estimate is more incorrect the more complex the geometry is which was combined into a single opacity value, i.e. it gets worse for higher mip-levels and lower resolutions. Additionally, as we do not perform solid voxelization, blockers vanish in high mipmap-levels especially for a high starting resolution. Larger cone angles produce higher errors as they perform samples more often in high mipmap-levels, while tight cones ray-march through (relatively) high-detailed volumes. Testing multiple VALs at once with a single cone (shadow LOD) is based on the assumption that the visibility between arbitrary VALs does not vary much, which is obviously wrong for many cases.
Each of the two ways in which lighting information is saved have their own accuracy issues. Since our SH representation (for diffuse lighting) uses only very few bands, it can catch radiance only at very low frequencies. Using higher SH basis would improve this but inevitably leads to stronger ringing artifacts. While high frequencies are usually not necessary to estimate the irradiance, the representation is not entirely accurate. The specular lighting on the other hand is limited by the per-cache specular environement map resolution that determines practically how large the maximal supported Blinn-Phong exponent is. Saving only a hemisphere does not introduce any new errors but the hemispherical projection creates some distortion - to filter samples on this map is not identical with filtering on the hemisphere. More importantly, the isotropic filtering process via mipmaps is flawed, as samples on higher mipmap-levels do not represent correct integrals over the anisotropic Blinn-Phong lobes. However, in practice the largest error originates from the assumption that a single VAL only affect a single pixel in the specular environment map. This is less accurate with *higher* specular environment map and lower RSM resolutions. Note that all specular reflections can only show directly lit geometry due to the path length restrictions.

5.4.2 Ground Truth Comparison

Figure 5.8 shows renderings of a simple diffuse-only scene computed with Mitsuba's path-tracer [Jak10] (top) and our technique (bottom). Please note that we were not able to include such ground truth images for the other scenes since it is rather difficult to share complex model and material setups in such way that fundamentally different renderers interpret them equally. We chose a single address volume with a resolution of 128^3, other than that we used the default settings without specular lighting since we were not able to replicate our BRDF setup in the path-tracer. Using this rather high cache frequency, our approach is able to depict most shadowing details, however fails to compute them accurately. The shadows are generally too bright and - not easily visible due to the tonemapping - tend to darken wide regions that should not be shadowed at all. The brightening originates primarily from the thin voxelization which yields too low blocking values in higher voxel volume mipmaps levels. Though, in general, the simple low frequent lighting in this scene is no problem for our approach. The jagged shadow border on the red ball originates from the regular cache distribution. While it is rarely visible on textured surfaces, it is also very hard to avoid.



(a) Mitsuba [Jak10] path-tracer, 2k samples per pixel ($\sim$ 15 min)

(b) Rendered with Dynamic Radiance Volumes ($\sim$ 25 ms)

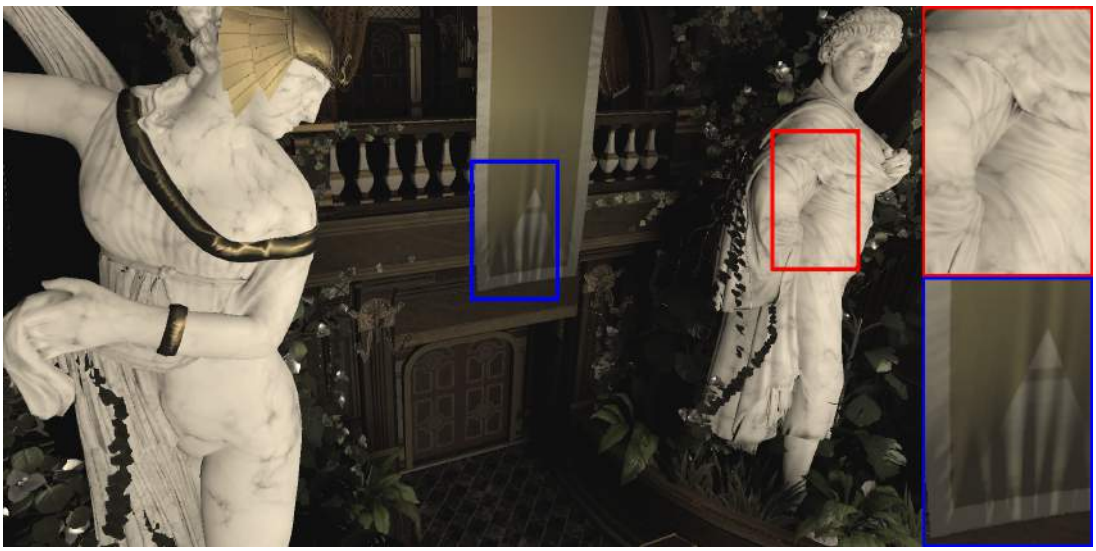
Figure 5.8: Comparison of a (diffuse only) scene rendered with the Mitsuba [Jak10] path-tracer and with our realtime solution.

5.4.3 SH Band Comparison

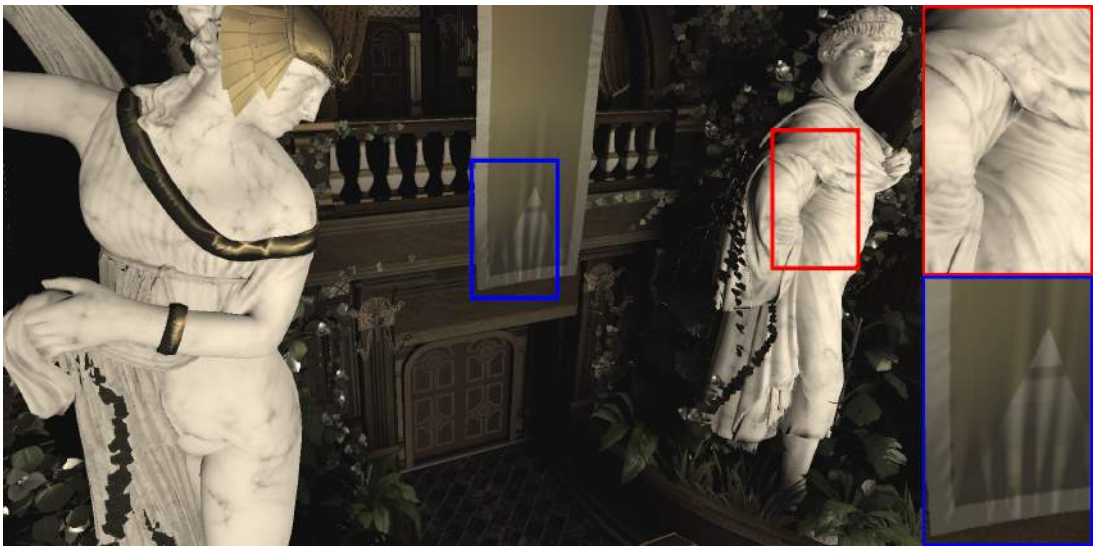
The quality difference between two or three spherical harmonics bands for diffuse lighting is often not very high. Figure 5.9 shows an example with both negligible and rather well visible differences. While the shading at the left statue changes only slightly, the right statue is more contrasted with three SH bands. Generally, the setting with more SH bands displays crisper transitions between dark and bright zones but does not necessarily lead to more variations. For example, the strongly normal-mapped flag in the middle of the scene barely changed. For two or three SH bands we found almost no ringing artifacts in our test cases.

5.4.4 Indirect Shadow

Figure 5.10 shows indirect shadow for different CAV and voxel resolutions. In Figure 5.11 the corresponding cache-density is visualized. Sometimes visible jaggies occur at a shadow's border for a lower cache density. Finer details get lost and Peter Panning is



(a) 2 SH bands.



(b) 3 SH bands.

Figure 5.9: Different spherical harmonics bases for diffuse lighting in the Republique scene.

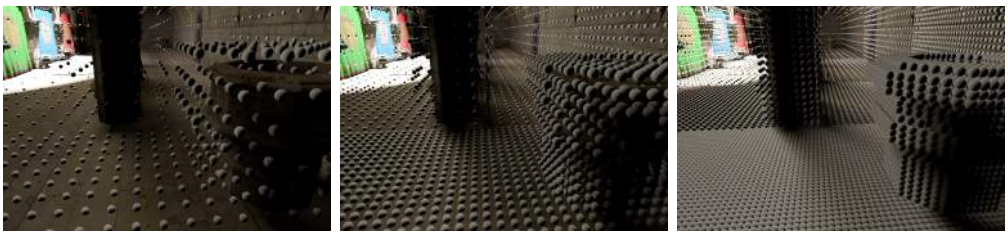
more likely to occur. Using multiple cascades, the more distant cascades may have too few caches to represent the shadow at all, as can be seen in the corridor in the upper right of the images. Higher voxel resolution does not only have advantages: While the shadow borders are more accurate (especially for high CAV resolutions), it is also disproportional brighter and tends to miss distant thin features since the volume is less dense in its higher mipmap-levels. To avoid these issues, solid voxelization would be necessary.
Similar behavior can be observed for the shadow LOD, see Figure 5.12. With raising shadow LOD, the shadow gets brighter and thin features are more likely lost. Besides, these images also feature the ability to produce "colored shadows" which are an implicit consequence of our technique.

(a) Voxelization: 128^3 CAV: 4×32^2 **(b)** Voxelization: 256^3 CAV: 4×32^2

(c) Voxelization: 128^3 CAV: 4×64^2 **(d)** Voxelization: 256^3 CAV: 4×64^2

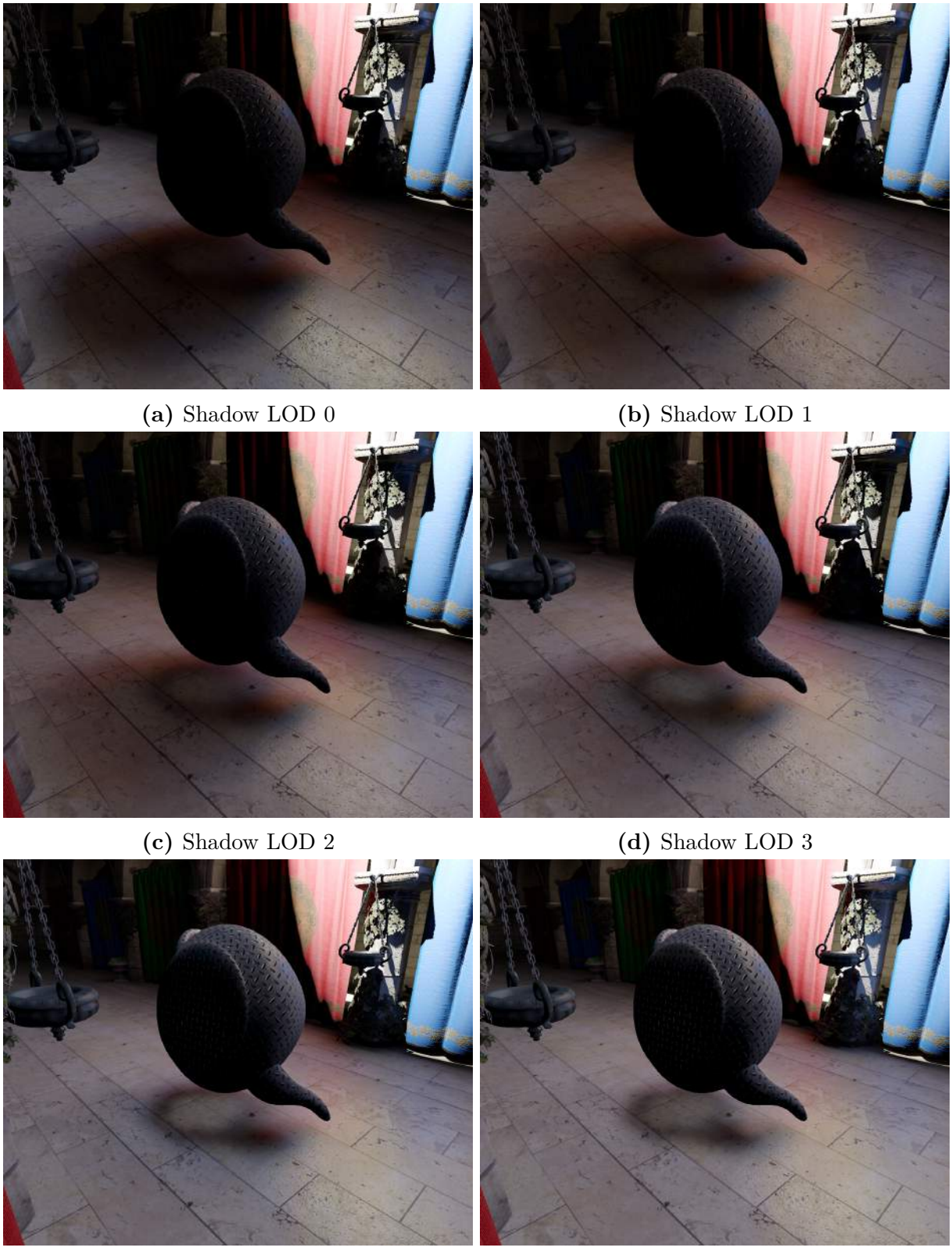
(e) Voxelization: 128^3 CAV: 4×128^2 **(f)** Voxelization: 256^3 CAV: 4×128^2

Figure 5.10: Indirect shadow for different voxel volume and cascaded address volume resolutions in the Sponza scene.



(a) CAV: 4×32^2 **(b)** CAV: 4×64^2 **(c)** CAV: 4×128^2

Figure 5.11: Cache density visualization for Figure 5.10.



(a) Shadow LOD 0

(b) Shadow LOD 1

(c) Shadow LOD 2

(d) Shadow LOD 3

(e) Shadow LOD 4

(f) Shadow LOD 5

Figure 5.12: Indirect shadow for different shadow LODs in the Sponza scene using a voxel volume resolution of 256^3 and CAV resolution of 64^3. High LODs suffer from the low density in higher voxel mipmap levels which is the result of thin voxelization (as opposed to solid voxelization).

5.4.5 Indirect Specular

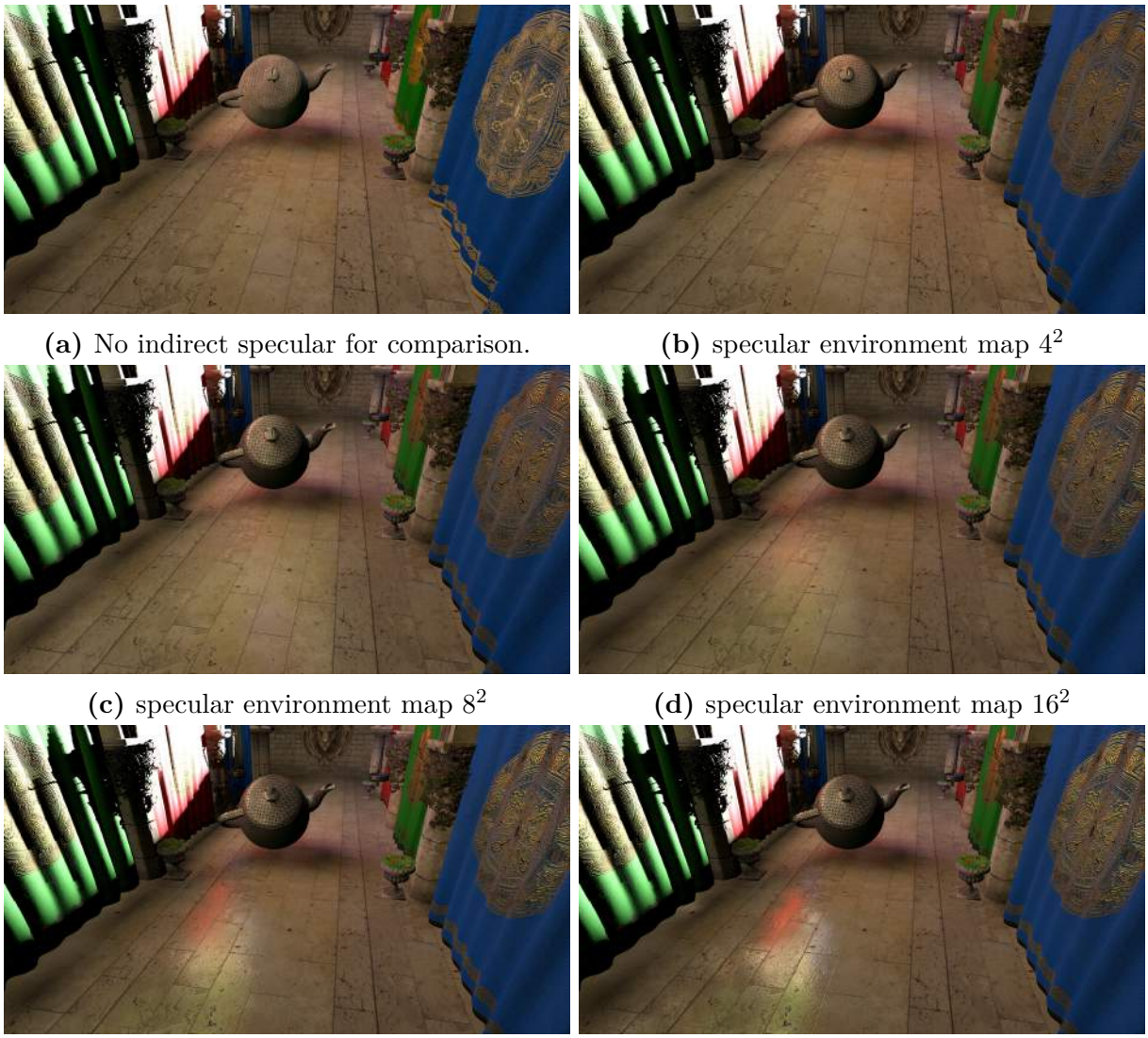
As seen in Figure 5.13, the quality of indirect specular lighting seemingly improves straight-forward with higher specular environment map resolutions. However, there are massive temporal coherence problems for resolutions beyond 16^2 per cache. The indirect specular lighting is the only part of our solution that has temporal coherency issues under camera movement, since the local hemispheres of the caches are view-dependent. In combination with the single write each VAL performs per cache, this can result in flickering. The effect is most problematic with high specular environment map resolutions. As Figure 5.14 shows, this leads to gaps on the specular environment map which are surrounded by pixels which are too bright, since their samples should have been distributed into neighboring pixels as well. We tried to close these gaps using a post-processing step similar to imperfect shadow mapping [RGK+08] but this worsened the temporal artifacts while not noticeably improving the visual quality. However, for smaller resolutions the results are still visually pleasing and rather stable during movements.

5.5 Memory Consumption

RSM	Flux + Depth + Normal	22.6 MiB
Flux	$1024^2 \cdot 6\,\mathrm{B}$ + mipmaps	*8.0 MiB*
Normal	$1024^2 \cdot 4\,\mathrm{B}$ + mipmaps	*5.3 MiB*
Depth + Depth²	$1024^2 \cdot 4\,\mathrm{B}$ + mipmaps	*5.0 MiB*
Depth/Shadow Map	$1024^2 \cdot 4\,\mathrm{B}$	*4.0 MiB*
CAV	$4 \times 32^3 \cdot 4\,\mathrm{B}$	0.5 MiB
Voxel Volume	$128^3 \cdot 1\,\mathrm{B}$ + mipmaps	2.3 MiB
Cache Buffer	$32768 \times$ cache	2.5 MiB
Position (+ padding)	$4 \cdot 4\,\mathrm{B}$	*16 B*
2 Band SH (3 colors)	$16 \cdot 4\,\mathrm{B}$	*64 B*
SpecEnvMap	$32768 \times 16^2 \cdot 4\,\mathrm{B}$	32.0 MiB
Overall		**59.9 MiB**

Table 5.6: Memory consumption break-down of the default configuration with a maximum of about 32k caches.

Table 5.6 shows a memory consumption break-down for the default sample configuration which was mentioned in Section 5.2. The RSM is responsible for a surprisingly large portion of the used space, since we start out with a rather high resolution and sample it down. The flux texture consists of three half-floats, the normals are saved in two signed integers (encoding angles). Further, two depth textures are needed: One for holding both depth and squared depth which is used for the cache-lighting and another one which serves as depth buffer during the RSM rendering and as shadow map later on.
The cascaded address volumes (CAV) consist of unsigned integers (4 B) that address caches in the cache buffer. While the voxel volume has a rather high resolution and needs mipmaps, it is still far below the memory consumption of the RSM textures since each



(a) No indirect specular for comparison.

(b) specular environment map 4^2

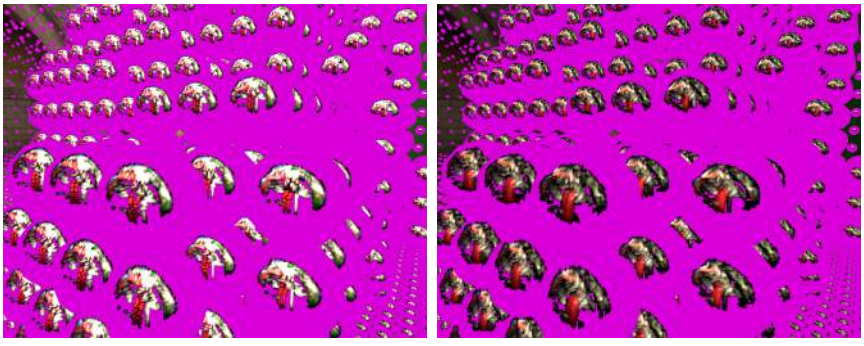
(c) specular environment map 8^2

(d) specular environment map 16^2

(e) specular environment map 32^2

(f) specular environment map 64^2

Figure 5.13: Indirect specular lighting for different specular environement map resolutions in the Sponza scene using CAV resolution of 64^3.



(a) RSM resolution 64^2.

(b) RSM resolution 256^2

Figure 5.14: Specular environment map cache visualization in the teapot in Figure 5.13. Resolution of the specular environment map is 64^2. Unwritten parts have been colored in magenta to highlight the gaps between samples.

voxel needs only a single byte.
The size of the cache buffer depends on the number of used SH bands (two: 64 B three: 112 B) and the number of reserved caches. For this overview we assumed a maximum of 32768 caches, which is well beyond everything we measured. It would also be possible to adjust this number during rendering. If indirect specular is activated, the maximum cache number plays a more important role since each cache needs an individual specular environment buffer (on the atlas). With 32 MiB this is the clearly the largest item in this configuration.

5.6 Comparison to other Techniques

Exact comparisons to other real-time global illumination techniques are difficult since both quality and performance are often highly dependent on various implementation details. To be able to make an appropriate attempt to address this issue, we would have needed to implement all techniques in an unified framework. However, this is beyond the scope of this work.
Therefore, we provide comparisons based only on theoretical facts and our personal estimates. These are summarized in Table 5.7. Diffuse indirections without shadows are generally handled well by all mentioned techniques which is why it is not listed in the table.

Indirect shadowing is comparatively accurate only with voxel cone tracing (VCT) [CNS+11] and our approach. It is not handled at all by reflective shadow mapping (RSM) [DS05] and could possibly be much better in realtime radiance caching by Vardis et al. [VPG14] if more samples would be used.
Indirect specular lighting is a major problem for all state-of-the-art global illumination techniques. To our knowledge only VCT is able to accurately display glossy features with high specular exponents. Our technique can provide moderate reflections but those have a disproportional effect on the performance. Considering that LightSkin [LB13] handles the problem with very low data overhead using interpolated virtual lights, its results are surprisingly good. RSM can handle the problem by brute force, evaluating the specular lighting for each virtual light in every pixel.
Most techniques, including our own, cannot handle thin geometries well and show light bleeding artifacts. Subsection 3.1.2 already discussed how to fix the typical "bright spot artifacts" of RSM. Techniques with pre-computations like LightSkin can alleviate this problem, but may still suffer from missing detail for smaller features.
With the exception of VCT, memory requirements are rarely a problem for the listed techniques. Especially our technique is able to run on a very limited budget, as has been shown in the last section.

Method	*Ind. shadows quality* & basic shadow approach	*Ind. specular freq.*& basic approach	Typical artifacts	*Memory requ.* & main bound	Main perf. bound
Ours	*rather accurate* cone traced (to RSM)	*medium* radiance hemispheres	specular flickering shadow jaggies, bleeding	*low* spec. env. map	shadow quality, RSM resolution
Vardis et al. [VPG14]	*coarse* ray march in low-res. occupancy volume	*low (theory)* spherical harmonics	light bleeding	*medium* SH bands, volumes	volume resolution RSM sample count
LightSkin [LB13]	*coarse, no color* blocker estimates	*low* similar to radiance cache encoded as virtual light	small features wrong slight bleeding	*medium* cache references	cache count
VCT [CNS+11]	*rather accurate* cone traced (in volume)	*high* approx. with thin cone	blocky reflections bleeding	*high* sparse voxel octree	cone count volume resolution
LPV [KD10]	*very coarse* injected blocker volume	*very low* spherical harmonics	heavy bleeding limited light range	*medium* SH bands, volumes	volume resolution, SH bands
RSM [DS05]	n/a	*arbitrary* brute force	bright splotches ("fixable")	*very low* RSM	RSM resolution, sample-count per pixel

Table 5.7: Coarse comparison of our technique with other realtime global illumination techniques. Except LightSkin [LB13], all listed approaches work without any pre-computations.

6 Conclusion

6.1 Summary

We have presented a novel real-time global illumination approach that works on arbitrary scenes without any pre-computations. Similar to Vardis et al. [VPG14] we use a regular grid of light caches that are activated on demand. Our cascaded address volume (CAV) points dynamically to positions in a large buffer rather than storing the data itself, allowing low memory consumption and straight forward processing of light caches. After an allocation pass that fills the CAV, the actual lighting pass is executed.
The lighting pass evaluates the radiance from a reflective shadow map, using virtual area lights as proposed by Lensing et al. [LB13]. Radiance is saved in a low order SH representation for each cache to perform diffuse lighting later on. Optionally, shadow cones are traced in a pre-filtered voxel volume. Governed by a global shadow LOD setting, cones may be traced for bundles of virtual lights (from the reflective shadow map). Since the SH representation is not sufficient to depict glossy reflections, the hemispherical specular environment maps can be used in addition: Each cache is associated with a small rectangular projection of the hemisphere pointing to the viewer. Radiance is added to this texture for each visible virtual light. The maps are pre-filtered to allow fast evaluation of reflection with different exponents.
Finally, the data is evaluated and interpolated per pixel in the cache interpolation pass using per-pixel normals and material properties.

6.2 Evaluation

The technique is entirely invariant under animations and allows to perform arbitrary scene changes at runtime. The same goes for materials which are allowed to have arbitrary variations over surfaces (contrary to techniques like LightSkin [LB13]).
All steps are executed on the GPU without any transfers to the CPU or back, except camera and scene settings. This way the approach is free of any synchronization stalls and frees the CPU for other tasks.

We listed temporal coherency as most important quality criterion. While our algorithm is often stable, we were not able to avoid this issue: Typical reflective shadow map flickering can occur when moving directly lit objects or the light itself. Another source of incoherency are high resolution specular environment maps. Note however, that there are no incoherencies at all for camera movements in general and animated objects that receive little direct light.

As far as other artifacts are concerned, our approach fares relatively well. While it suffers from light bleeding like most techniques, our cascaded address volume is able to limit

these problems adaptively. Using simplified voxel cone tracing we were able to create soft and convincing indirect shadows. Indirect specular lighting is quality-wise clearly solved better in a few other techniques like voxel cone tracing [CNS+11]. Still, our new idea of hemispherical environment maps achieves better results than most competing methods with similar constraints, even at low resolutions. Subsection 5.4.1 provided a concise analysis of all error sources when compared to a ground truth solution within the given light path types.

Contrary to previous work our approach selects relevant caches in screen space and stores them in a continuous buffer. This allows us to compute indirect lighting very efficiently. However, as we traverse every single reflective shadow map pixel (similar to in Light-Skin [LB13] but unlike to most techniques which only sample the RSM) this pass is still very expensive. In this context the simplifications introduced by the shadow LOD were shown to be vital for the overall performance. In comparison to the actual cache lighting, the addressing and interpolation overhead seems to be rather low. Because of this we refrained to use down-sampling for our screen-depended passes. As all cache-based methods, the technique is relatively invariant to resolution changes and is ready for modern "ultra high definition" displays without the need of upsampling. In general, the performance of our approach can compete with other techniques on the field as long as indirect specular lighting is not activated. The latter unfortunately has, depending on the RSM resolution, a seemingly disproportionate impact on the duration of the cache lighting pass.

By using modern GPU programming strategies, we were able to work with a very limited memory budget which is superior to the consumptions of most techniques. Again, the specular lighting has the highest impact, followed by the RSM which may be rendered with a lower resolution (loosing the advantages of down-sampling and sharing the shadow map for direct lighting).

By cascading our address volumes we are able to handle large scenes easily. Though the large gaps between caches in distant cascades may lead to artifacts, especially for indirect shadows. It may be advisable to perform shadowing and specular lighting only for caches in the closer cascades.

6.3 Future Work

Voxel cone traced shadows worked rather well in our context. For higher quality, fast solid voxelization as proposed by Schwartz et al. [SS10] needs to be evaluated. Higher performance for thin cones may be achieved by empty space skipping within the volume. It may be possible to quickly generate conservative distance fields by leveraging the volume hierarchy.
Our shadow LOD value algorithm is very rigid at the moment and could be made adaptive: By choosing target shadow cone angle, the size of the VAL groups could be varied depending on their distribution. However, this is more difficult to implement without severe thread divergence.

A big problem of our specular environment map system are the holes that inevitably originate from the way it is filled. These could be fixed either by a more clever hole filling algorithm or a novel way to fill the map. Likely it would be much better to fill the map

by searching the most relevant virtual lights for each pixel and thus write to the map only once. This process is rather difficult since the most important unshadowed light(s) needs to be found as quickly as possible.
Another way to alleviate the issues could be to accumulate results from several past frames. By jittering the RSM samples, new specular environment map pixels could be filled every frame. Currently, the positions at which caches are saved within the cache buffer and the specular environment map atlas change every frame, which makes such multi-frame approaches difficult. Additionally, caches entering and leaving the view area would need special care.
Currently, the specular environment map covers the entire hemisphere, no matter which parts will be sampled in the interpolation pass later on. The cache allocation pass could additionally store some information on which parts are actually needed for a given frame.

We have assumed that all operations are performed for every cache. Which features are computed for which cache could be selected in different frequencies, depending on the feature and distance. For example, specular could fade out for distant objects and diffuse lighting does not need to be obtained as often as shadows and reflections.
The cache placement itself is regular. This can sometimes lead to jagging artifacts, especially for indirect shadow borders. Jittered or even geometry adaptive alignments might be able to solve this problem.
Chroma down-sampling for SH coefficients as proposed by Vardis et al. [VPG14] might enhance the speed of the interpolation pass.

There are many big challenges on the field of real-time global illumination. Ultimately, our work did offer some new useful combinations and ideas, but no truly efficient solutions to indirect shadows or even indirect (glossy) reflections which remain open problems. We excluded several important aspects like multi-bounce lighting and caustics entirely which need to be explored as well.

Appendices

A. Abandoned Cache Generation Approaches

Before we aligned all caches in camera centered, regular grids, where each cache has no information other than a position, we tried various other techniques. The final regular grid technique has several disadvantages: A cache has only a position and roughly half of the caches lie behind the geometry to be lit. Since all pixels that will be influenced by a cache may be known up-front, it should be possible to feed more information to the caches. Even better, placing caches directly *on* geometry should result in more accurate lighting and could yield a local normal vector and material properties as well, similar to the pre-computed caches in LightSkin [LB13]. As explained by the following examples we were not able to achieve such an intelligent placement of "complex" caches in real-time.

We started with experiments involving data extractions from the G-buffer. Since we realized soon that it is extremely difficult to produce flicker-free movements when relying only on screen-space information without any world-space reference/anchor, we computed property averages of all pixels that lie in world-oriented grid cells. The final cache would lie at the average position of all involved pixels. The obtained normals and positions were most of the time smooth but faced temporal coherency problems at depth discontinuities. Also this approach was rather slow since many threads tended to read/write the same data.

Another grid-based approach was to gather information during the voxelization pass in which normals, texture coordinates and thus material properties are available. Using the depth buffer from the G-buffer stage, we were able to quickly decide which cells need to be filled with additional data. Performance characteristics were a bit better as with the G-buffer approach since there were not as many writes into the same cache. However, even when super sampling the voxel grid, the triangles that influence a cache can change rapidly from frame to frame. This results in abruptly changing cache-normals. For other material properties this problem is less dominant since they are read from textures where it is possible to sample the right mipmap to cover the cell.

A completely different way to obtain caches is by using all vertices as caches. For triangles that cover too much screen-space, more vertices/caches can be generated by using tessellation. Otherwise if for objects that are very far random vertices are discarded, using the vertex ID as random-seed. The most difficult part however, is how the resulting caches can be interpolated. Both tessellation and vertex discarding disrupts the original topology which is either way only doubtfully a good starting point for interpolation (due to cuts in meshes etc.). Tessellation itself introduces many problems: The vertices have

no longer unique identifiers and the maximal tessellation level is not only slow but often not sufficient. Therefore, it is not possible to guarantee a minimal cache density in screen-space this way.

Interpolation without any topology (grid or mesh) is a difficult problem that is addressed by several (irr)radiance caching papers, most notably by Ward's Irradiance Gradients [WH08]. Adding more data to caches makes the interpolation more difficult as well: Caches with position, normal and roughness have a six dimensional parameter space. For strict linear interpolation within a regular grid on all parameters, a pixel with individual parameters would need to access neighbors on all axis which yields 64 caches. Since this is clearly too expensive, other ways of interpolation (and cache organization) are necessary like ignoring specific parameters for interpolation or interpolating within tetrahedrons instead of cubes.

B. Spherical Harmonics Evaluation

All vectors in this appendix are given in spherical coordinates. To convert to Cartesian coordinates see Equation 2.15.

B.1 Polynomial Forms of Spherical Harmonics Basis

$$y_0^0(\theta,\varphi) = \frac{1}{2\cdot\sqrt{\pi}} \tag{.1}$$

$$y_1^{-1}(\theta,\varphi) = \frac{-\sqrt{3}\cdot\sin(\theta)\cdot\sin(\varphi)}{2\sqrt{\pi}} \tag{.2}$$

$$y_1^0(\theta,\varphi) = \frac{\sqrt{3}\cdot\cos(\theta)}{2\sqrt{\pi}} \tag{.3}$$

$$y_1^1(\theta,\varphi) = \frac{-\sqrt{3}\cdot\sin(\theta)\cdot\cos(\varphi)}{2\sqrt{\pi}} \tag{.4}$$

$$y_2^{-2}(\theta,\varphi) = \frac{\sqrt{15}\cdot\sin(\theta)\cdot\sin(\varphi)\cdot\sin(\theta)\cdot\cos(\varphi)}{2\sqrt{\pi}} \tag{.5}$$

$$y_2^{-1}(\theta,\varphi) = \frac{-\sqrt{15}\cdot\sin(\theta)\cdot\sin(\varphi)\cdot\cos(\theta)}{2\sqrt{\pi}} \tag{.6}$$

$$y_2^0(\theta,\varphi) = \frac{\sqrt{5}\cdot\left(3\cdot\cos(\theta)^2-1\right)}{4\sqrt{\pi}} \tag{.7}$$

$$y_2^1(\theta,\varphi) = \frac{-\sqrt{15}\cdot\sin(\theta)\cdot\cos(\varphi)\cdot\cos(\theta)}{2\sqrt{\pi}} \tag{.8}$$

$$y_2^2(\theta,\varphi) = \frac{\sqrt{15}\cdot\left((\sin(\theta)\cdot\cos(\varphi))^2-(\sin(\theta)\cdot\sin(\varphi))^2\right)}{4\sqrt{\pi}} \tag{.9}$$

B.2 Clamped Cosine Lobe in Spherical Harmonics Representation

Spherical Harmonics coefficients for a clamped cosine lobe in a given direction.

$$c_0^0 = \frac{\sqrt{\pi}}{2} \tag{.10}$$

$$c_1^{-1} = -\frac{\sqrt{\pi} \cdot \sin(\varphi) \cdot \sin(\theta)}{\sqrt{3}} \tag{.11}$$

$$c_1^0 = \frac{\sqrt{\pi} \cdot \cos(\theta)}{\sqrt{3}} \tag{.12}$$

$$c_1^1 = -\frac{\sqrt{\pi} \cdot \cos(\varphi) \cdot \sin(\theta)}{\sqrt{3}} \tag{.13}$$

$$c_2^{-2} = \frac{\sqrt{15\,\pi} \cdot \cos(\varphi) \cdot \sin(\varphi) \cdot \sin(\theta)^2}{8} \tag{.14}$$

$$c_2^{-1} = -\frac{\sqrt{15\,\pi} \cdot \sin(\varphi) \cdot \cos(\theta) \cdot \sin(\theta)}{8} \tag{.15}$$

$$c_2^0 = \frac{\sqrt{5\,\pi} \cdot \left(3 \cdot \cos(\theta)^2 - 1\right)}{16} \tag{.16}$$

$$c_2^1 = -\frac{\sqrt{15\,\pi} \cdot \cos(\varphi) \cdot \cos(\theta) \cdot \sin(\theta)}{8} \tag{.17}$$

$$c_2^2 = \frac{\sqrt{15\,\pi} \cdot \left(\cos(\varphi)^2 \cdot \sin(\theta)^2 - \sin(\varphi)^2 \cdot \sin(\theta)^2\right)}{16} \tag{.18}$$

Bibliography

[AMD15a] AMD. *APP Profiler Kernel Occupancy*, 2015. URL: http://developer.amd.com/tools-and-sdks/archive/amd-app-profiler/user-guide/app-profiler-kernel-occupancy/.

[AMD15b] AMD. *Developer Guides, Manuals & ISA Documents*, 2015. URL: http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/.

[AMHH08] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[Bli77] James F. Blinn. Models of light reflection for computer synthesized pictures. *ACM SIGGRAPH Computer Graphics*, 11(2):192–198, 1977.

[Cam15] Camouflaj. République tech demo, April 2015. URL: https://www.assetstore.unity3d.com/en/#!/content/34352.

[CCD06] Mark Claypool, Kajal Claypool, and Feissal Damaa. The effects of frame rate and resolution on users playing first person shooter games. In *Multimedia Computing and Networking*, volume 6071, January 2006.

[CG12] Cyril Crassin and Simon Green. Octree-based sparese voxelization using the gpu hardware rasterizer. In Patrick Cozzi and Christophe Riccio, editors, *OpenGL Insights*, pages 303–319. CRC Press, July 2012.

[Cho15] Michael Chock. *NV_conservative_raster OpenGL Extension*. Nvidia, March 2015. URL: https://www.opengl.org/registry/specs/NV/conservative_raster.txt.

[CNS+11] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum*, 30(7):1921–1930, September 2011.

[DCB+04] Zhao Dong, Wei Chen, Hujun Bao, Hongxin Zhang, and Qunsheng Peng. Real-time voxelization for complex polygonal models. In *Computer Graphics and Applications, 12th Pacific Conference*, pages 43–50, 2004.

[DGR+09] Zhao Dong, Thorsten Grosch, Tobias Ritschel, Jan Kautz, and Hans-Peter Seidel. Real-time indirect illumination with clustered visibility. In *Vision, Modeling, and Visualization Workshop*, VMV '09, pages 187–196, 2009.

[DKH+14] Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. Scalable Realistic Rendering with Many-Light Methods. *Computer Graphics Forum*, 33(1):88–104, February 2014.

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 161–165, 2006.

[DMAC03] Frédéric Drago, Karol Myszkowski, Thomas Annen, and Norishige Chiba. Adaptive logarithmic mapping for displaying high contrast scenes. *Computer Graphics Forum*, 22(3):419–426, September 2003.

[Dog13] Hawar Doghramachi. Rasterized voxel-based dynamic global illumination. In Wolfgang Engel, editor, *GPU Pro 4*, pages 155–171. CRC Press, 2013.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 203–231, 2005.

[ED06] Elmar Eisemann and Xavier Décoret. Fast scene voxelization and applications. In *Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 71–78, 2006.

[ED08] Elmar Eisemann and Xavier Décoret. Single-pass gpu solid voxelization for real-time applications. In *Proceedings of Graphics Interface*, GI '08, pages 73–80, 2008.

[FC00] Shiaofen Fang and Hongsheng Chen. Hardware accelerated voxelization. *Computers and Graphics*, 24(3):433–442, June 2000.

[Gie11] Fabian Giesen. A trip through the graphics pipeline, 2011. URL: https://fgiesen.wordpress.com/2011/07/09/a-trip-through-the-graphics-pipeline-2011-index/.

[Gre03] Robin Green. Spherical harmonic lighting: The gritty details. In *Archives of the Game Developers Conference*, 2003.

[GSHG98] Gene Greger, Peter Shirley, Philip M Hubbard, and Donald P Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, March 1998.

[Gue07] Paul Guerrero. Approximative real-time soft shadows and diffuse reflections in dynamic scenes. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, October 2007. URL: http://www.cg.tuwien.ac.at/research/publications/2007/guerrero-2008-dip/.

[HKWB09] Miloš Hašan, Jaroslav Křivánek, Bruce Walter, and Kavita Bala. Virtual spherical lights for many-light rendering of glossy scenes. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 143:1–143:6, 2009.

[HM12] Ladislav Hrabcak and Arnaud Masserann. Asynchrounous buffer transfers. In Patrick Cozzi and Christophe Riccio, editors, *OpenGL Insights*, pages 391–414. CRC Press, July 2012.

[Hof14] Naty Hoffmann. Physics and math of shading. In *Siggraph PBR Course*. 2k Games, 2014.

[Hol11] Daniel Holbert. Saying "goodbye" to shadow acne, 2011. Poster presented at Games Developer Conference, San Fransisco. URL: http://www.dissidentlogic.com/old/images/NormalOffsetShadows/GDC_Poster_NormalOffset.png.

[HREB11] Matthias Holländer, Tobias Ritschel, Elmar Eisemann, and Tamy Boubekeur. Manylods: Parallel many-view level-of-detail selection for real-time global illumination. *Computer Graphics Forum*, 30(4):1233–1240, June 2011.

[Int15] Intel. *Developer Documents for Intel® Processor Graphics*, 2015. URL: https://software.intel.com/en-us/articles/intel-graphics-developers-guides.

[Jak10] Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[Kaj86] James T. Kajiya. The rendering equation. In *Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–140, 1986.

[KD10] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 99–107, 2010.

[Kel97] Alexander Keller. Instant radiosity. In *Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 49–56, 1997.

[KGPB05] Jaroslav Krivanek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):550–561, July 2005.

[Khr14a] Khronos Group. *OpenGL 4.5 Core Profile Specification*, October 2014. URL: https://www.opengl.org/registry/doc/glspec45.core.pdf.

[Khr14b] Khronos Group. *OpenGL 4.5 Quick Reference Card*, 2014. Rev. 0814. URL: https://www.khronos.org/files/opengl45-quick-reference-card.pdf.

[KK04] Thomas Kollig and Alexander Keller. Illumination in the presence of weak singularities. In *Monte Carlo and Quasi-Monte Carlo Methods*, pages 245–257. Springer, 2004.

[LB13] Philipp Lensing and Wolfgang Broll. Lightskin: Real-time global illumination for virtual and mixed reality. In *Joint Virtual Reality Conference*, JVRC '13, pages 17–24, 2013.

[Len14] Philipp Lensing. *LightSkin: Globale Echtzeitbeleuchtung für Virtual und Augmented Reality*. PhD thesis, TU Illmenau, July 2014.

[McG11] Morgan McGuire. Computer graphics archive, August 2011. URL: http://graphics.cs.williams.edu/data.

[McL14] James McLaren. Cascaded voxel cone tracing in the tomorrow children. In *Computer Entertainment Developers Conference (CEDEC)*. Q-Games Ltd., 2014. URL: http://fumufumu.q-games.com/archives/2014_09.php#000934.

[MEW+13] Morgan McGuire, Daniel Evangelakos, James Wilcox, Sam Donow, and Michael Mara. Plausible Blinn-Phong reflection of standard cube MIP-maps. Technical Report CSTR201301, Williams College Department of Computer Science, September 2013.

[Mor66] Guy M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM, Ottawa, Ontario, March 1966.

[NED11] Jan Novák, Thomas Engelhardt, and Carsten Dachsbacher. Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 119–124, 2011.

[NPG05] Mangesh Nijasure, Sumanta Pattanaik, and Vineet Goel. Real-time global illumination on gpus. *Journal of Graphics, GPU, and Game Tools*, 10(2):55–71, 2005.

[Nvi15] Nvidia. *CUDA Programming Guide v7.0*, 2015. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[OSK+14] Ola Olsson, Erik Sintorn, Viktor Kämpe, Markus Billeter, and Ulf Assarsson. Implementing efficient virtual shadow maps for many lights. In *ACM SIGGRAPH 2014 Talks*, SIGGRAPH '14, page 50:1, 2014.

[Pap11] Georgios Papaioannou. Real-time diffuse global illumination using radiance hints. In *Symposium on High-Performance Graphics*, HPG '11, pages 15–24, 2011.

[Pes15] Alexandre Pestana. Base color, roughness and metallic textures for Sponza, Februrary 2015. URL: http://www.alexandre-pestana.com/pbr-textures-sponza/.

[PF05] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[PKD12] Roman Prutkin, Anton Kaplanyan, and Carsten Dachsbacher. Reflective shadow map clustering for real-time global illumination. In *Eurographics Short Papers*, pages 9–12, 2012.

[Pre14] Shaun Prescott. Oculus founder palmer luckey thinks 30 frames per second is 'a failure', June 2014. URL: http://www.pcgamer.com/oculus-founder-palmer-luckey-thinks-30-frames-per-second-is-a-failure/.

[RDGK12] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. *Computer Graphics Forum*, 31(1):160–188, Februrary 2012.

[REG+09] Tobias Ritschel, Thomas Engelhardt, Thorsten Grosch, Hans-Peter Seidel, Jan Kautz, and Carsten Dachsbacher. Micro-rendering for scalable, parallel final gathering. *ACM Transactions on Graphics*, 28(5):132:1–132:8, December 2009.

[REH+11] Tobias Ritschel, Elmar Eisemann, Inwoo Ha, James DK Kim, and Hans-Peter Seidel. Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. *Computer Graphics Forum*, 30(8):2258–2269, December 2011.

[RGK+08] Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics*, 27(5):129:1–129:8, December 2008.

[RZD14] Hauke Rehfeld, Tobias Zirr, and Carsten Dachsbacher. Clustered pre-convolved radiance caching. In *Eurographics Symposium on Parallel Graphics and Visualization*, EGPGV '14, pages 25–32, 2014.

[SC97] Peter Shirley and Kenneth Chiu. A low distortion map between disk and square. *Journal of Graphics Tools*, 2(3):45–52, December 1997.

[Slo08] Peter-Pike Sloan. Stupid spherical harmonics (SH) tricks. In *Archives of the Game Developers Conference*, 2008.

[SNRS12] Daniel Scherzer, Chuong H. Nguyen, Tobias Ritschel, and Hans-Peter Seidel. Pre-convolved radiance caching. *Computer Graphics Forum*, 4(31):1391–1397, June 2012.

[SRS14] Masamichi Sugihara, Randall Rauwendaal, and Marco Salvi. Layered reflective shadow maps for voxel-based indirect illumination. In *Symposium on High-Performance Graphics*, HPG '14, pages 117–125, 2014.

[SS10] Michael Schwarz and Hans-Peter Seidel. Fast parallel surface and solid voxelization on GPUs. *ACM Transactions on Graphics*, 29(6):179:1–179:9, December 2010.

[THGM11] Sinje Thiedemann, Niklas Henrich, Thorsten Grosch, and Stefan Müller. Voxel-based global illumination. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 103–110, 2011.

[VPG14] Kostas Vardis, Georgios Papaioannou, and Anastasios Gkaravelis. Real-time radiance caching using chrominance compression. *Journal of Computer Graphics Techniques*, 3(4):111–131, December 2014.

[WH08] Gregory J. Ward and Paul S. Heckbert. Irradiance gradients. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 72:1–72:17, 2008.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *ACM SIGGRAPH Computer Graphics*, 12(3):270–274, August 1978.

[Wil83] Lance Williams. Pyramidal parametrics. *ACM SIGGRAPH Computer Graphics*, 17(3):1–11, July 1983.

[WRC88] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics*, 22(4):85–92, 1988.

[WS82] Günther Wyszecki and W. S. Stiles. *Color science: concepts and methods, quantitative data and formulae*. John Wiley & Sons, 1982.

Statement of Authorship / Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig und außschließlich unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.
Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder einer anderen Prüfungsbehörde vorgelegt oder noch anderweitig veröffentlicht.

Unterschrift

Datum