Tampere University

Julius Ikkala

REAL-TIME PATH TRACED SPHERICAL HARMONICS PROBES

Master's thesis
Faculty of Information Technology and Communication Sciences
Examiners: Prof. Pekka Jääskeläinen
Dr. Markku Mäkitalo
November 2020

ABSTRACT

Julius Ikkala: Real-time path traced spherical harmonics probes
Master's thesis
Tampere University
Signal Processing
November 2020

Virtual reality headsets and light field displays are relatively new display technologies that are becoming more commonplace. They pose problems because of the extremely high number of pixels that they require to be rendered at relatively high framerates. Simultaneously, the latest high-end GPUs contain acceleration hardware for ray tracing, which enables photorealistic rendering algorithms such as path tracing to be used in real-time to a limited extent. Using the computational power of these power-hungry devices over a network connection can enable more realistic graphics in mobile devices as well.

This thesis proposes a novel hybrid real-time computer graphics rendering algorithm that is particularly well-suited for distributed computing and reproduces indirect lighting phenomena. The algorithm works by computing and storing lighting state at specified points in space, called light probes. These probes are calculated and updated using path tracing, which is a highly realistic light simulation algorithm, during the runtime of the program. Light probes solve the issue of indirect lighting in the rendering pipeline; direct lighting is computed using lightweight rasterization-based rendering.

In the proposed rendering method, the heavy-to-compute indirect lighting part of the rendering pipeline is highly independent of the rest of the pipeline and can be offloaded to a powerful server, leaving the client device with only the lightweight rasterization workload. This allows low-powered devices, such as wireless virtual reality headsets with built-in rendering capabilities, to render highly realistic and dynamic 3D content. The independence of the indirect lighting component from the rest of the system also benefits light field rendering and other multi-viewport rendering, as the intensive indirect lighting computation can be shared between all viewports and only the lightweight rasterization has to be duplicated.

Additionally, a new technique for computing approximated specular lighting from spherical harmonics light probes is proposed. This method applies the split-sum approximation to spherical harmonics and allows for fast approximation of specular lighting for materials with varying roughness. With this technique, the proposed method is able to include both the diffuse and specular lighting components of indirect lighting.

Compared to single-sample-per-pixel path tracing with state-of-the-art denoising, which is another realistic real-time rendering pipeline option, the proposed method achieves greater quality as measured by PSNR in each of the three scenes tested and the effect of increasing resolution reduces performance significantly less.

Keywords: computer graphics, real-time, global illumination, indirect lighting, spherical harmonics, light probe, path tracing

TIIVISTELMÄ

Julius Ikkala: Reaaliaikanen polunjäljityksellä päivitetty palloharmoniavalaistus
Diplomityö
Tampereen yliopisto
Signaalinkäsittely
Marraskuu 2020

Virtuaalitodellisuuslasit ja valokenttänäytöt ovat suhteellisen uutta ja alati yleistyvää näyttöteknologiaa. Niiden vaatima korkeat päivitystaajudet ja suuri pikselimäärä aiheuttavat pulmia. Samanaikaisesti, viimeisimmät ja tehokkaimmat näytönohjaimet sisältävät laitteistokiihdyttimiä säteenjäljitystä varten. Säteenjäljitys mahdollistaa fotorealististen renderöintialgoritmien, kuten polunjäljityksen, toteuttamisen rajallisissa määrin. Näiden paljon tehoa vaativien laitteiden laskentatehon hyödyntäminen verkkoyhteyden yli voi mahdollistaa aiempaa realistisempaa grafiikkaa myös mobiililaitteissa.

Tässä työssä esitetään uusi reaaliaikainen tietokonegrafiikan renderöintialgoritmi, joka soveltuu erityisen hyvin laskentaan hajautetussa järjestelmässä ja kykenee jäljentämään epäsuoria valoilmiöitä. Algoritmi toimii laskemalla ja säilömällä valaistuksen tilaa määritellyissä tilan pisteissä, joita kutsutaan valoluotaimiksi. Nämä luotaimet lasketaan ja päivitetään sovelluksen ajon aikana käyttäen polunjäljitystä, joka on hyvin realistinen valon simulointialgoritmi. Valoluotaimet ratkaisevat epäsuoran valaistuksen renderöintialgoritmissa, kun taas suora valaistus lasketaan kevyitä rasterointipohjaisia menetelmiä käyttäen.

Esitetyssä renderöintialgoritmissa raskas epäsuoran valaistuksen laskentaan liittyvä osuus on hyvin riippumaton muista osista ja voidaan siirtää laskettavaksi etänä tehokkaalla palvelimella, mikä jättää asiakaslaitteelle vain kevyen rasterointityön. Tämä sallii matalatehoisten laitteiden, kuten langattomien virtuaalitodellisuuslasien sisäänrakennettujen renderöintilaitteiston, hyödyntämistä realistisen ja dynaamisen 3D-sisällön kuluttamiseen. Epäsuoran valaistuksen laskennan riippumattomuus muusta järjestelmästä hyödyttää myös valokenttien ja muiden useampaan näkymään perustuvan näyttötekniikan renderöintiä, sillä tuo intensiivinen osa laskennasta voidaan jakaa kaikkien näkymien kesken ja vain kevyt rasterointiosuus tarvitsee toistaa näkymäkohtaisesti.

Myös uusi menetelmä heijastuneen valon karkeaan arviointiin palloharmonisista valoluotaimista esitellään tässä työssä. Tämä menetelmä soveltaa “split-sum approximation”-nimistä aiempaa menetelmää palloharmoniaan ja mahdollistaa heijastuneen valon nopean laskennan materiaaleille, joiden karheus ei ole vakio. Tätä menetelmää käyttäen esitetty renderöintialgoritmi pystyy sisällyttämään sekä sironta- että heijastusosuuden epäsuorasta valaistuksesta.

Eräs toinen realistinen ja reaaliaikainen renderöintimenetelmä on luoda kuva polunjäljityksellä laskien yksi näyte per pikseli ja peittämällä aiheutuva kohina modernilla kohinanpoistoalgoritmilla. Tähän lähestymistapaan verrattuna esitetty renderöintialgoritmi saavuttaa paremman laadun PSNR-metriikalla mitattuna kaikissa kolmessa testitapauksessa, ja resoluution kasvattaminen vähentää sen tehokkuutta huomattavasti vähemmän.

Avainsanat: tietokonegrafiikka, reaaliaikainen, globaali valaistus, epäsuora valaistus, palloharmonia, valoluotain, polunjäljitys

PREFACE

I want to thank Pekka Jääskeläinen and Markku Mäkitalo for their guidance during the Master's thesis process. Additionally, thanks to Matias Koskela, Petrus Kivi, Jan Solanti and Atro Lotvonen for the discussions that eventually led to the ideas presented in this thesis. Thanks to Sofia Purontaus for help with proofreading.

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 783162. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Netherlands, Czech Republic, Finland, Spain, Italy.

Tampere, 26th November 2020

Julius Ikkala

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

3D	Three-Dimensional
BLAS	Bottom-Level Acceleration Structure
BMFR	Blockwise Multi-Order Feature Regression
BRDF	Bidirectional Reflectance Distribution Function
BSDF	Bidirectional Scattering Distribution Function
DDGI	Dynamic Diffuse Global Illumination
GGX	A material model, abbreviation not specified [1]
GPU	Graphics Processing Unit
HMD	Head-Mounted Display
L2	Second-order spherical harmonics representation
LF	Light Field
MSAA	Multisample Anti-Aliasing
PCF	Percentage-Closer Filtering
PCSS	Percentage-Closer Soft Shadows
PSNR	Peak Signal-to-Noise Ratio
PT	Path Tracing
RT	Ray Tracing
SH	Spherical Harmonics
spp	Samples Per Pixel
TLAS	Top-Level Acceleration Structure
VR	Virtual Reality
ZH	Zonal Harmonics

1 INTRODUCTION

With the introduction of ray tracing acceleration in graphics hardware available to consumers in 2018 [2], using ray casting and tracing as part of a real-time rendering algorithm has become reality. While the performance isn't yet there for the high-quality offline methods used in movie rendering to be real-time, many new and better approximations are now possible.

At the same time, interest in multi-viewport rendering is on the rise; virtual reality headsets have been in the hands of consumers for a while and light field rendering is emerging. A greater number of pixels than before have to be rendered at high frequencies, which can make ray tracing every pixel prohibitively slow. For example, a light field display which was used during the testing of the method presented in this thesis, used 32 viewports of 1280x720 pixels each. This is almost 30 megapixels. Another example is the Varjo VR-2 virtual reality headset, which contains two 1920x1080 displays and two 1440x1600 displays, with support for a refresh rate up to 90 Hz [3].

Another aspect is the desire for content on-the-go: the device where content is consumed is usually a rather weak mobile device that is incapable of calculating realistic lighting in real-time. Services like Google Stadia [4] and PlayStation Now [5] stream game content that is rendered on a server as a video stream to the client device. They effectively reduce the amount of overall hardware needed, since instead of each user having a fast GPU that sits idle most of the day, the server hardware can be in use 24/7 by different users. However, the bandwidth cost of streaming image frames like this is high, input delay is a concern and network issues immediately interrupt the experience in its entirety as the image may stop updating. Further, they leave most of the hardware that does exist at the client underutilized.

Taking all of the aforementioned aspects into account, we propose a rendering method based on real-time updated spherical harmonics light probes with support for rough specular reflections. These light probes are computed with path tracing, a highly realistic rendering algorithm. Then, they are consumed by a rasterization-based rendering pipeline, which is the classic and highly optimized computer graphics technique for 3D rendering.

The representation of the light probes is very compact, only 56 bytes per probe without any kind of compression. For a medium-sized scene, a 3D grid of these probes called an “irradiance volume” could contain in the order of 10x10x10 probes, which would only take 56000 bytes to transmit over the network. Further, the irradiance volume is independent of camera location and orientation.

The independence from camera greatly benefits the multi-viewport rendering needed for light fields or VR headsets: each viewport can share the same path traced lighting data, and only the fast rasterization step needs to be duplicated. Another benefit in the context of remote rendering is that this makes the rendering method resilient to poor network reliability. Even if the probe update occurs at a different rate than camera movement or even stops entirely for a while, the only major symptom is slightly outdated lighting in the scene. In the worst-case scenario of not having a connection in the first place, the probes can easily be replaced by static, precalculated values stored on the client device.

Further, separating the lighting from camera position lets the same lighting be shared between multiple users. A conceivable situation where this would be useful is an online game, where all players exist in the same scene. This way, the overall energy requirement of the system would likely be reduced when compared to rendering everything locally on each device, as only the server has to compute the lighting and even that would have to be done only once to serve all players.

However, there is a tradeoff. Both the directional and spatial detail become undersampled in indirect lighting, which manifests as blurry lighting. This only affects indirect lighting, which is usually mostly low-frequency to begin with, making the concession more palatable. In addition to using the probes for diffuse lighting like is usually done, we also present a very lightweight approximation for computing specular reflections from the probes. Reflections from mirror-like objects are not well-handled by this method due to the loss of such directional detail, but rough surfaces are quite well approximated.

In Chapter 2 the general computer graphics theory regarding materials, rasterization and ray tracing is introduced. After that, Chapter 3 explores a method for approximating indirect lighting called “light probes”. Chapter 4 discusses the implementation of the proposed rendering pipeline in detail. Results measured using this implementation are presented in Chapter 5. Similar probe-based real-time rendering algorithms are discussed and compared to the proposed method in Chapter 6. Finally, Chapter 7 concludes the thesis and outlines some possible future improvements.

2 COMPUTER GRAPHICS THEORY

As a research field, 3D computer graphics is concerned with generating images from a mathematically precise model that describes three-dimensional shapes and volumes. Typically, the target is photorealism, in which the generated images should be indistinguishable from a photograph in real world. Another common target is real-time operation, in which the image generation must occur dozens of times in a second. These two goals are difficult to satisfy simultaneously, because the algorithms used for achieving photorealism through accurate simulation of light transport are typically extremely heavy.

Because of the conflicting goals, there have classically been two different approaches: rasterization and ray tracing. The former is a very performant way of rendering 3D models and has been able to produce acceptable images in real-time for decades, but comes with severe limitations. The latter, ray tracing, can be used to simulate light transport to a high degree of accuracy that achieves photorealism in all but the most extreme scenes. However, this kind of precise simulation is also very demanding. To an extent, both approaches can utilize a realistic model of the interaction between individual rays of light and surfaces.

2.1 Surface-Light Interaction

At the core of all realistic computer graphics is the interplay of light and geometry. This interaction between light and material is described by the *Bidirectional Scattering Distribution Function* (BSDF), which is often divided into two parts: the *Bidirectional Reflectance Distribution Function* (BRDF) governing the part of light that gets reflected off the surface, and the *Bidirectional Transmittance Distribution Function* (BTDF) which covers the part of light that gets transmitted through the surface. [6]

There are lots of different BRDFs and BTDFs in use for different materials with varying degrees of realism. The Lambertian reflection models perfect diffuse reflection, meaning that a surface reflects an equal amount of light into all outgoing directions and the intensity only depends on the albedo of the material and the cosine of the angle between the normal of the surface and the direction vector of arriving light [7].

The *GGX* BSDF contains both a BRDF and a BTDF with parameters that allow them to

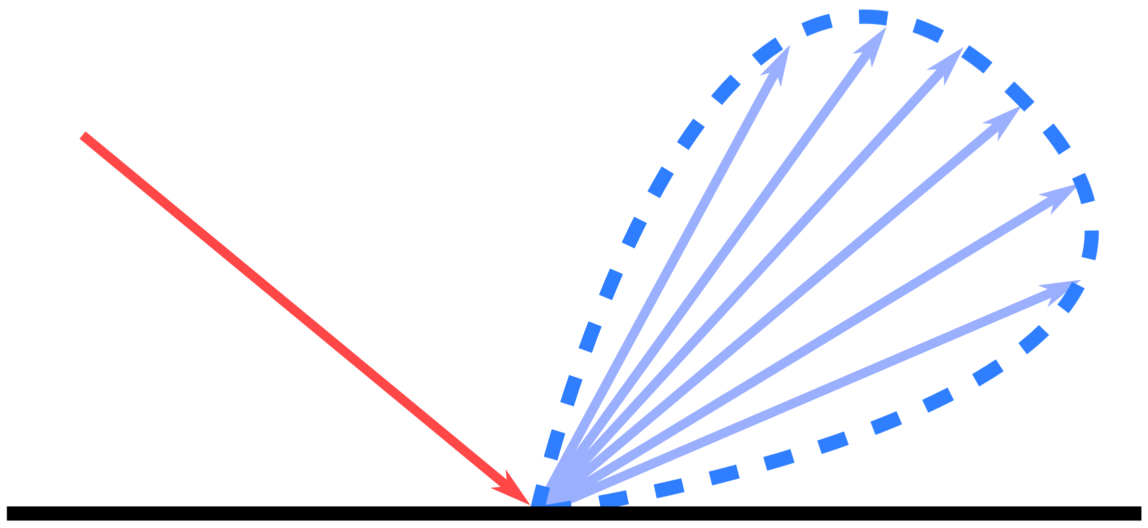


Figure 2.1. *A 2D cross-section diagram of a BRDF lobe. The red arrow represents the outgoing ray, and the blue blob is the lobe. The length of each blue arrow visualizes the value of the BRDF at that direction. The directions of the rays are opposite to the actual direction light would move in; light rays are usually traced in reverse.*

be used for a large variety of different real-world materials with a good degree of realism [1] and is quite fast to evaluate, which is why it has seen quite a lot of use in the real-time rendering industry. One of the parameters is commonly called "roughness", even though it is introduced as the "width parameter" α_g in [1]. This parameter controls how rough the surface appears, with zero being a perfectly smooth, polished surface and one being an extremely matte surface.

The BSDF can be thought of as a spherical function that describes how much light an incoming direction contributes towards an outgoing direction. When the outgoing direction is given, they are often visualized with 3D shapes where the distance of each point from the origin corresponds to the value of the BSDF for that incoming direction. These images resemble a lobe, which is why the BSDF for a set outgoing direction is often called as such. We are often only interested in one half of the BSDF, which is why some lobes only describe the BRDF or BTDF sides. Figure 2.1 shows an example of a BRDF lobe visualization.

2.2 Rasterization

For decades, rasterization has been the most popular technique for rendering real-time computer graphics due to its high performance arising from the simple nature of the algorithm. Rasterization works by projecting only the vertices of each rendered polygon to a 2D plane representing the viewport and then coloring all pixels within the bounds of the projected polygon. This way, each polygon is only considered for those pixels that they cover, which keeps the algorithm performant.

Rasterization has some critical limitations, however. Because each polygon must remain a polygon after the projection operation, only projections that preserve straight lines can be used with it. Fortunately, this includes orthographic and perspective projection which are vital for rendering the first bounce of light, i.e. the light that arrives at the camera sensor. Because modern hardware allows user-programmable code to determine the coloring for each pixel covered by the projected polygon (so-called “Fragment shaders”), some techniques exist that allow bending these rules to an extent [8].

The modern rasterization pipelines supported by 3D acceleration hardware roughly follow the same structure, with some additional features and stages. Vertex coordinates and other per-vertex data is fed to a programmable pipeline stage called the “vertex shader”. This stage applies the desired projection to vertex coordinates and can perform additional transformations on them as well. The vertex data output from the vertex shader is interpolated for each pixel within the projected polygon and fed into the “fragment shader”. This fragment shader is then responsible for calculating the color of the pixel that is written to the frame buffer. Here, the interpolated vertex data and some programmer-defined data buffers are often used to perform lighting simulation.

A “compute shader” stage also exists, and can be used to perform arbitrary GPU-accelerated computation. Technically, it is possible to implement ray tracing in both the fragment shader and the compute shader. In this thesis, the rasterization category only includes those methods that can be implemented with the modern rasterization pipeline but are not equivalent to ray tracing.

2.2.1 Direct Lighting

Direct lighting is the component of lighting that is *local* to a single surface, i.e. it does not take light bounces from other surfaces into account at all; only the direct light source is used. This category encompasses only the rays that travel from a light source into a surface, then to the camera. Thus, only the potentially shadowed reflection of that light from the surface has to be calculated. Rasterization-based approximations can typically be quite accurate with direct lighting, and the performance is not a major issue. Figure 2.2 shows the contribution of direct light to an example scene.

Primarily, the BSDF must be evaluated for each light source and shadowing should be taken into account. When the light is punctual, evaluating the BSDF is typically quite simple. For area lights, a more complex integration is often needed, which is why they are either approximated or omitted. Some real-time approaches for specific types of area lights do exist [9] [10].

Solving the visibility of a light source to the surface is another issue. *Shadow mapping* is a very common method for real-time shadows in rasterization [11]. It works by rendering

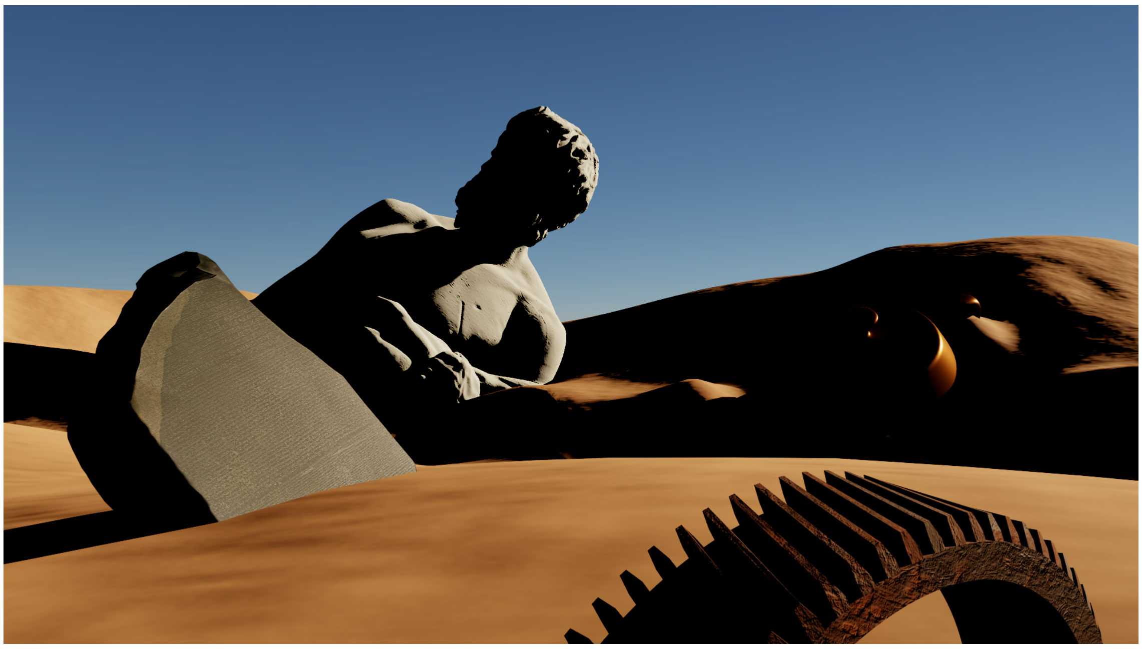


Figure 2.2. *An example scene with only direct lighting.*

a depth image from the light's point of view, which can then be used in a fragment shader later on to determine whether the shaded pixel is occluded to that light, i.e. in shadow or not. The naive implementation results in aliased edges in the shadow, but further filtering can be done to reduce this aliasing and to approximate soft shadows quite convincingly [12] [13].

2.2.2 Indirect Lighting

Indirect lighting, which is the part of lighting that does multiple bounces on the way from the light to the camera, is much more difficult for rasterization-based methods. This category encompasses such phenomena as reflections, refractions and color bleeding, all of which can be crucial to form the visual "look" of a material. Because of the difficulties in implementation, rasterization-compatible methods related to indirect lighting often use some degree of offline precomputation, resulting in the loss of dynamic interaction with some aspects of the scene. Figure 2.3 shows the contribution of indirect light in the same scene as Figure 2.2.

For simulating reflections, one method is to render a so-called environment map, which is a representation of the lighting around a point in space [11]. There can only be a limited number of these environment maps in each scene due to memory limitations, which results in somewhat incorrect reflections as points use the nearest available environment map that may not match with the actual lighting conditions. A typical modern approach to reflections would be Screen-Space Ray Tracing [14], which traces rays of light with depth information acquired from an earlier pass. This method is only limited to what has been

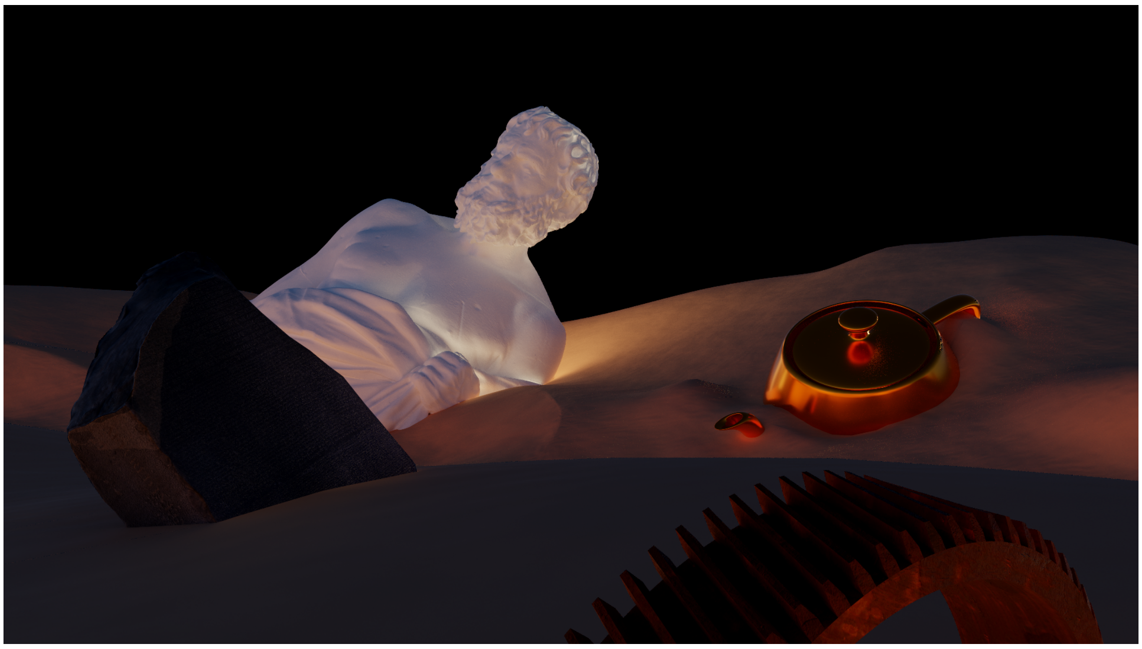


Figure 2.3. *An example scene with only indirect lighting shown.*

rendered in that earlier pass, which is typically only the things visible on-screen. This means that reflections of off-screen objects would not be visible. Planar reflections are an older method for reflections with fewer quality issues, but they are limited to planar surfaces and only support sharp reflections naturally [11].

Color bleeding, which is the visual end result of indirect lighting between colored diffuse surfaces, is very difficult to simulate in rasterization-based methods and is still mostly precomputed or very heavily approximated in interactive 3D content [11]. Even the real-time methods rarely work with pure rasterization alone. Instead, parts are implemented with ray tracing, cone tracing or some other method, using compute shaders. Voxel cone tracing is one example of such a method [15], though it is quite heavy and suffers from "light leaking", where light passes through walls even when it shouldn't due to limited precision. Reflective shadow maps are another method [16], but they suffer from several limitations as well: they too can be heavy to apply and have issues with visibility of indirect light (indirect shadowing).

2.2.3 Anti-Aliasing

Likewise, lots of anti-aliasing methods have been devised with rasterization in mind. While many operate as a post-processing effect, some more accurate methods exist. *Super-sampling Anti-Aliasing* (SSAA) operates by simply rendering the scene with multiple samples per pixel, and then averaging these samples to produce an anti-aliased image. However, this has a high performance cost. Often, the shading color for each sample is nearly the same.

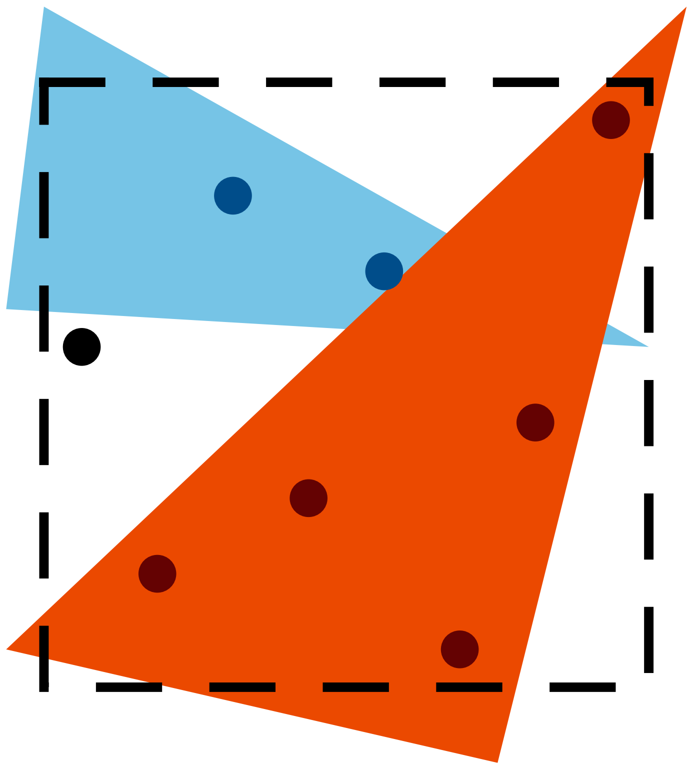


Figure 2.4. *Multisample Anti-Aliasing shades each polygon only once per pixel, but allows multiple polygons with partial coverage.*

Instead of shading each sample separately, *Multisample Anti-Aliasing* (MSAA) works by computing the coverage of each polygon for each sample, but only calculating shading once and applying it to all covered samples within the pixel for that polygon. This method is usually supported by graphics acceleration hardware for rasterization.

Figure 2.4 shows a visualization of coverage. In it, the blue triangle has a coverage of 2/8 and the orange triangle has a coverage of 5/8. The resolved color of the pixel would then be a weighted average of the triangles by their coverage.

While MSAA is usually an efficient solution, it is not helpful in cases where aliasing is caused by the shading itself. One typical case is shadows: sharp shadows always appear aliased even with MSAA, because each pixel is shaded only once and thus is either in shadow or not. There is no anti-aliased edge in the shading because effectively only one sample is taken.

2.3 Ray Tracing

In contrast to rasterization, ray tracing has only recently become a viable option for real-time 3D graphics, as hardware acceleration for the method has surfaced from multiple vendors during the last few years [2][17]. As the name suggests, ray tracing determines the intersections of a ray within the given scene, sometimes multiple times, continuing from the previous intersection in order to simulate certain aspects of lighting. The method does not require the scene to be defined in terms of polygons and can be implemented with any geometry representation for which a ray intersection can be computed, but triangle meshes are a very common geometry representation and discussion of ray tracing is limited to them in this thesis.

Ray tracing requires considering all geometry for each ray, which is why much research has been conducted towards making acceleration structures that allow disqualifying large swaths of triangles quickly. One such method is Bounding Volume Hierarchy, which constructs a hierarchy of quick-to-intersect volumes encompassing multiple subvolumes or triangles [18]. This can actually sometimes yield higher performance than rasterization when there are very high amounts of triangles [19], but the acceleration structures need to be updated whenever the geometry changes which can negate this benefit to an extent.

Because each ray can be individually evaluated, ray tracing does not impose limitations on projection unlike rasterization. This also makes shadows, reflections, refractions and even indirect light possible to simulate quite directly, though special consideration can still be necessary to make them relatively performant in certain conditions.

2.3.1 Path Tracing

Path tracing is one method based on ray tracing. It attempts to produce photorealistic images by following individual “photons” within the scene, bouncing around into different directions from each intersection. Monte Carlo path tracing is a variation that attempts to solve the rendering equation

$$I\left(x,x'\right) = g\left(x,x'\right)\left(\epsilon\left(x,x'\right) + \int_S \rho\left(x,x',x''\right) I\left(x',x''\right) dx''\right), \tag{2.1}$$

where $I(a,b)$ is the intensity of light transported from b to a, $g(a,b)$ determines whether there is an occlusion between a and b, $\epsilon(a,b)$ is the intensity of light emitted by the surface at b towards point a, $\rho(a,b,c)$ determines the fraction of light that passes from c to a through b and is derived from the BSDF. S is the set of all points on all surfaces and $x, x', x'' \in S$. As can be seen from the integral, this equation is recursive and in practice, is very rarely analytically solvable. Therefore, numerical integration methods are used instead. In this thesis, we use *Monte Carlo* integration

$$F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}, \tag{2.2}$$

where F_N is the approximated value, N is the number of samples taken, X_i is a sample taken from a distribution whose probability distribution function is $p(x)$. $f(x)$ is the value

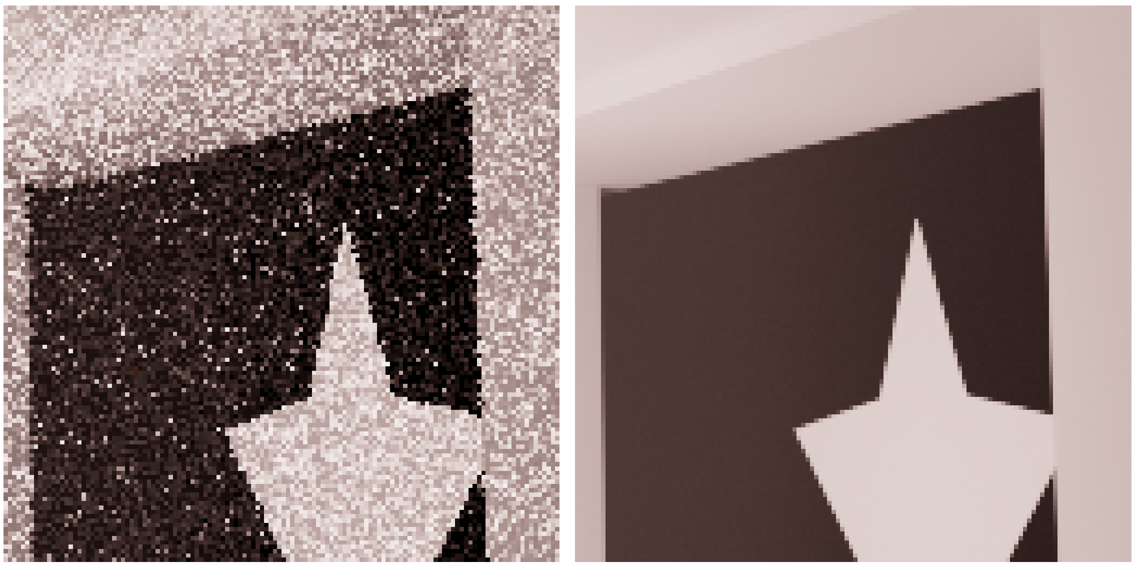


Figure 2.5. *The visual difference between 1 and 16384 samples per pixel in Monte Carlo path tracing.*

of the integrated expression. As N grows, F_N converges towards $\int_A f(x)dx$ where A is the domain of values that each sample is taken from.

In practice, this means that each bounce direction along the path is picked randomly from a probability distribution. Images produced with Monte Carlo integration appear noisy until they converge. Figure 2.5 exemplifies this difference between the number of samples taken. *Importance sampling* is the act of picking the probability distributions in a way that follows the sampled value from sample, which generally makes Monte Carlo integration convergence faster than picking the samples uniformly [20].

Path tracing is able to naturally simulate almost all visually important aspects of lighting. Relativistic effects and diffraction cannot be modeled with plain path tracing as the rays no longer travel in straight lines in those cases, though some extensions do exist for both. On the other hand, it is typically not possible to render enough samples for the Monte Carlo to converge to a visually noise-free image in real-time with current hardware. Depending on scene and resolution, the maximum number of samples per pixel that current hardware can deliver at real-time framerates is in the order of 1-10, whereas the noise becomes mostly invisible generally around 1000-10000 samples per pixel. Because of this, fewer samples are typically used and fed to a noise removal filter, which often causes loss of high-frequency lighting details.

3 LIGHT PROBES

A *light probe* is a representation of the lighting around a specified point in space. When they contain high-resolution information, light probes are also often called *environment maps* because they act as a globe of the surrounding environment. This information can then be used to approximate lighting for nearby surfaces. A particular representation of a light probe is called a *basis*. One common and easy-to-understand basis is the *cubemap* basis, in Figure 3.1. This basis is often used in precomputed environment maps. Another basis often used when distributing photographically captured environment maps is the equirectangular projection, shown in Figure 3.2.

High resolution bases are useful when the probes are used to simulate sharp, detailed reflections, but very wasteful when shading rough surfaces. For this use-case, several bases exist. The *ambient cube* basis is an extreme simplification of the cubemap basis, storing only one color per face. This method, however, experiences quite noticeable quality loss even for the rough surfaces [21].

Traditionally, light probes have been used to precalculate lighting in the scene. Another common method is *lightmapping*, which works by precalculating a texture of the diffuse illumination that the covered objects receive. While lightmapping can cover the diffuse lighting for static objects, it usually cannot contribute to specular lighting or to the lighting of dynamic objects. Probes are then used to cover those cases. Cubemap environment maps have been used for specular lighting and spherical harmonics probes for the diffuse lighting of dynamic objects. Both data structures are such that computing the lighting for a given vector is very quick, which is why they are useful for real-time rendering.

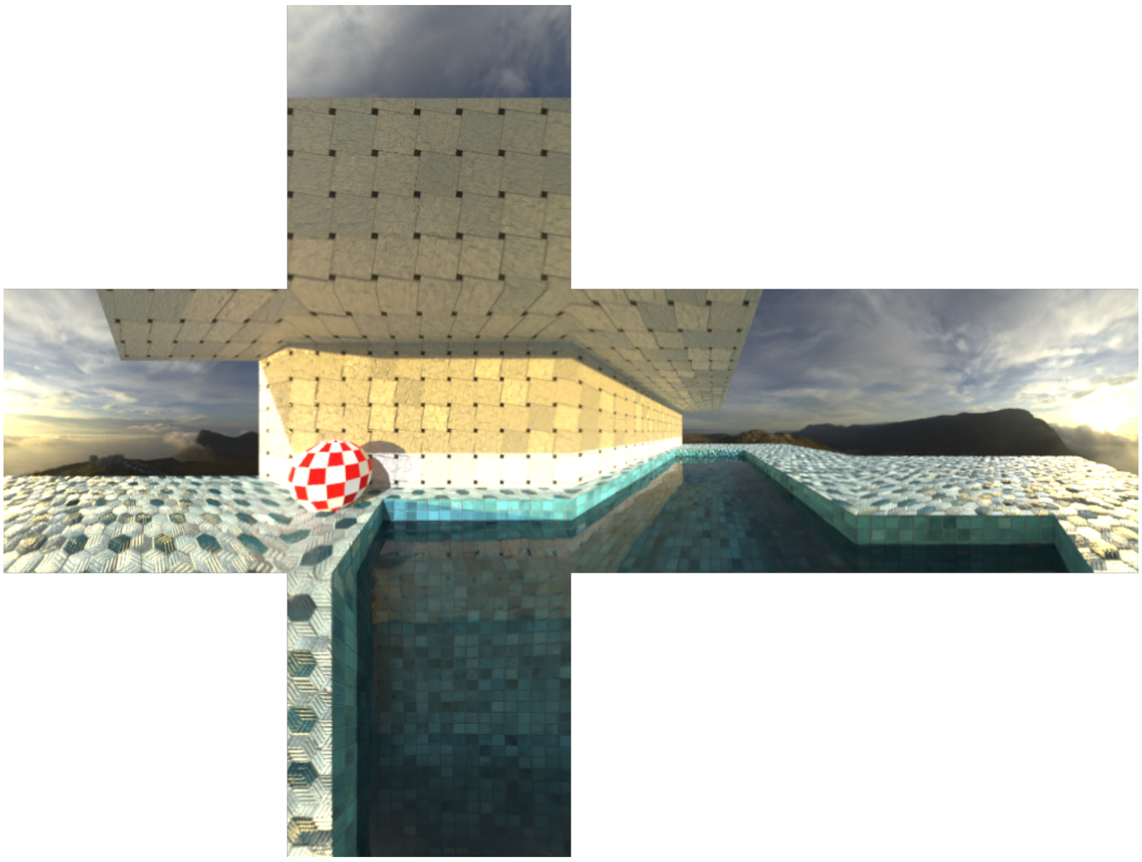


Figure 3.1. *A cubemap environment map, shown as an unfolded cube.*

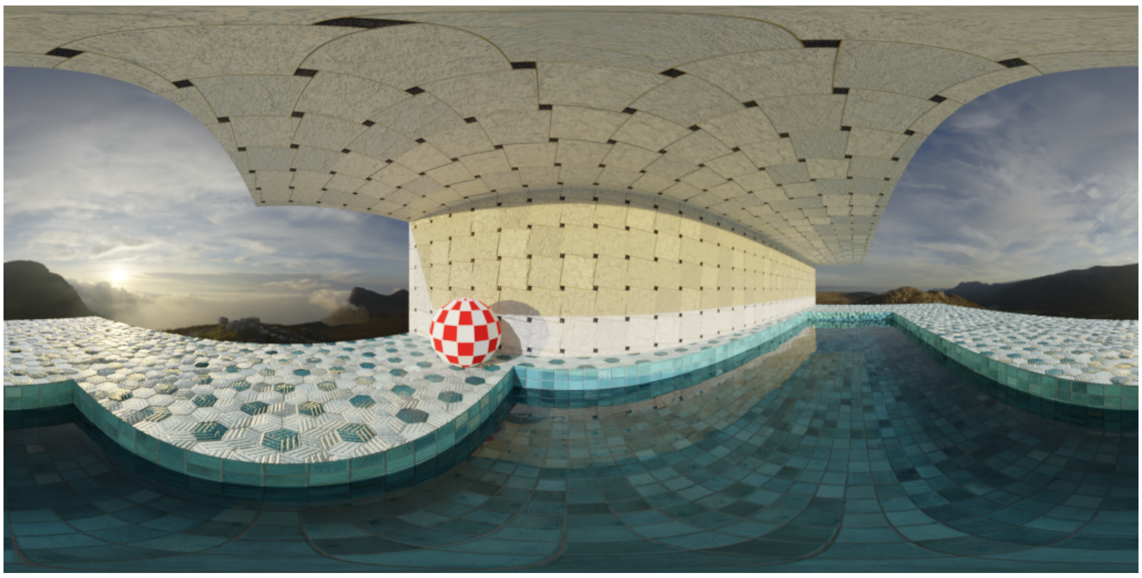


Figure 3.2. *An equirectangular environment map.*

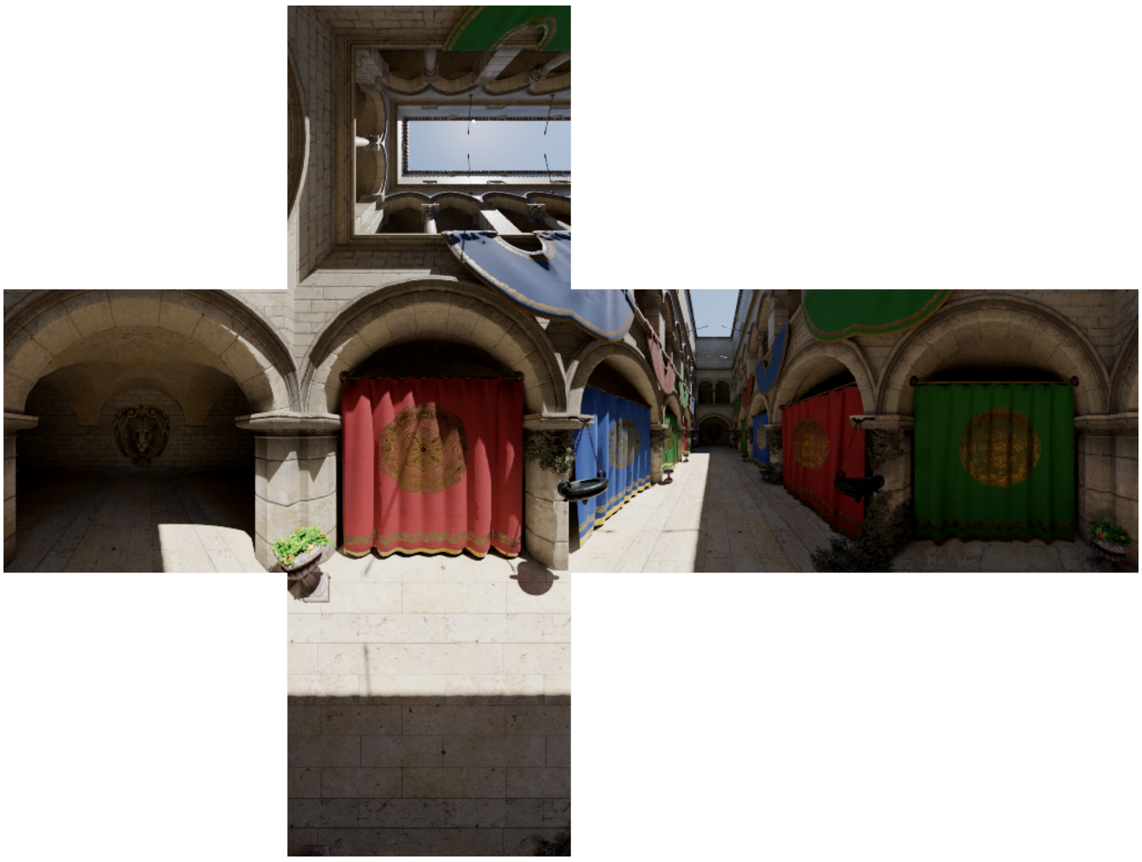
3.1 Spherical Harmonics

Spherical harmonics (SH) probes have a long history in real-time computer graphics, where they have been used to represent environment maps since [22]. They have found much use especially in video games, where there is an opportunity to precalculate lighting for large and static scenes with only few moving components. SH probes are especially good for modeling indirect diffuse lighting: the L2 representation is very compact and yet their average error is always below 3% for irradiance [22]. It is also very compact: when using 16-bit floating point values per coefficient and assuming that a multiple of 4 bytes are needed for memory alignment reasons, the L2 SH representation is only 56 bytes. A cubemap representation at a resolution of 256x256 per cube face would weigh in at ~2 megabytes.

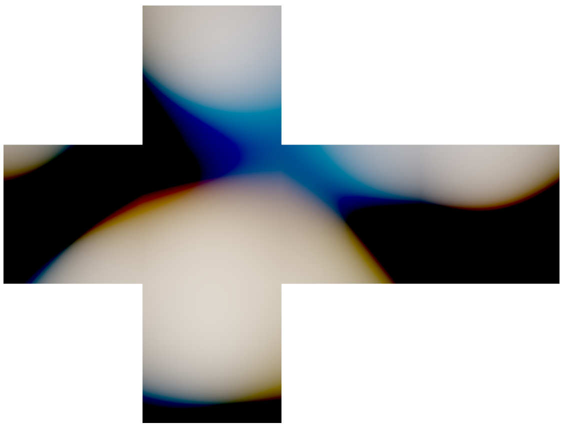
Spherical harmonics can be used to represent lighting information in a bit less direct manner. It is similar to how frequency space representations with 1D signals work, but is defined on a spherical surface instead. This brings many performance and quality benefits. Convolution of a probe by a lobe is very fast and is a very common use case when computing lighting from a probe. Additionally, the representation is rotationally invariant, meaning that the orientation of the probe does not affect the value of the probe in any other way than just the rotation, i.e. re-orienting the probe is equivalent to rotating the sampling direction instead. This is not true for the cubemap and ambient cube bases due to the way they discretize directions into pixels. Low-frequency data can be stored with spherical harmonics without high-frequency artifacts such as those caused by linear interpolation in pixel-based bases. The spherical harmonics representation suffers from ringing, meaning that overshooting can occur near high-intensity areas. [23]

Figure 3.3 shows the lack of detail and strong ringing present in the L2 SH representation in a difficult case. The surrounding area has significantly large areas of both high and low-intensity light. The image has been tonemapped for display in a manner that does not fully represent the range of values. In linear colorspace, typical shady areas have color values around 0.005 in magnitude, whereas the bright ground has values greater than 1.5. Even though truly black color is very rare in the surroundings, the SH basis representation has areas which appear black. In fact, these areas had negative values which got clamped to zero. The result after a convolution needed for diffuse irradiance much closer matches a heavily blurred version of the cubemap probe.

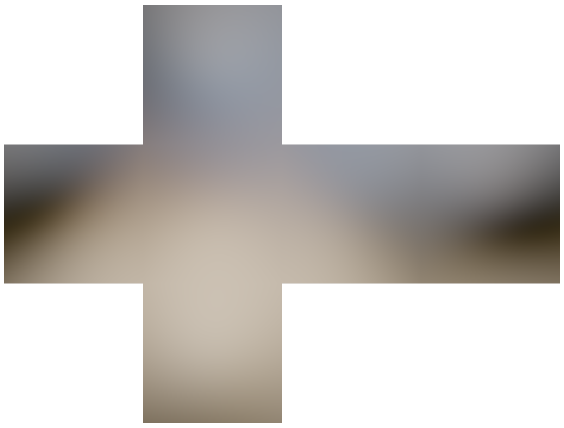
The light is represented by fitting a linear combination of spherical harmonics, shown in Table 3.1, to the lighting intensity. The process of doing so is called spherical harmonics projection. For each probe, only the coefficients for the linear combination are stored, as the spherical harmonics are always the same. The number of spherical harmonics used for the linear combination affects the precision of the lighting reconstruction. Table 3.1 shows the spherical harmonics of four bands; a probe using these would be called to be



***(a)** Cubemap basis*



***(b)** L2 SH basis*



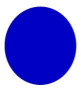
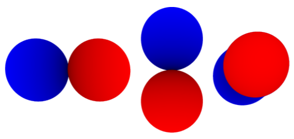
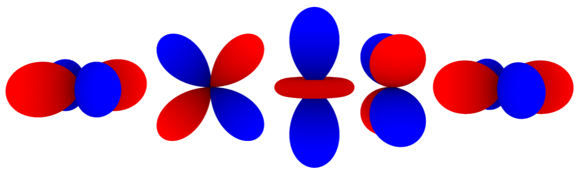
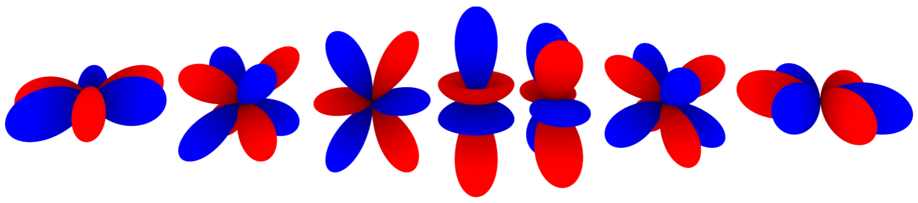
***(c)** L2 SH basis after convolution for irradiance*

***Figure 3.3.** The data loss in L2 spherical harmonics compared to a high-resolution cube-map basis.*

of order 3, or L3. Equivalently, an L2 probe would contain 3 bands and 9 coefficients. In practice, triple the number of coefficients are needed because color is encoded as separate light intensities for red, green and blue.

Because the spherical harmonics representation is simply a linear combination of the different spherical harmonics, linear interpolation of probe data is trivial as it is equivalent to simply interpolating the coefficients. This enables using hardware texture interpolation

Table 3.1. *A visualization of Spherical Harmonics, where blue is positive and red is negative magnitude. The central column is the Zonal Harmonics subset. Pictures generated using a modified version of [24].*

L0	
L1	
L2	
L3	

support to gain a performance benefit. The coefficients can be stored in a 3D texture from which they can be directly sampled and automatically interpolated for the desired point in space.

3.2 Zonal Harmonics

In Table 3.1, the central column contains the subset of spherical harmonics called *zonal harmonics*. This subset is used to represent rotationally symmetric functions. They have certain properties which speed up common calculations significantly compared to using all spherical harmonics. [23]

In particular, the ZH representation is often used to represent rotationally symmetric BRDF lobes. The convolution of an SH and ZH function can be performed significantly faster than SH-SH convolution. This is why using ZH approximations for lobes is beneficial for overall renderer performance.

3.3 Irradiance Volumes

Irradiance volumes are 3D grids of light probes that are placed in a 3D scene [25], usually by an artist. Because the data of an irradiance volume can be packed in a 3D texture, they allow for fast GPU interpolation of spherical harmonics light probes, which is vital when per-pixel interpolation is desired. They are also fairly compact in nature, because the

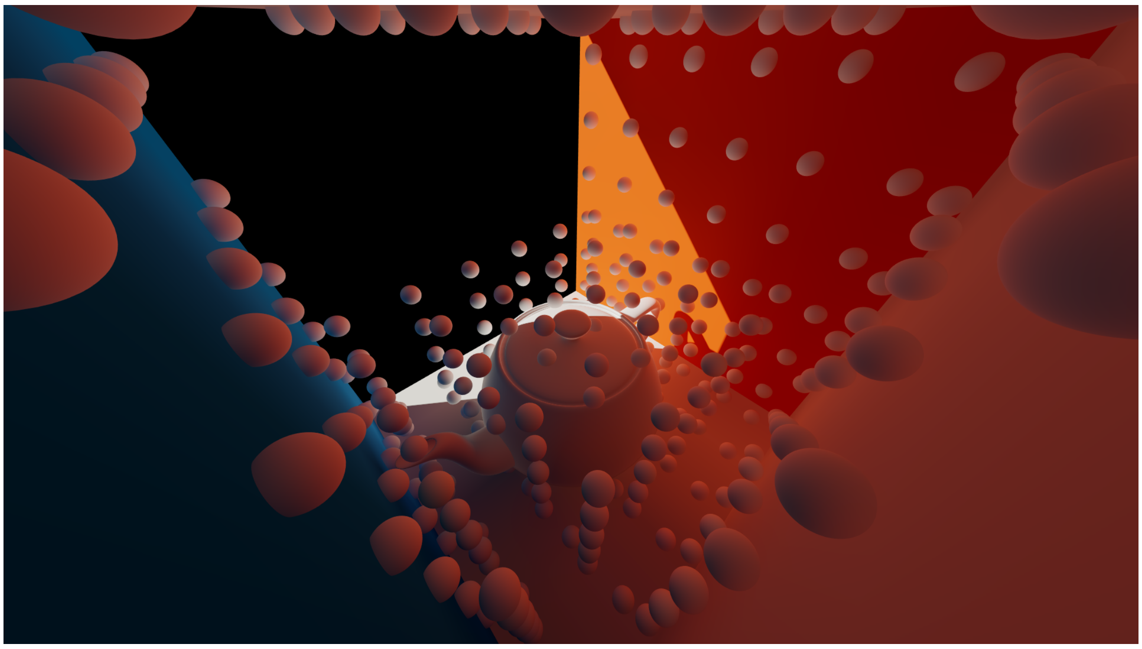


Figure 3.4. *Visualization of probe grids (irradiance volumes). In this scene, each wall has a 6x6x1 grid and the teapot area has a 6x5x6 grid. Each probe in the volume is visualized as a white lambertian sphere lit by the probe alone. In our system, the probes only carry indirect lighting data, which is why the visualizers do not react to the direct sunlight.*

data structure is simply a 3D array. Figure 3.4 visualizes the probes of multiple irradiance volumes in a single scene.

Compared to freely placing individual probes, it is hard to achieve a good placement for every probe in an irradiance volume. This can result in worse quality even when there are more probes. Often, probes end up inside objects, which causes issues with lighting conditions leaking through walls when the probe data is interpolated with naive trilinear interpolation. Workarounds, such as not rendering back faces of surfaces or introducing smarter interpolation with visibility information, help mitigate these issues.

4 IMPLEMENTATION

The proposed method is a hybrid rendering algorithm which uses a path tracing pipeline for computing indirect lighting and using the results with a classic rasterization pipeline. This is specifically done in such a way that minimizes communication and dependencies between the two pipelines, making this algorithm well-suited for distributed and remote rendering.

Figure 4.1 presents an overview of the pipeline structure. In the lime “transformation matrices” step, all transformation matrices which determine the state of the scene are calculated based on user input and animation. The dashed red step is the overall “SH update” part of the pipeline, and the dashed blue is the “SH usage” part. These are discussed in more detail in Sections 4.1 and 4.2, respectively.

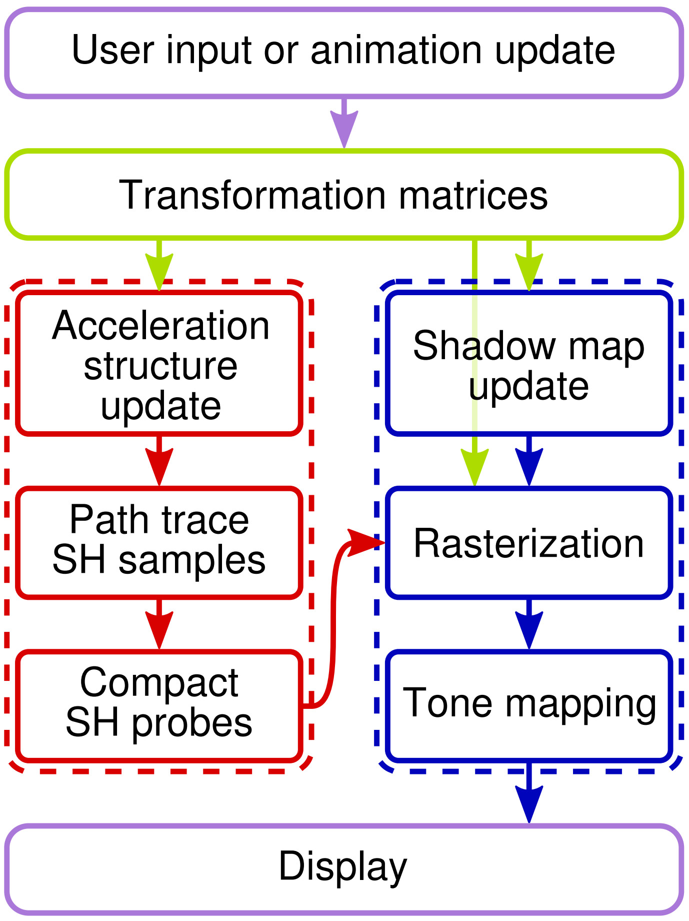


Figure 4.1. *The pipeline setup of the proposed method. The SH update (red) can run asynchronously relative to SH usage (blue) with few negative effects.*

4.1 Spherical Harmonics Probe Update

The parts of the pipeline responsible for keeping the spherical harmonics probes updated works entirely by path tracing. It is also quite configurable as the number of probes, number of samples per probe, amount of temporal accumulation and ray bounces can be adjusted to suit the needs. Further, it does not need to run synchronously with the pipeline stages described in Section 4.2.

These properties lend the algorithm great potential for distributing the computation across multiple devices; a relatively weak client device could leverage the power of a remote but powerful server. The data that needs to be transferred from this stage to the client device is very compact and latency-resilient.

4.1.1 Path Tracing

The path tracing implementation in the proposed method uses a recent Vulkan extension, VK_KHR_ray_tracing_pipeline [26], to enable hardware acceleration of the ray tracing workload. The implementation is very focused on performance but is still in its early days and therefore not extensively optimized yet.

In the framework of the extension, acceleration structures are divided into bottom-level (BLAS) and top-level acceleration structures (TLAS) [27]. The former contains geometry such as triangles, and the latter consists of these bottom-level acceleration structures with an applied matrix transformation. The TLAS is updated for each frame since doing so is quite cheap; the tested scenes do not contain deforming geometry which is why BLAS's are not updated. BLAS updates could be somewhat costly.

To simplify shading, a material model matching the glTF 2.0 specification [28] is used. The specular and transmission components are based on the GGX model [1] and the diffuse component is the simple lambertian model. The resulting BSDF is then used for importance sampling. Next-event estimation is used, which significantly reduces noise in directly-lit areas but roughly halves performance.

Path tracing is used rather directly to update the SH probes. For each probe position, a ray direction is generated and then path traced. The resulting brightness value and direction are then projected into an SH representation. Multiple samples are then averaged in order to cover the entire SH.

Path tracing with low sample counts can result in some noise, which is still the case for SH probes. While at first it may seem that a low-frequency representation like an SH would only need few samples to work, it must be kept in mind that an SH probe still carries much more information than a single pixel would and thus requires significantly more samples to appear noise-free than a single pixel would. Even thousands of samples per probe causes distracting flickering.

To alleviate the noise issue, a couple of techniques are used. Firstly, ray directions for a probe are not generated fully randomly; instead, samples are picked from a Fibonacci lattice which approximates evenly distributed samples on a sphere [29]. Secondly, results from the previous pass are blended with the new ones using a constant ratio. This kind of temporal averaging does not carry other issues than outdated information if objects in the scene move. Since the probes are independent of the camera used for viewing the data, specular details dependent on view direction are not smudged by camera movement like commonly seen in temporal reprojection methods. The Fibonacci lattice could be randomly oriented for each pass so that as many directions as possible would be covered in the temporally blended SH probes, but doing so has virtually no effect with the sample counts used in the test scenes.

Particularly difficult rays can cause the equivalent of the "fireflies" artifact, which appears as sudden large-scale flashes in the SH representations. Those particularly bright samples are caused by paths that are not very likely to be sampled and that would yield a high contribution to the sample's lighting, such as those created by caustics. At the cost of some realism, clamping lighting contribution of each bounce into a limited range can significantly reduce or even eliminate these issues.

The probe update presents a parallelization challenge: typical grid volume sizes such as 10^3 are not enough to saturate the GPU completely and looping to calculate hundreds samples per probe results in both low GPU utilization and long-running kernels. Instead of looping, each projected SH sample is stored separately in a buffer. These samples are then averaged into the final SH probe in a separate merge step that implements a parallel reduction. The final SH probes are stored in a 3D texture for easy sampling. Since there are more coefficients than available channels per voxel, the coefficients are packed into the texture in layers; one block matching the volume size contains the first 4 coefficients, another one that is placed next to it contains the next 4 and so on.

The resulting volume is all of the information that needs to be given to the rasterization pipeline. It is quite compact, a 10^3 grid of L2 probes using 16-bit floating point values would take roughly 56 kilobytes. Furthermore, the data is independent of camera location and angle and is therefore somewhat resilient to a high delay between the update and usage at the rasterization, allowing highly asynchronous updates.

4.2 Rasterization and SH Probe Usage

In the proposed method, the final image is rasterized. Most GPU acceleration hardware is still highly geared towards rasterization and is very efficient at it, which is why it presents a clear performance benefit over ray tracing the entire scene. Since the ray tracing itself is decoupled from this step, the resolution of the viewport does not affect ray tracing performance and the number of samples taken for path tracing does not affect rasterization performance.

The rasterization part of the pipeline is responsible for producing direct illumination and adding indirect illumination using the SH probes which are generated as described in Section 4.1. Rasterizing the direct illumination is not straightforward but can be approximated quickly to a plausible degree of realism, with the main obstacle being shadows.

4.2.1 Multisampling

The Vulkan API exposes an option for “Sample Shading”. This option can be used to force the implementation to compute shading for a portion (or all) of the taken samples separately, which reduces aliasing in the shading itself. This essentially turns MSAA into SSAA, though the ratio of shading samples to coverage samples can be somewhere between 0 and 1 if desired. It can be significantly slower, especially when the shading is complicated as the number of shading calculations per frame is multiplied by the number of samples taken per pixel.

MSAA can also be used to implement order-independent alpha blending, which is done in the proposed method. This is achieved through another API feature called alpha-to-coverage, which uses the output transparency value to adjust the ratio of samples which are covered by the triangle for each pixel. The number of samples thus affects the precision at which alpha blending occurs. To avoid this causing aliasing, the hardware used for the results implements alpha-to-coverage with dithering, where certain pixels are weighted more than others to better visually match the desired shading.

4.2.2 Applying Spherical Harmonics Probes

The coefficients of the spherical harmonics probes updated by the path tracing step can be sampled from the 3D textures they are saved in. This sampling typically has some degree of hardware acceleration, and is thus often beneficial compared to accessing an array of SH data directly. Since spherical harmonics coefficients can be linearly combined, per-coefficient trilinear interpolation is a completely valid way to sample them.

No windowing method is used in the proposed method. While it could be beneficial in some cases to reduce ringing artefacts, the visual effect of windowing is also quite noticeable and potentially worse than ringing. Diffuse lighting is implemented using the method presented in [22].

An implementation for specular lighting with spherical harmonics probes has been presented in [30] but is quite heavy, requiring multiple lookup texture taps and SH rotation. Furthermore, the tighter convolution lobe causes the low-frequency nature of SH data to become apparent. Instead of using such a heavy method for an effect which cannot be very accurate anyway because of SH limitations, we propose applying the *split sum approximation* to spherical harmonics in order to simplify and speed up the operation.

The split sum approximation is a method for splitting the integral part of the rendering equation 2.1 in such a way that incoming light is integrated separately from the surface response to it [9]. In its original use case, the split sum approximation focuses on cubemap-basis environment maps and allows for precomputation of both halves of the split but forces symmetric specular lobes. The application of this method to spherical harmonics probes is novel in this thesis, and allows for them to be used for specular lighting in addition to the more traditional diffuse component.

Unlike in the original use case of the split sum approximation, we cannot precalculate the convolution because our probes are dynamically updated. This requires modeling the GGX specular lobe with an SH approximation. Because the split sum approximation forces isotropy in the lobe, the projected SH representation ends up only using zonal harmonics (ZH), which are the axially symmetric subset of spherical harmonics that are even faster to convolve with. There is only one ZH coefficient per SH order, which is why

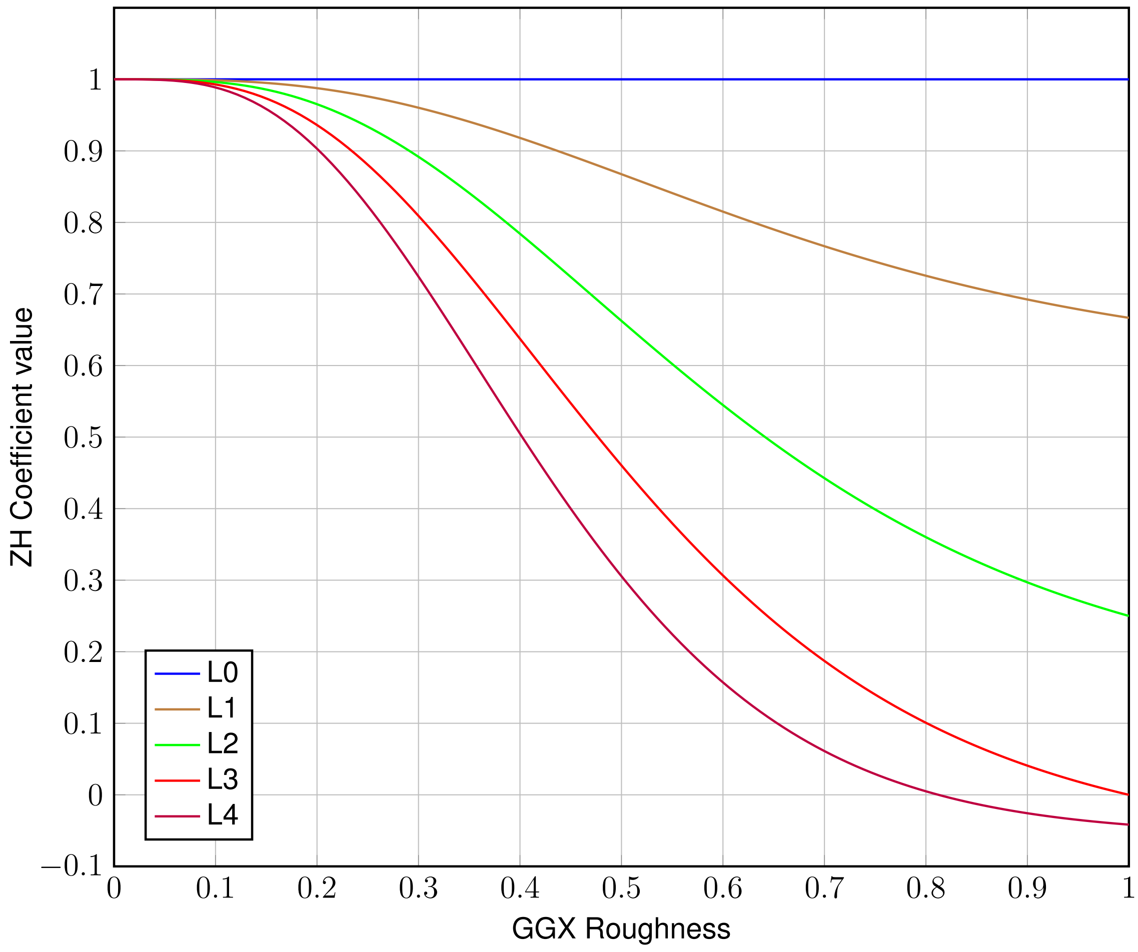


Figure 4.2. *Values of the precomputed ZH coefficients for approximating GGX lobes.*

referring to them with the order name (L0-L4) is unambiguous.

The projected lobes for 1024 roughness values were calculated numerically, then a curve was fit to these so that the required ZH coefficients can quickly be computed for a lobe with a given roughness. This curve elegantly fades from a lobe representing impulse response to a lobe representing a clamped cosine lobe. Figure 4.2 displays the numerically calculated lobe ZH coefficients as a function of the GGX roughness parameter. See Appendix A for the resulting shader code for the L2 case.

Since direct lights are explicitly excluded from the probes in the update step, the probes only provide the indirect component. Direct lighting is computed by modeling the light sources as infinitely small, then evaluating the BRDF directly. This loses some detail with larger lights; particularly, specular highlights become infinitesimal in perfectly specular surfaces.

4.2.3 Shadow Mapping

Because direct lights are calculated with traditional rasterization in the proposed method, shadow mapping was chosen. If used naively, the visibility tests done in shadow mapping would cause heavy aliasing along the shadow edge. Percentage Closer Filtering (PCF) [31] is used to soften the edges. To simulate soft shadows caused by area lights, Percentage Closer Soft Shadows (PCSS) [12] are used to determine the softening radius for PCF. These methods are simple to implement but fairly heavy and not necessarily optimal for this use case. Moment Shadow Mapping is a more recent method that could potentially be more efficient for rasterized shadow mapping [13].

To improve performance while having high-resolution shadow maps near the viewer, cascaded shadow mapping is used for directional lights such as the sun. This can cause some artefacts near the border between cascades. A small guard band is automatically determined based on the solid angle of the directional light so that the PCF radius doesn't sample beyond a single cascade very often.

4.2.4 Tone Mapping

All lighting calculations are done in linear color space. However, modern display devices expect non-linear colorspaces. Further, values can easily exceed the "maximum brightness" of 1.0 and thus appear overexposed. Tone mapping can be used to work around these issues.

Figure 4.3 shows the transformation caused by several different tonemapping methods which convert colors from linear color space into something else. The sRGB color space is often expected by display devices. Gamma correction is a fast approximation for the sRGB correction and is used very often because of that. Displays and printers expect the given value to be in the 0-1 range, and thus the gamma and sRGB corrections would lose detail on all color values greater than 1. The filmic tone mapping method always keeps the values in the valid range, no matter what the linear color value is.

For the PSNR results of this thesis, simple gamma-correction was used in the tone mapping, with no clamping. For shown images, a filmic tonemapping operator was used since it compresses intensity values exceeding one into the 0-1 range that can be printed while preserving details.

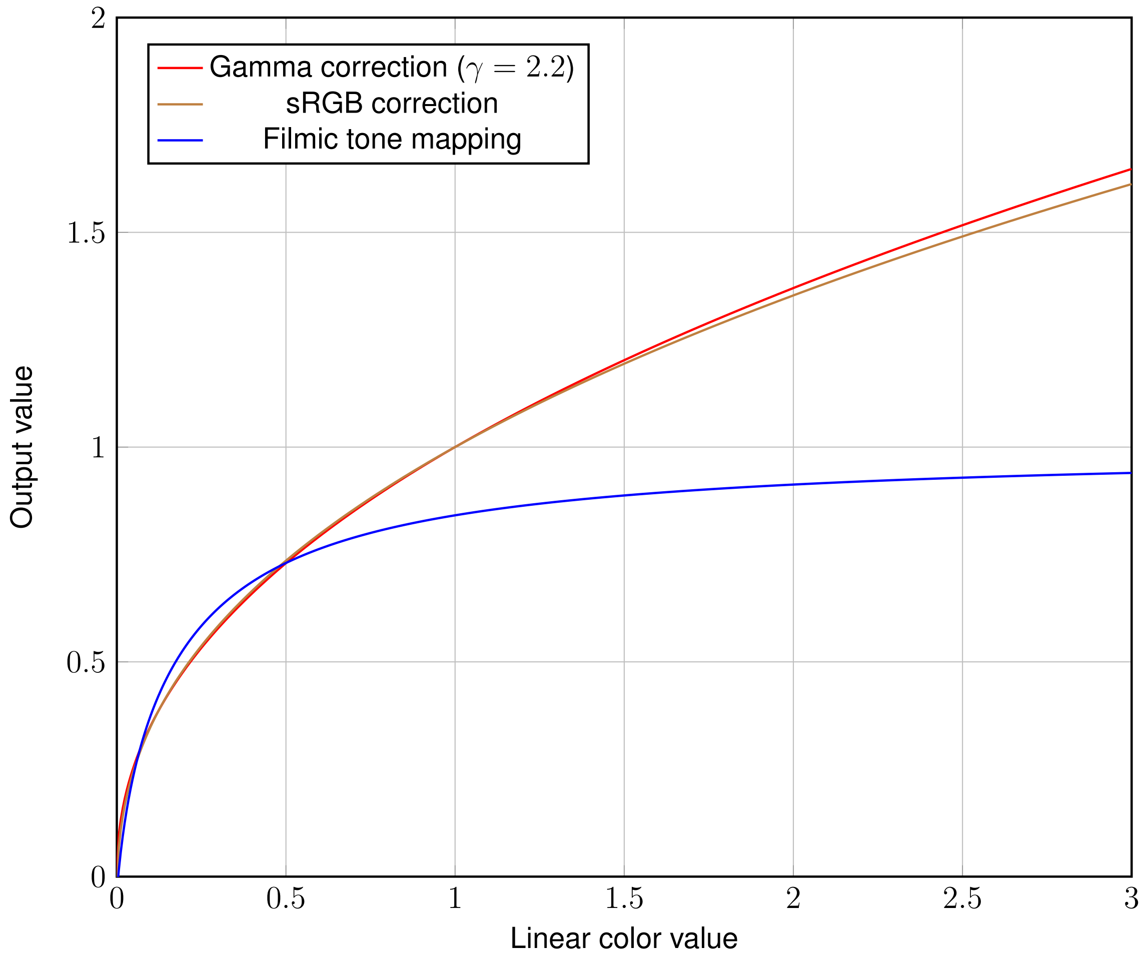


Figure 4.3. *Color remapping caused by different tonemapping algorithms.*

5 RESULTS

The quality of the proposed method is evaluated in three scenes: the “Breakfast room” scene [32], shown in Figure 5.1; the “San Miguel” scene [32], shown in Figure 5.3; and the “Sponza” scene [32], shown in Figure 5.5. The performance characteristics for the scenes are categorized in Tables 5.1, 5.2 and 5.3. Materials in the scenes are redefined because the original materials are not compatible with the physically-based material model used in the renderer. Lastly, Figures 5.7 and 5.8 contain graph which display the frametime and quality scaling, respectively, in the “Breakfast room” scene.

These scenes and the lighting conditions and camera angles in them are selected to feature indirect lighting as a significant part of the lighting. This lets the quality comparison focus on the core problem tackled by the proposed method, instead of relying on direct lighting which is handled by the rasterizer component alone in the proposed method.

All images are rendered at a resolution of 1280x720 pixels, with the exception of those used in Figures 5.7 and 5.8 where the resolution varies and the aspect ratio is square. Reference images are generated by path tracing using 4096 samples per pixel and a maximum of 4 ray bounces per path. “Fireflies”, i.e. extremely bright pixels caused by high light contribution through an improbable path, can be a major source of noise that does not easily converge. To mitigate them and the issues they cause in all compared methods, the contribution of indirect light bounces is clamped to an extent that visually removes fireflies but does not significantly affect the brightness of the image.

In addition to the reference, the proposed method is compared to denoised path tracing results in order to outline the unique performance characteristics of this method. The BMFR algorithm, which is one of the state-of-the-art real-time denoising methods tailored for 1spp path tracing, is used [33] for the denoising comparisons.

Dynamic Diffuse Global Illumination (DDGI) is a rendering method quite similar to the proposed method; it uses a different probe basis, more advanced probe interpolation and only handles diffuse lighting with the probes [34]. Even though it is the closest related work to this thesis and comparison to it would be desirable, it is omitted from this thesis due to the amount of work related to replicating the method in the same renderer. Furthermore, the DDGI paper lacks objective quality measurements for the method.

For the proposed method, the probe grids are placed manually in each scene. In “Breakfast room”, the grid contains $8 \cdot 4 \cdot 8 = 256$ probes. In “San Miguel”, the grid contains $8 \cdot 8 \cdot 8 = 512$ probes. Finally, in “Sponza”, the grid contains $10 \cdot 6 \cdot 7 = 420$ probes. This resolution and the placement of the probe grid is chosen to produce reasonable results. In all scenes, 512 path traced samples are calculated per SH probe for each frame, and these are blended with the probes from the previous frame with a factor of 0.01. That value for the factor is quite extreme and is a strong cause for delay in the lighting. More samples or more aggressive firefly clamping would be needed to make higher blending factors flicker-free. While these sampling parameters are selected conservatively to avoid all visual flickering while using the same parameters in all scenes, they are still quite arbitrarily chosen and should be tweaked based on the specific needs in a real use case.

L2, L3 and L4 spherical harmonics probes are evaluated for the proposed method. The visual results of L3 and L4 are exceedingly similar to the L2 results and are thus omitted from the comparison figures.

For the comparison images and performance tables, particularly heavy but high-quality settings are used in the rasterization component. 8x multisampling and full sample shading are used, meaning that every pixel is shaded 8 times. These parameters fully eliminate aliasing both in geometry and in shadow boundaries. Sample shading is disabled for the graph in Figures 5.7 and 5.8, so only geometric edges are antialiased but performance is significantly better. In all cases, shadow filtering uses 16 samples for PCF and 16 samples for PCSS blocker search.

Because BMFR is not integrated into the renderer, all BMFR results are computed by first rendering the buffers required for the default settings of BMFR using our renderer, then feeding those into the standalone BMFR implementation [35]. Because of this, the timing results are computed by summing the time taken to render the 1spp path traced input image and the average filtering time reported by the BMFR implementation.

The BMFR results are compared to a reference image with the exact same setup as the aforementioned reference, but with antialiasing disabled. This is done because TAA jitter, the feature needed by BMFR for antialiasing, is not yet implemented in the renderer at the time of writing. Comparisons to the antialiased reference would cause an unfair disadvantage for the BMFR method.

Quality measurement is done by rendering 60 frames with a static camera and scene and comparing the last frame to the reference using the PSNR metric. The PSNR is calculated after gamma correction is applied. The purpose of rendering multiple frames preceding the examined frame is to let temporal computation reach equilibrium. All timing results are averaged over 50 frames, the first and last few frames are excluded because they may contain initialization and image saving delays, respectively. The reference images are an exception to this, only one frame is rendered per result image since there is no temporal component in them. Likewise, their frametime is measured from only that single frame.

Table 5.1. *The performance details of the "Breakfast room" scene.*

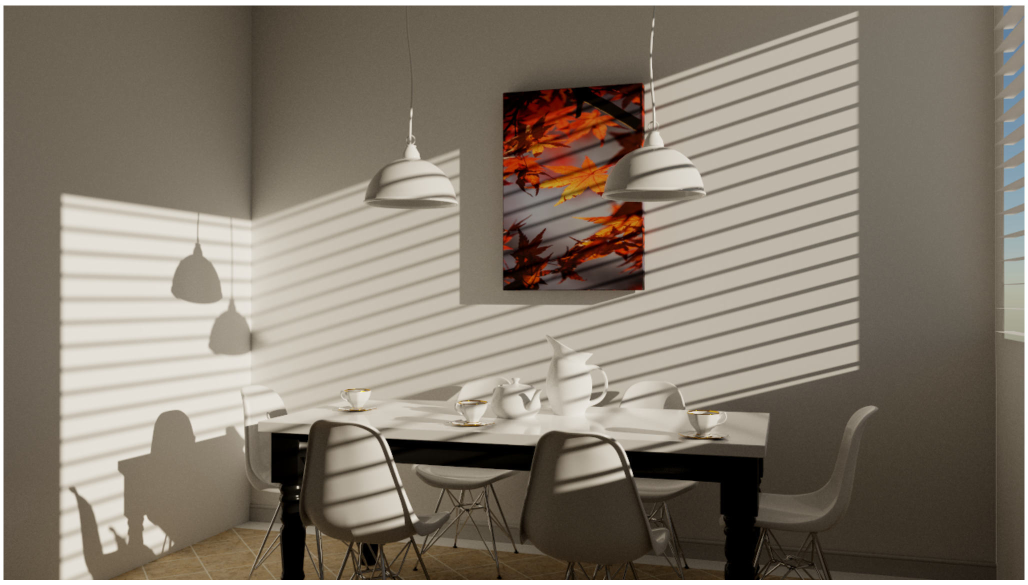
Method	PSNR (dB)	SH update (ms)	Rasterization (ms)	Total (ms)
L2 SH	25.80	3.92	5.35	10.88
L3 SH	25.78	5.11	55.50	62.04
L4 SH	25.84	10.89	41.53	53.73
1spp PT + BMFR	22.51	N/A	N/A	13.94
4096spp PT	N/A	N/A	N/A	59077.1

Table 5.2. *The performance details of the "San Miguel" scene.*

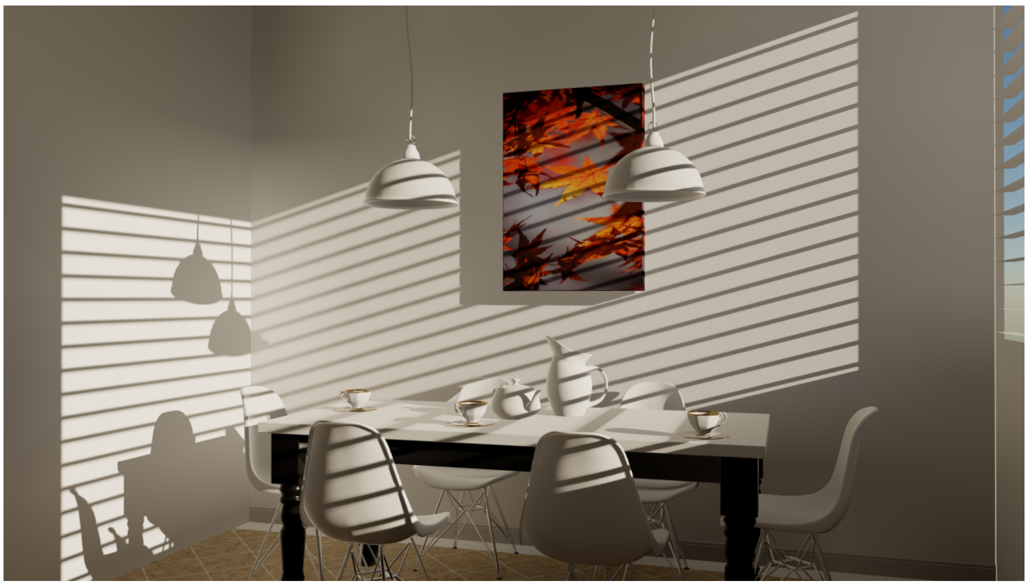
Method	PSNR (dB)	SH update (ms)	Rasterization (ms)	Total (ms)
L2 SH	27.93	13.17	20.05	38.88
L3 SH	27.92	13.70	225.95	244.86
L4 SH	27.88	17.93	176.01	198.97
1spp PT + BMFR	24.25	N/A	N/A	35.286
4096spp PT	N/A	N/A	N/A	151483

Table 5.3. *The performance details of the "Sponza" scene.*

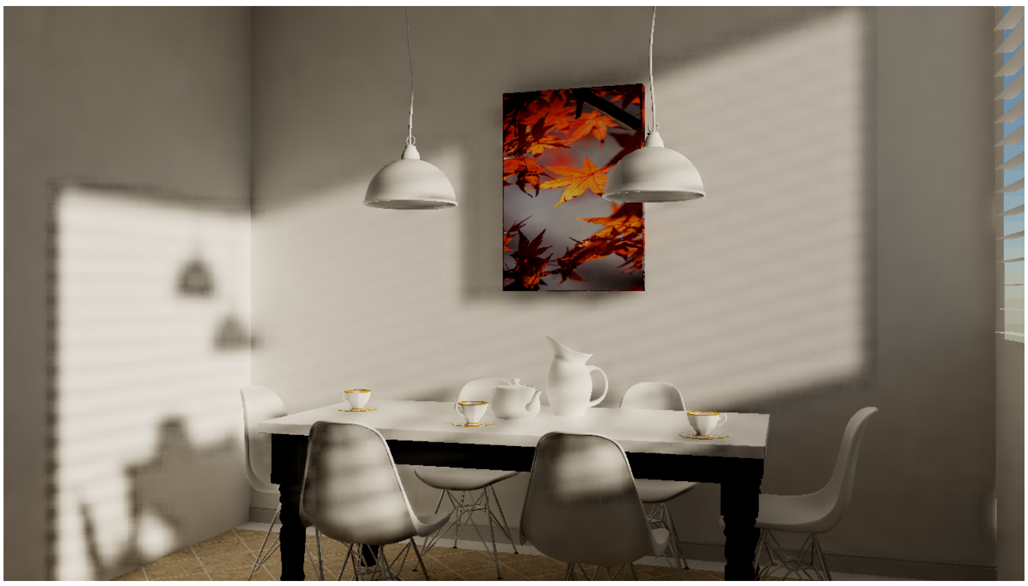
Method	PSNR (dB)	SH update (ms)	Rasterization (ms)	Total (ms)
L2 SH	25.02	5.89	10.36	18.50
L3 SH	25.05	6.82	95.50	104.32
L4 SH	24.99	13.4	74.21	89.33
1spp PT + BMFR	18.99	N/A	N/A	15.362
4096spp PT	N/A	N/A	N/A	151483



(a) *4096spp path traced reference*

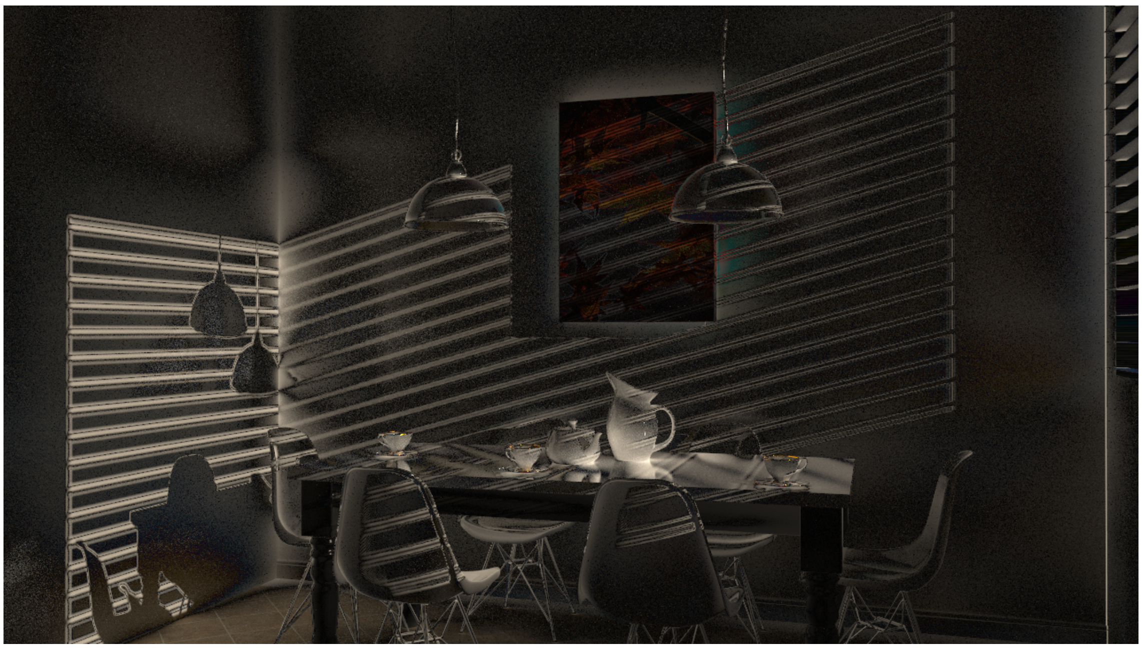


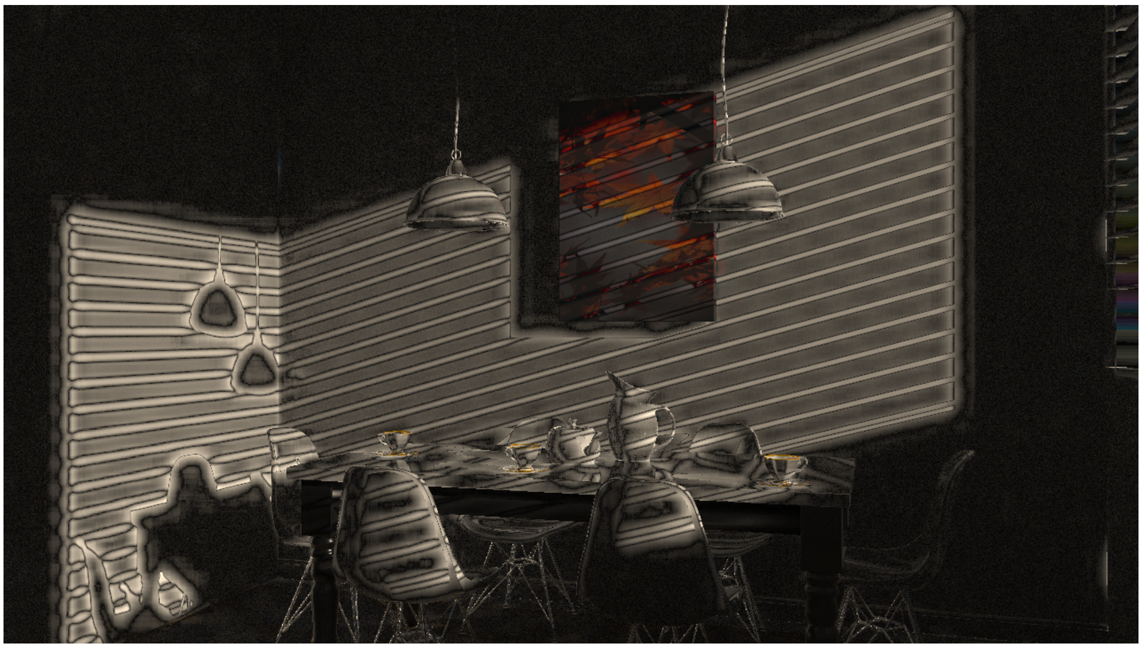
(b) *L2 spherical harmonics (25.80 dB)*

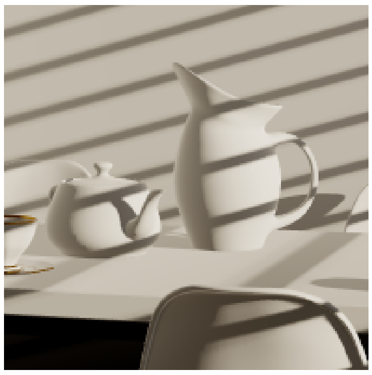


(c) *1spp PT + BMFR (22.51 dB)*

Figure 5.1. *Quality measurement of the methods in “Breakfast room” (PSNR).*


(a) *L2 difference to reference*


(b) *1spp PT + BMFR difference to reference*


(c) *L2 closeup*

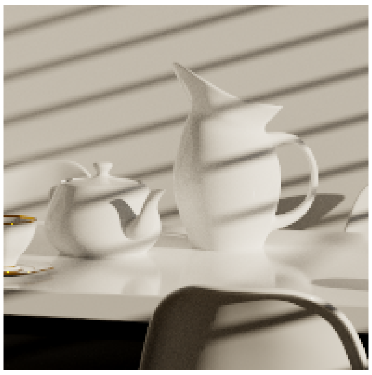
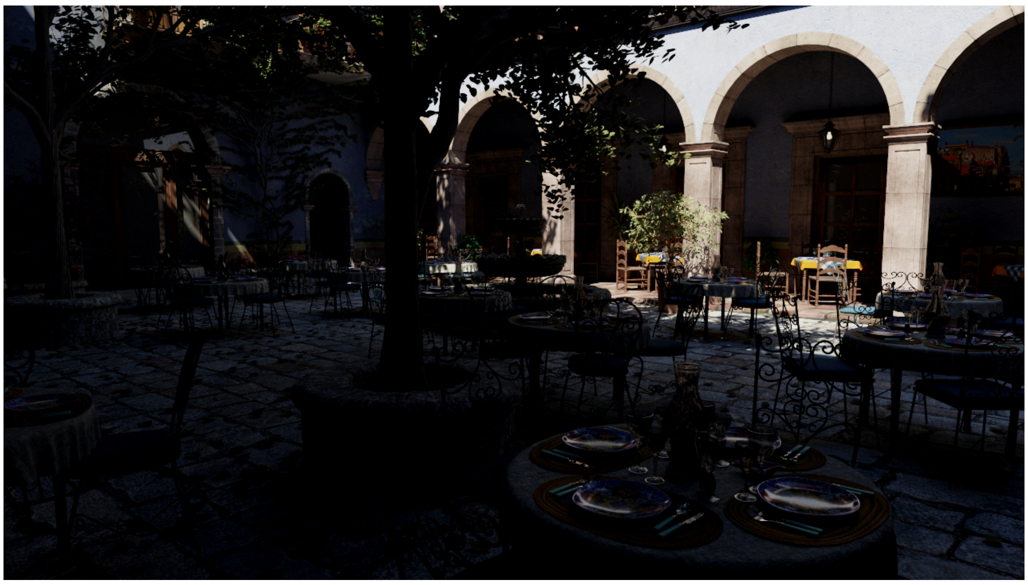
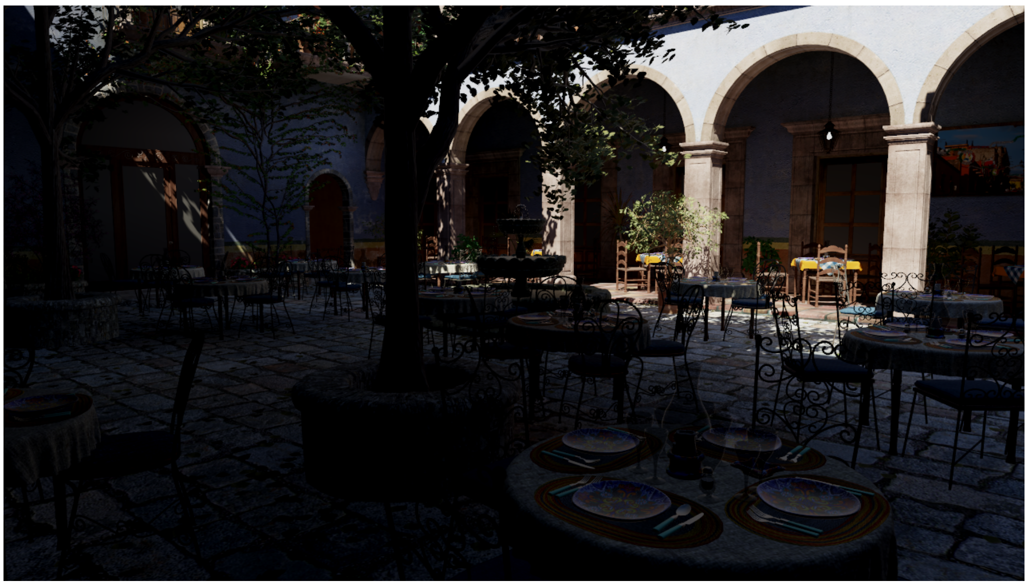

(d) *Reference closeup*

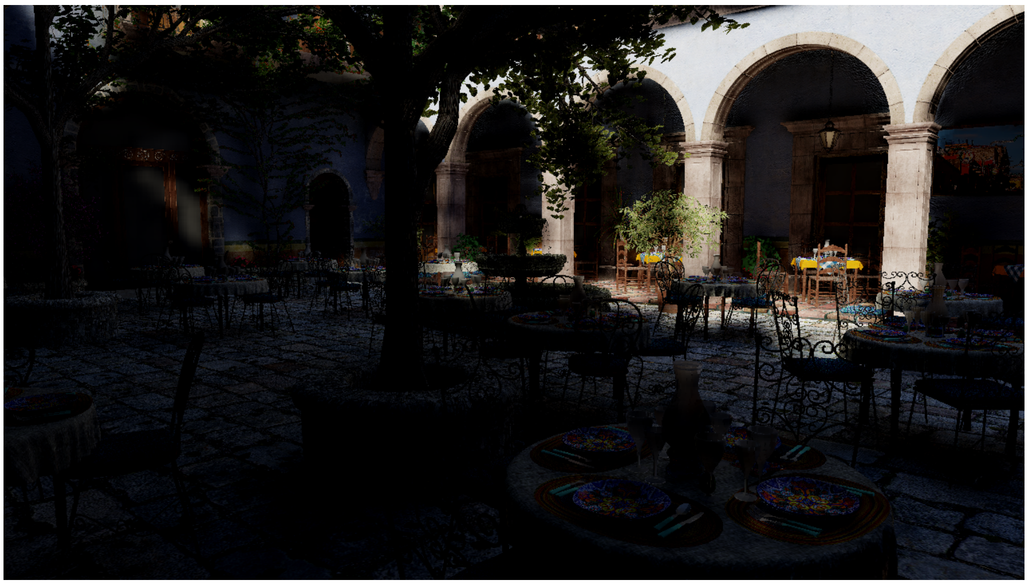
Figure 5.2. *Quality comparison between the methods in "Breakfast room".*



(a) *4096spp path traced reference*

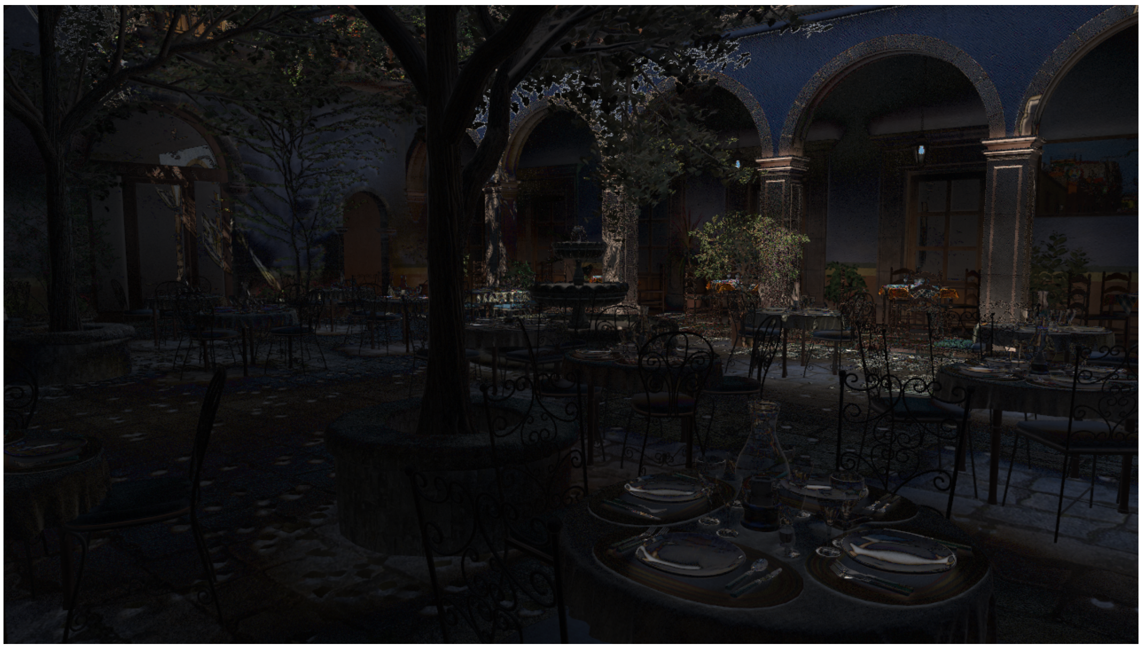


(b) *L2 spherical harmonics (27.93 dB)*

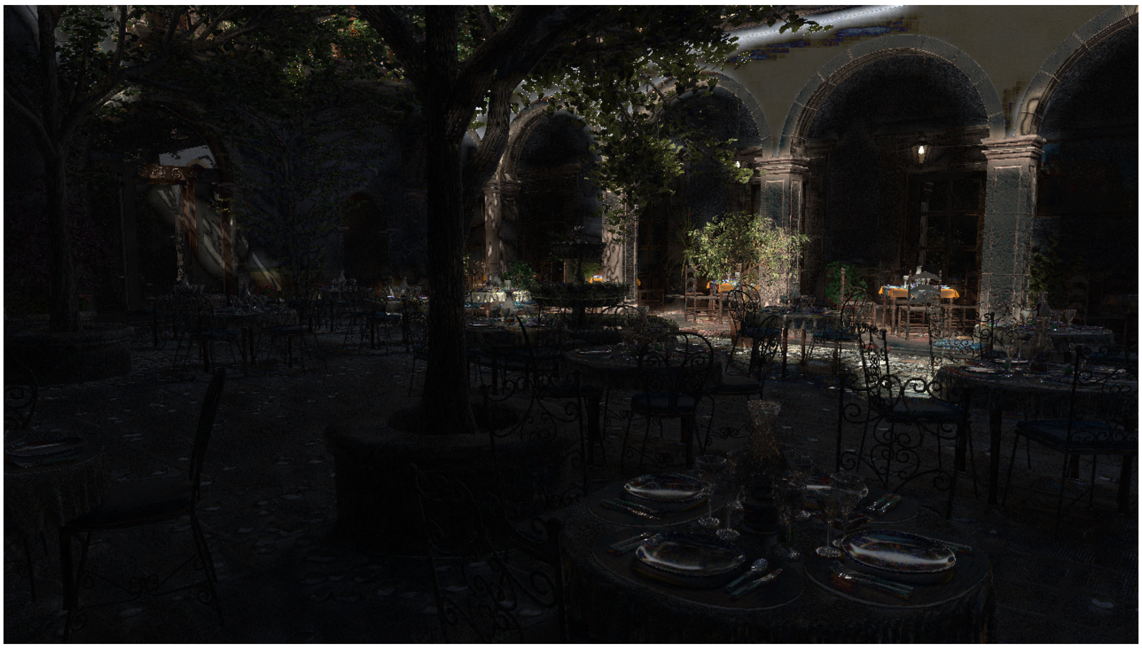


(c) *1spp PT + BMFR (24.25 dB)*

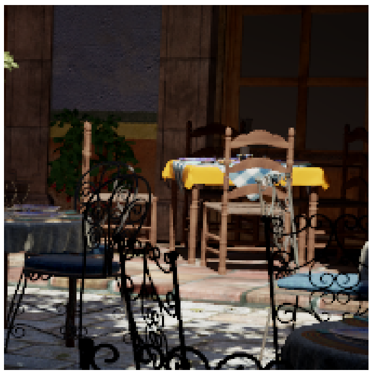
Figure 5.3. *Quality measurement of the methods in “San Miguel” (PSNR).*



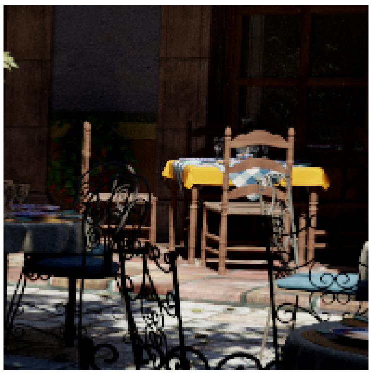
***(a)** L2 difference to reference*



***(b)** 1spp PT + BMFR difference to reference*

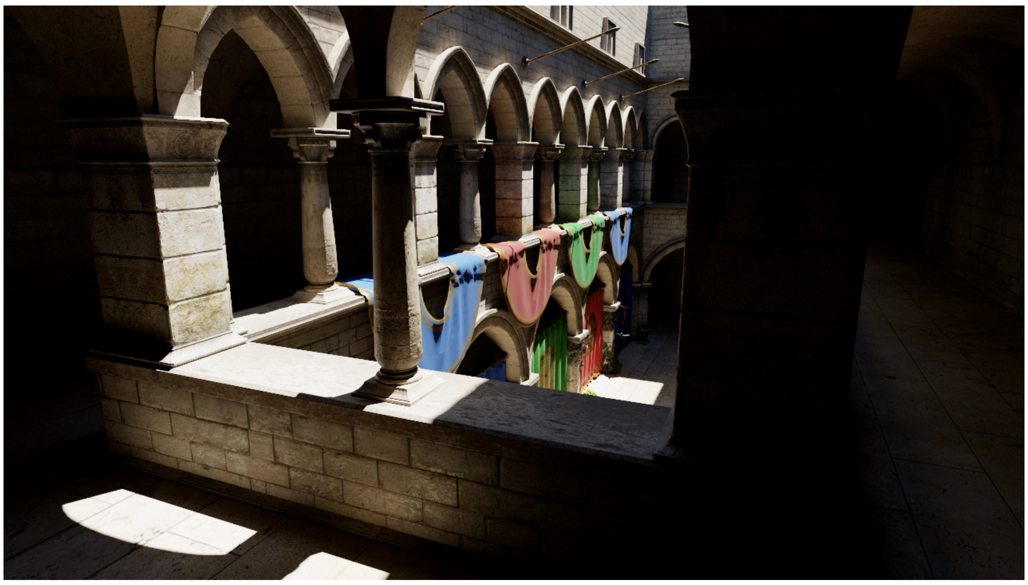


***(c)** L2 closeup*

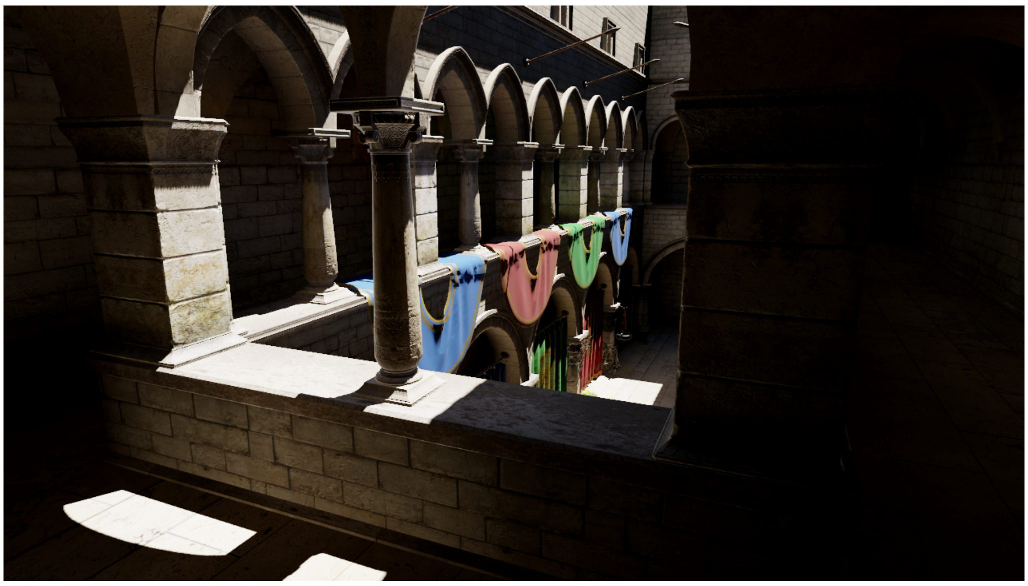


***(d)** Reference closeup*

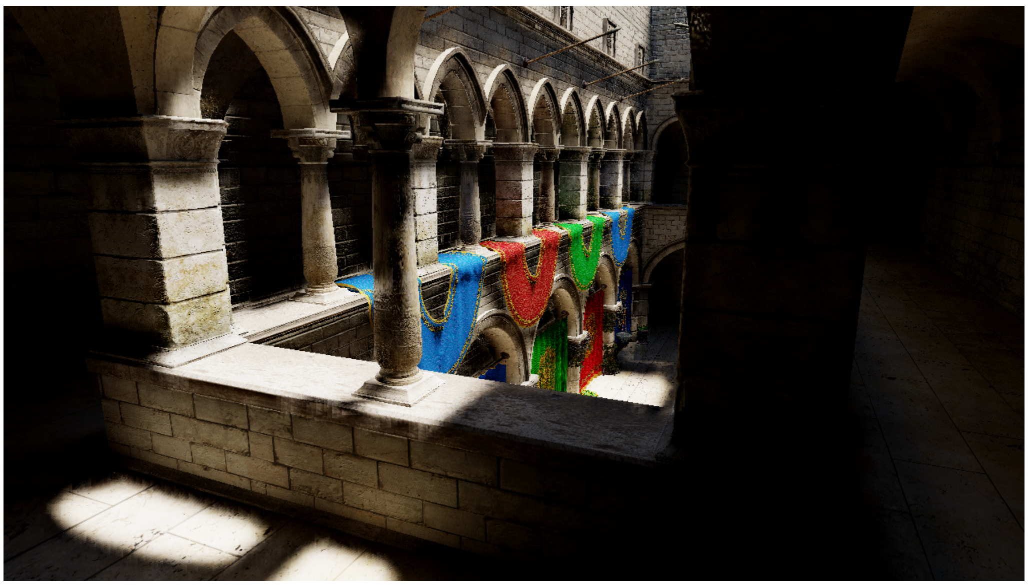
***Figure 5.4.** Quality comparison between the methods in “San Miguel”.*



(a) *4096spp path traced reference*

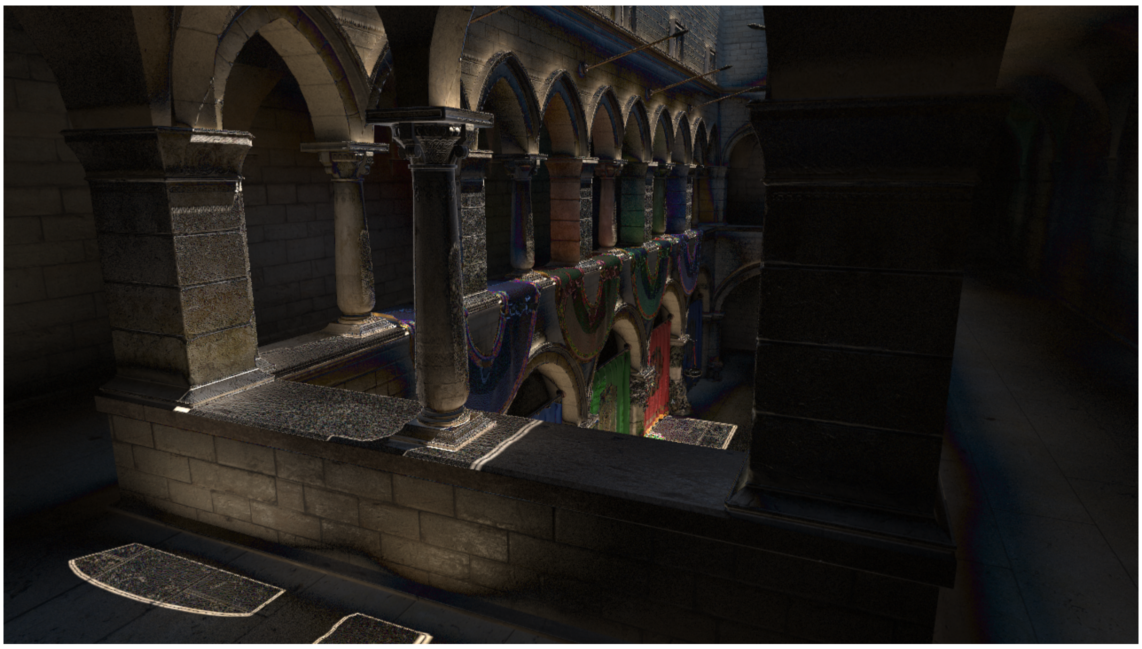


(b) *L2 spherical harmonics (25.02 dB)*

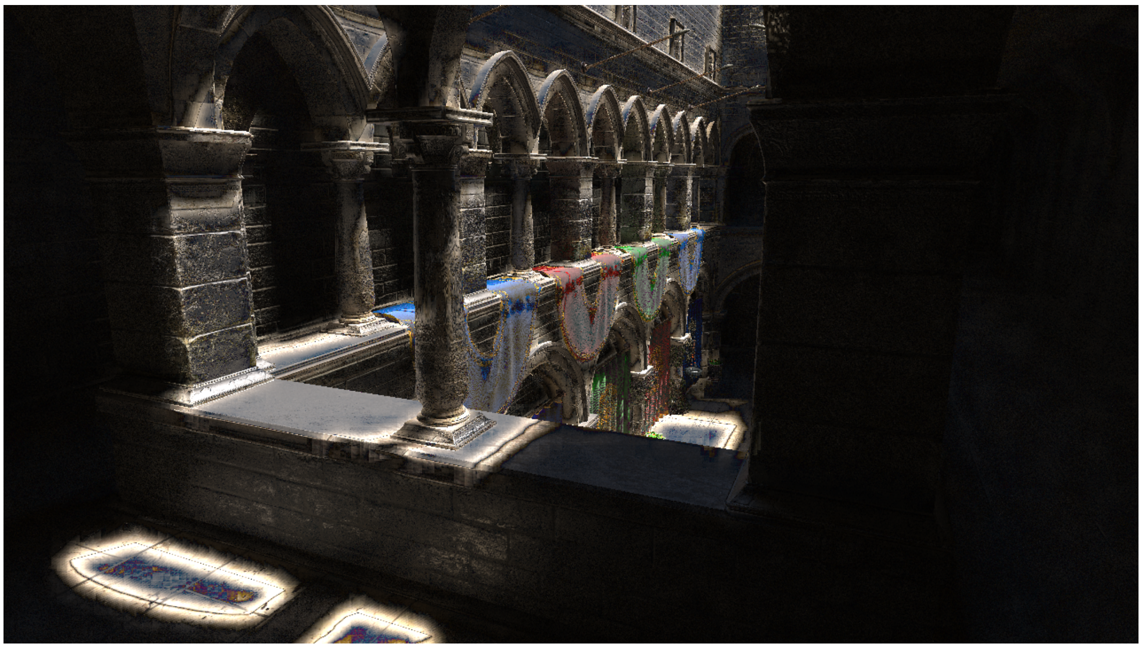


(c) *1spp PT + BMFR (18.99 dB)*

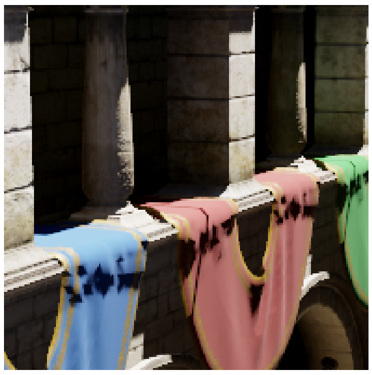
Figure 5.5. *Quality measurement of the methods in “Sponza” (PSNR).*



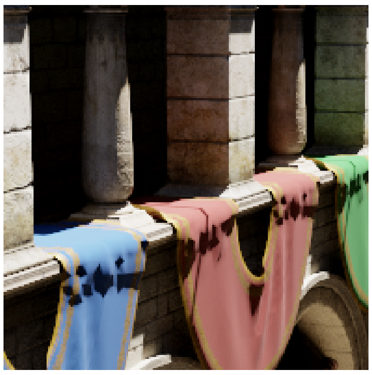
***(a)** L2 difference to reference*



***(b)** 1spp PT + BMFR difference to reference*



***(c)** L2 closeup*



***(d)** Reference closeup*

***Figure 5.6.** Quality comparison between the methods in "Sponza".*

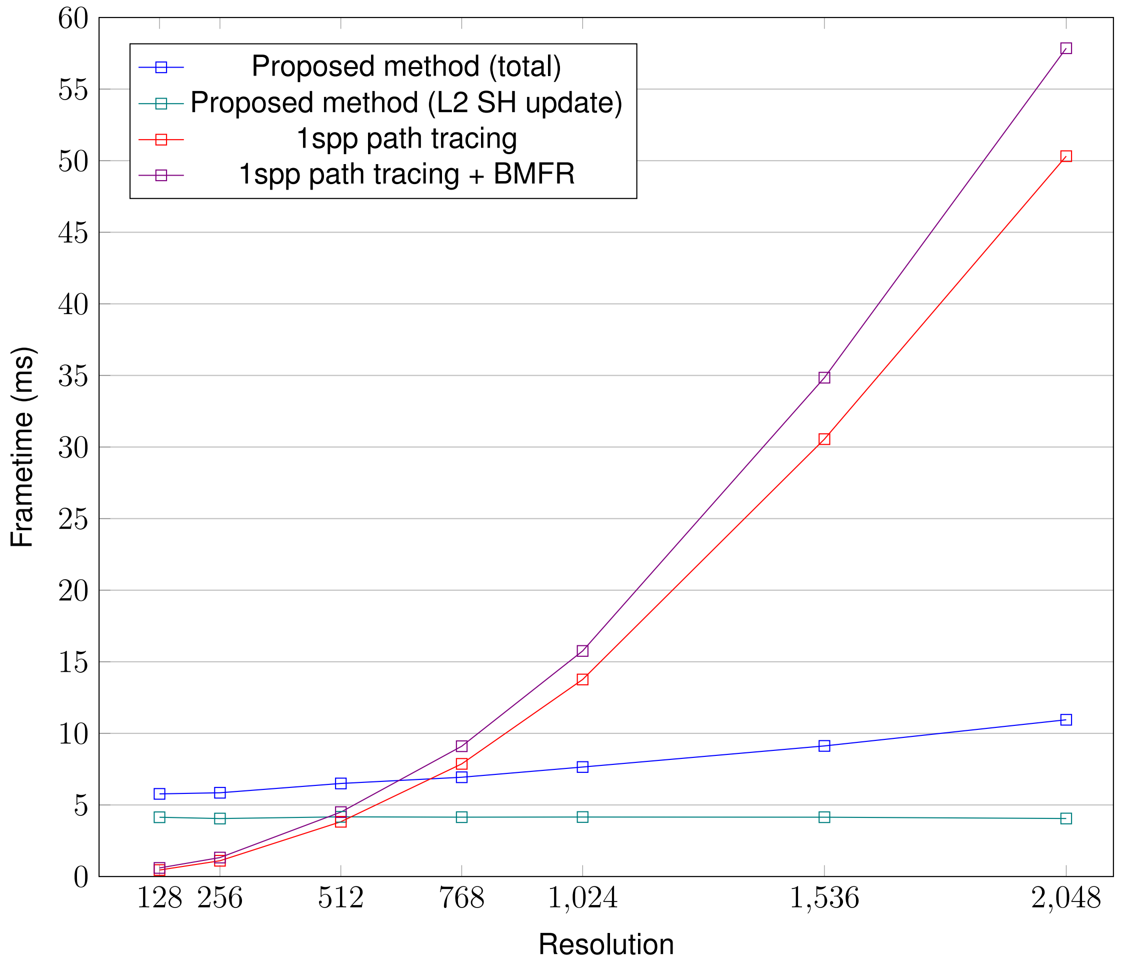


Figure 5.7. *A graph of the frametime as a function of resolution in the "Breakfast room" scene.*

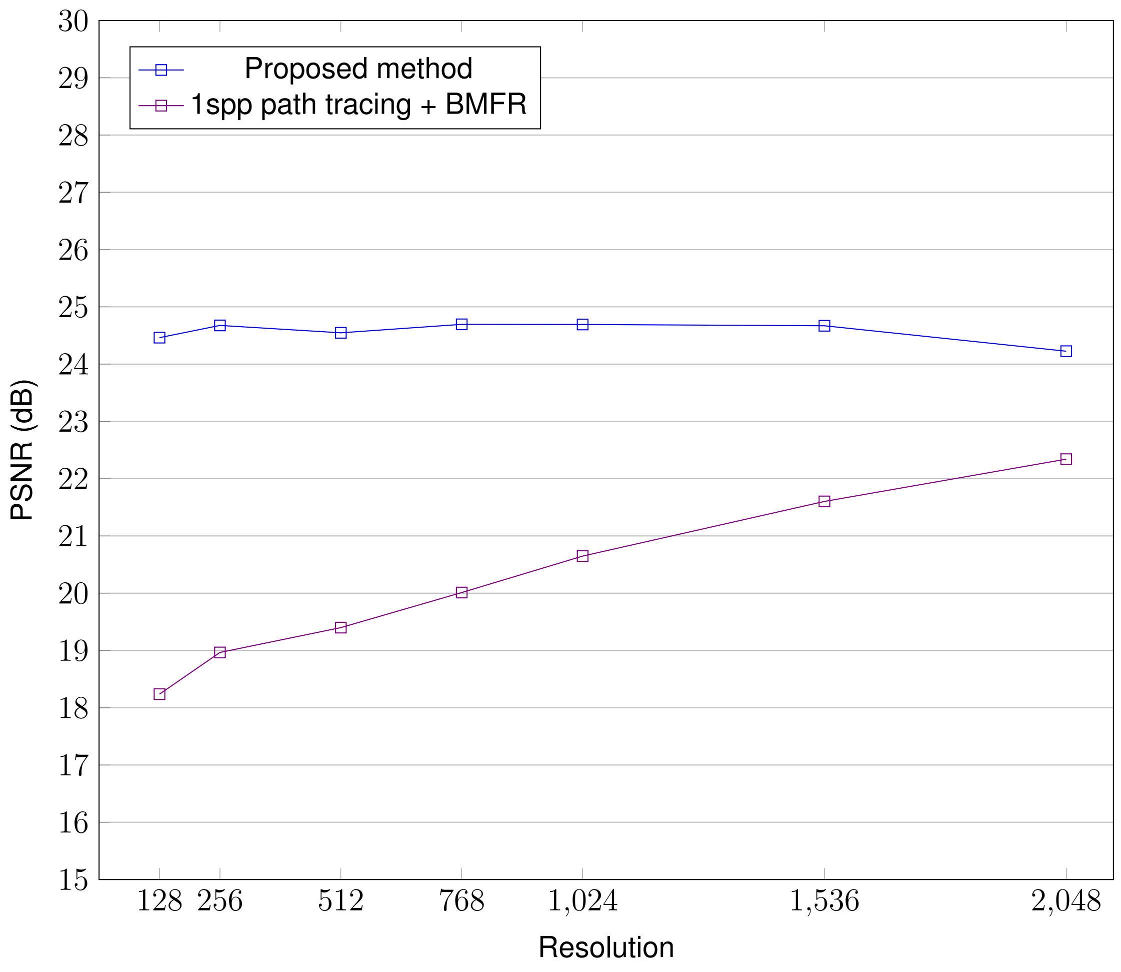


Figure 5.8. *A graph of the quality compared to reference as a function of resolution in the "Breakfast room" scene.*

5.1 Quality Characteristics

Compared to the quality of denoised path traced images, the proposed method fares better in every tested scene. One of the systematic issues with contemporary real-time denoisers such as BMFR is the reliance on simplistic albedo demodulation and re-modulation, which is fundamentally broken with the specular reflections of objects whose albedo does not affect the color of the reflection. In practice, this causes dielectric materials to appear overly saturated as their reflections become colored by their albedo color even when they should not be. This effect is apparent in the hanging cloths of the C image of Figure 5.5.

As seen in Figure 5.8, the quality of the proposed method is quite independent from resolution, whereas BMFR-denoised 1spp path tracing grows closer to reference with higher resolutions. This is expected; the constant tile size of BMFR means that more tiles are used in larger images and thus the ratio of tile size to resolution shrinks, which results in relatively less blurry images when the resolution is higher. In the proposed method, the number of pixels covered by each probe scales naturally with the resolution, so a similar major quality change does not occur.

5.1.1 Shadow Issues

The shadow quality is highly dependent on shadow mapping parameters. While PCSS is used to simulate soft shadows with moderate success, the difference images of Figure 5.2 make it quite clear that the shadow edge is one of the largest error sources. Other shadow mapping methods, such as moment shadow mapping [13], could provide higher-quality soft shadows with similar performance characteristics. PCF+PCSS are chosen for this implementation simply for ease of integration into a basic rasterization pipeline.

Compared to the path traced image denoised with BMFR, the proposed approach seems to have less error near the shadow edges. The shadowing occurs mainly on flat planar surfaces such as the walls, where the features within a block are highly uniform, preventing BMFR's regression component from expressing detail that could match such high-frequency changes in lighting, which results in overly blurred shadows. While the rasterized shadows do not match the reference perfectly, they at least do not appear any softer than they should.

5.1.2 Light Probe Issues

Looking at the difference images in Figures 5.2, 5.4 and 5.6, it can be seen that areas with high-frequency details from indirect light are usually a major error source. The left back corner of the room in Figure 5.1 and tiles just above the arches in Figure 5.5 are a couple of example cases demonstrating this. This is caused by the sparsity of the probes within the irradiance volume. This representation is mostly an undersampled version of the lighting, which is why such small details are easily lost. This could be remedied by adding more probes, but the cost of updating them would likely become prohibitive.

Another problem that is less visible is that the positioning of the probes affects the resulting lighting. This along with the undersampling means that spatial aliasing can be drastic; a probe placed in a small but dark area causes the influence of that area to be disproportionately large. This makes automatic placement of the irradiance volume difficult, since such situations must be avoided. The volumes are placed manually in the scene in the proposed method.

A related problem is light leaking, where the naive trilinear interpolation between probes can cause a bright probe to influence lighting through a wall. Methods have been derived to combat these issues, one is to store visibility information along with the probes and intelligently interpolate between them according to that information [34].

5.1.3 Spherical Harmonics Issues

While results for L2-L4 were measured in Tables 5.1, 5.2, 5.3, quality for orders beyond L2 was somewhat surprisingly barely different, sometimes even worse. L3 and L4 should technically be able to carry more high-frequency data and thus work better with specular reflections. One potential reason for this is the fitted specular lobe shape; shown in Figure 5.9, it can be seen that L3 and L4 feature more prominent lateral rings which are simply a instance of the ringing artefact typical of spherical harmonics. Because the performance and memory cost of L3 and L4 is significantly higher than L2, there seems to be no reason to use them over L2.

The result of fitting a spherical function into an SH representation often results in negative values where there should be none. This is evident in Figure 3.3. After filtering for irradiance, the result is good, but specular lobes will suffer from these negative values. If the lighting contribution from the SH would be negative, it is set to zero in the proposed method. The lack of directional detail also significantly reduces the attainable quality of sharper specular details.

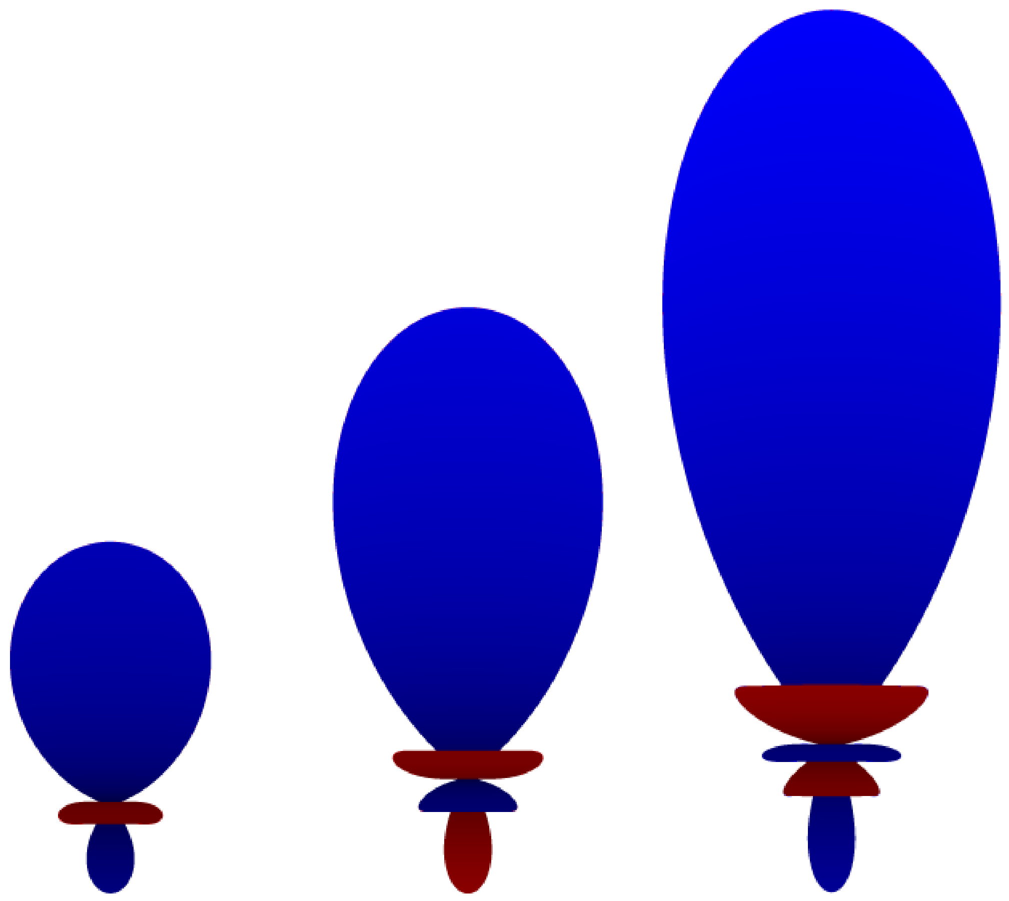

(a) L2 impulse *(b) L3 impulse* *(c) L4 impulse*

Figure 5.9. *Visualizations of L2-L4 SH lobes fitted to an impulse.*

5.2 Performance Characteristics

With the proposed method, the path tracing workload is entirely independent of the number of the viewports and the resolution of the viewport. Figure 5.7 shows that the time it takes to update the SH probes is constant as resolution changes, which is to be expected because the number of path traced samples only depends on the number of probes in the scene and how many samples per probe we wish to render.

In Tables 5.1, 5.2 and 5.3, it can be seen that the rasterization cost is unexpectedly high. This is primarily caused by the extremely heavy settings used in the test scenario: shadow filtering is done using 16 PCF + 16 PCSS samples and MSAA of 8 samples with full sample shading is used, meaning that every single pixel is evaluated 8 times in slightly different positions. Simply disabling sample shading massively improves performance, and using fewer samples for the shadows further drops the overhead. For example, in the “Breakfast room” scene, the rasterization step takes only about 0.3 ms on the same hardware after disabling MSAA and using bilinear interpolation for shadow maps.

In the proposed method, one L2 SH probe takes 56 bytes. A half-precision floating point coefficient for each SH band and color channel is stored resulting in 54 bytes, and this is then aligned to the next multiple of 4 bytes. An irradiance volume of 8x8x8 probes would then consume a mere 28672 bytes, making the bandwidth needed to transfer this lighting information across a network very low. This detail should aid when considering a scenario where the heavy path tracing workload of probe updates is computed by a server and the probes are then streamed to a low-power client responsible for rasterization and using the irradiance volume.

6 RELATED WORK

Because games have been relying on light probes for a long time, dynamically updating light probes are an easy-to-integrate method for providing real-time global illumination. Because of this, several rendering methods similar to what was proposed in this thesis have been published. The primary novelties of the proposed method are the usage of the spherical harmonics basis for real-time path traced probes, which allows very compact representation with rotational invariance and no high-frequency artefacts; and the usage of the light probes for specular lighting using a fitted zonal harmonics representation of GGX specular lobes. Both additions provide benefits for splitting the rendering work over a network: the compact representation reduces network bandwidth and the specular approximation reduces need for heavy computation on the client side.

Dynamic Diffuse Global Illumination (DDGI) is one method that updates light probes in real-time using ray tracing, achieving impressive performance. As the name implies, the probes are limited to diffuse illumination and a ray tracer is used to fill in glossy parts of the lighting as needed. The method uses an octahedral basis for its light probes with additional visibility information used to improve interpolation [34]. The octahedral representation does not suffer from ringing, unlike spherical harmonics. The DDGI paper proposes using 8x8 resolution for color data and 16x16 for visibility data per probe, with each pixel using a format that is 4 bytes in size. With these parameters, the probes can store more detailed information than L2 spherical harmonics probes. However, a single octahedral probe with those parameters would consume at least 1280 bytes of memory. This is almost 23 times more than the 56 bytes consumed by an L2 SH probe. DDGI has since been extended to support probe-based glossy reflection as well, although it does not take surface roughness into account [36].

Signed Distance Fields Dynamic Diffuse Global Illumination (SDFDDGI) uses a signed distance field approximation of the scene, which can be used to quickly trace rays even without ray tracing hardware. Like DDGI, this method uses an octagonal basis for the probes as opposed to the spherical harmonics used in this thesis. It has a fully automatic system for probe placement, which neither this thesis nor DDGI propose. Unlike in this thesis, the probes are not used for specular reflections in SDFDDGI, and the paper suggests using other methods such as screen-space reflections or ray tracing to cover that part. [37]

An SH-based approach for dynamic global illumination is presented in [38]. Probes are placed in the scene with an algorithm that ensures that each light receiver is visible in some probe. The method uses a real-time lightmap of direct illumination in order to update the probes. Since the probes are placed ahead of time, the contribution of the lightmap pixels to the probe can be precomputed. During runtime, the probes are then updated according to the state of the lightmap. The lighting from relevant probes is then interpolated to calculate lighting at the receiver. The interpolation takes advantage of precalculated visibility information to obtain high-quality lighting. Dynamic geometry is somewhat limited due to the amount of geometry-dependent precalculation needed and is as such not directly comparable to the fully dynamic method presented in this thesis. Compared to the method proposed in this thesis, it places more limitations on dynamic geometry and requires more precalculation, but seems to suffer from fewer artefacts and does not use ray tracing during runtime. Further, it does allow specular highlights from its probes unlike DDGI, but does this by approximating a directional light source from the probe and calculating the specular reflection from that.

7 CONCLUSION

This thesis proposed a new real-time rendering method using dynamically updated spherical harmonics light probes that can approximate both diffuse and specular indirect lighting. The probes are updated using path tracing and stored in an irradiance volume, which is a compact representation for light information.

This irradiance volume data is independent of camera position, which benefits the sharing of the data between multiple viewports and allows for outdated or less frequently updating lighting data to be used without much penalty. A simple yet novel method for approximating specular reflections from spherical harmonics probes was also presented as a part of this rendering method.

In all three measured test scenes, the proposed rendering method achieves greater quality at similar framerates than denoised single-sample-per-pixel path tracing, even when the capability for asynchronous SH updates is not utilized. Some systematic artefacts are visible; high-frequency details in indirect lighting such as short-distance light bounces are missing.

Unlike currently predominant methods of offloading rendering work from low-power devices to servers such as game-streaming services [5][4], the proposed method does not impose network latency on the entire image. Instead, all such latency can only affect the lighting of the scene; the camera and all objects in the scene can still move freely and without network latency.

Additionally, the bandwidth cost is easier to manage, as reduction of bandwidth can be achieved by less frequent lighting updates. This does not affect the user as drastically as reducing the framerate or quality of a video stream would, because all geometry still updates smoothly with no network latency. Instead, dynamic indirect lighting in the scene would become increasingly delayed.

A further advantage is that the proposed method can operate very well even when the connection is lost or there never was a connection in the first place: in such situations, the probes can simply use precalculated static values and the user only loses some visual flare, whereas with frame streaming such a situation would prevent the usage of the application entirely.

Integration of the proposed method into an engine built to support spherical harmonics probes should be simple. When there is a network connection to a server, the spherical harmonics data that would otherwise be precalculated and static can simply be replaced with data that is being streamed from the server in real-time.

A potential use-case for the proposed method would be gaming on a virtual reality headset similar to the Oculus Quest, where the headset itself contains the needed hardware to render less-demanding graphics[39]. In the pipeline of the proposed method, the headset would then be responsible for the light rasterization part the pipeline, and a connected server would then do the heavy computation of light probe data and stream it separately. This way, the latency issues of frame streaming would not hinder the virtual reality experience, which is known to be fairly latency-sensitive [40].

Generating high-quality real-time content for light-field displays is another area where the proposed method could be used. The heavy lighting computation would only have to be done once per frame, and only the relatively quick and tweakable rasterization step would need to be duplicated per viewport during the frame. Assuming a 1280x720 resolution per viewport, 32 viewports and a 4 ms light probe update, the rasterization step for each viewport must take less than 0.4 ms to reach 60 frames per second. With current hardware, this should be quite achievable in simpler scenes, at least without MSAA. Multiple GPUs could also be used to rasterize the different viewports.

The scenes used for the measurements are all constrained and quite compact scenes. For larger scenes, it would be necessary to place much larger irradiance volumes with more probes. Adaptively selecting which probes need updating and which should be transferred from the remote devices to the local ones should prove quite useful in keeping a stable level of performance. Simply picking probes by the view frustum would be an easy first step towards this. Using a lower-resolution volume for probes further away from the camera could be another effective optimization to the proposed method.

Adding support for a visibility term in the probes would likely significantly help with light leaking issues. Such approaches have been taken in related work. An additional coefficient could be added to the SH probes to encode depth. This could then operate as a visibility term, which would be used in probe interpolation in such a way that probes would not affect further than the depth encoded for each direction. Currently, irradiance volume placement is done manually. It would be interesting to do this automatically. The visibility term should make this viable, because light leaking is the primary consideration in probe placement.

To overcome the lack of short-distance high-frequency detail in lighting, the fragment shader in the rasterization could cast short rays to recover and re-add some of that information. Vulkan's ray query feature should be usable for this purpose.

In conclusion, the initial goal of the thesis was achieved, though with certain quality issues stemming from the naive probe interpolation that should be rectifiable in further research. The implementation of the method was much more involved than the thesis may lead on to believe, as an entirely new real-time path tracer was written from scratch during the thesis process, using experimental Vulkan extensions for hardware-accelerated ray tracing. This led to some unfortunate delay in the thesis schedule. The end result is satisfactory: the proposed method is fast enough to fulfill its intended role with acceptable quality.

REFERENCES

[1] Walter, B., Marschner, S. R., Li, H. and Torrance, K. E. Microfacet Models for Refraction through Rough Surfaces. *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France, 2007. ISBN: 9783905673524.

[2] NVIDIA Corporation. *NVIDIA RTX Ray Tracing*. URL: `https://developer.nvidia.com/rtx/raytracing` (visited on 10/18/2020).

[3] Varjo Technologies Oy. *Varjo VR-2*. URL: `https://varjo.com/products/vr-2/` (visited on 10/20/2020).

[4] Google LLC. *Stadia*. URL: `https://stadia.google.com/` (visited on 10/18/2020).

[5] Sony Interactive Entertainment LLC. *PlayStation Now*. URL: `https://www.playstation.com/en-us/ps-now/` (visited on 10/18/2020).

[6] Pharr, M., Jakob, W. and Humphreys, G. *Physically Based Rendering: From Theory to Implementation, Third Edition*. eng. Third Edition. Place of publication not identified, 2017. ISBN: 9780128006450.

[7] Lambert, J. H. *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. Klett, 1760.

[8] Friston, S., Ritschel, T. and Steed, A. Perceptual Rasterization for Head-Mounted Display Image Synthesis. *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: `10.1145/3306346.3323033`.

[9] Karis, B. Real Shading in Unreal Engine 4. 2013.

[10] Heitz, E., Dupuy, J., Hill, S. and Neubelt, D. Real-Time Polygonal-Light Shading with Linearly Transformed Cosines. *ACM Trans. Graph.* 35.4 (July 2016). ISSN: 0730-0301. DOI: `10.1145/2897824.2925895`.

[11] Möller, T. *Real-time rendering*. Fourth edition. ISBN: 0-429-22540-7.

[12] Fernando, R. Percentage-Closer Soft Shadows. *ACM SIGGRAPH 2005 Sketches*. SIGGRAPH '05. Los Angeles, California, 2005. ISBN: 9781450378277. DOI: `10.1145/1187112.1187153`.

[13] Peters, C., Munstermann, C., Wetzstein, N. and Klein, R. Beyond Hard Shadows: Moment Shadow Maps for Single Scattering, Soft Shadows and Translucent Occluders. *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '16. Redmond, Washington, 2016. ISBN: 9781450340434. DOI: `10.1145/2856400.2856402`.

[14] McGuire, M. and Mara, M. Efficient GPU Screen-Space Ray Tracing. *Journal of Computer Graphics Techniques (JCGT)* 3.4 (Dec. 2014). ISSN: 2331-7418. URL: `http://jcgt.org/published/0003/04/04/`.

[15] Crassin, C., Neyret, F., Sainz, M., Green, S. and Eisemann, E. Interactive Indirect Illumination Using Voxel Cone Tracing. eng. *Computer graphics forum* 30.7 (2011). ISSN: 0167-7055.

[16] Dachsbacher, C. and Stamminger, M. Reflective shadow maps. eng. I3D '05. 2005. ISBN: 1595930132.

[17] Advanced Micro Devices, Inc. *RDNA 2 Architecture*. URL: `https://www.amd.com/en/technologies/rdna-2` (visited on 10/18/2020).

[18] Kay, T. L. and Kajiya, J. T. Ray Tracing Complex Scenes. *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986). ISSN: 0097-8930. DOI: `10.1145/15886.15916`.

[19] Dietrich, A., Wald, I. and Slusallek, P. Large-scale CAD Model Visualization on a Scalable Shared-memory Architecture. 2005.

[20] Robert, C. P. *Monte Carlo statistical methods*. eng. 2nd ed. Springer texts in statistics. New York. ISBN: 0-387-21239-6.

[21] Mitchell, J., McTaggart, G. and Green, C. Shading in Valve's Source Engine. *ACM SIGGRAPH 2006 Courses*. SIGGRAPH '06. Boston, Massachusetts, 2006. ISBN: 1595933646. DOI: `10.1145/1185657.1185832`.

[22] Ramamoorthi, R. and Hanrahan, P. An efficient representation for irradiance environment maps. eng. SIGGRAPH '01. 2001. ISBN: 9781581133745.

[23] Sloan, P.-P. Stupid Spherical Harmonics (SH) Tricks. *Game Developers Conference* (Jan. 2008).

[24] Quilez, I. *SH - visualizer*. May 2014. URL: `https://www.shadertoy.com/view/lsfXWH`.

[25] Greger, G., Shirley, P., Hubbard, P. and Greenberg, D. The Irradiance Volume. *Computer Graphics and Applications, IEEE* 18 (1998). DOI: `10.1109/38.656788`.

[26] The Khronos Group Inc. *VK_KHR_ray_tracing_pipeline(3) Manual Page*. 2014-2020. URL: `https://www.khronos.org/registry/vulkan/specs/1.2-extensions/man/html/VK_KHR_ray_tracing_pipeline.html`.

[27] The Khronos Group Inc. *Vulkan® 1.2.156 - A Specification*. 2014-2020. URL: `https://www.khronos.org/registry/vulkan/specs/1.2/html/vkspec.html`.

[28] The Khronos Group Inc. *glTF Version 2.0 - Specification*. 2013-2017. URL: `https://github.com/KhronosGroup/glTF/blob/master/specification/2.0/README.md`.

[29] González, Á. Measurement of Areas on a Sphere Using Fibonacci and Latitude–Longitude Lattices. *Mathematical Geosciences* 42.1 (Nov. 2009). ISSN: 1874-8953. DOI: `10.1007/s11004-009-9257-x`.

[30] Chen, H. and Liu, X. Lighting and Material of Halo 3. *ACM SIGGRAPH 2008 Games*. SIGGRAPH '08. Los Angeles, California, 2008. ISBN: 9781450378499. DOI: `10.1145/1404435.1404437`.

[31] Reeves, W., Salesin, D. and Cook, R. Rendering antialiased shadows with depth maps. eng. SIGGRAPH '87. 1987. ISBN: 0897912276.

[32] McGuire, M. *Computer Graphics Archive*. July 2017. URL: https://casual-effects.com/data.

[33] Koskela, M., Immonen, K., Mäkitalo, M., Foi, A., Viitanen, T., Jääskeläinen, P., Kultala, H. and Takala, J. Blockwise Multi-Order Feature Regression for Real-Time Path-Tracing Reconstruction. eng. *ACM Transactions on Graphics (TOG)* 38.5 (2019). ISSN: 0730-0301.

[34] Majercik, Z., Guertin, J.-P., Nowrouzezahrai, D. and McGuire, M. Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields. *Journal of Computer Graphics Techniques (JCGT)* 8.2 (June 2019). ISSN: 2331-7418. URL: http://jcgt.org/published/0008/02/01/.

[35] Koskela, M. *BMFR Standalone implementation*. Apr. 2019. URL: https://github.com/maZZZu/bmfr.

[36] Majercik, Z., Marrs, A., Spjut, J. and McGuire, M. *Scaling Probe-Based Real-Time Dynamic Global Illumination for Production*. 2020. arXiv: 2009.10796 [cs.GR].

[37] Hu, J., Yip, M., Alonso, G. E., Gu, S., Tang, X. and Jin, X. Signed Distance Fields Dynamic Diffuse Global Illumination. eng. (2020).

[38] Silvennoinen, A. and Lehtinen, J. Real-Time Global Illumination by Precomputed Local Reconstruction from Sparse Radiance Probes. *ACM Trans. Graph.* 36.6 (Nov. 2017). ISSN: 0730-0301. DOI: 10.1145/3130800.3130852.

[39] Facebook Technologies LLC. *Oculus Quest*. URL: https://www.oculus.com/quest/ (visited on 10/18/2020).

[40] Abrash, M. *What VR could, should, and almost certainly will be within two years*. Steam Dev Days. 2014.

A FITTED ZONAL HARMONIC COEFFICIENTS FUNCTION FOR GGX SPECULAR LOBES

This GLSL function represents a function fitted to a set of numerically calculated ZH lobes for the specular contribution of GGX. Anisotropy is not modeled in any manner.

The L0 ZH is always equal to 1, so it is not included in the vector returned by get_ggx_specular_zh. Instead, it contains the L1, L2, L3 and L4 ZH coefficients in that order.

```
vec4 get_ggx_specular_zh(float roughness) {
    vec4 zh = vec4(0.27793123f, 0.59372022f, 0.2400839f, 0.000250700498);
    zh += vec4(0.905501229f, 10.57518269f, 21.6480923f, 5.53340572f) * cos(
        fma(vec4(roughness),
            vec4(2.49220829f, 3.49132073f, 3.92510137f, 3.98902127f),
            vec4(2.88755638f, 0.56672964f, 0.50116945f, 0.705097221f)));
    zh += vec4(1.98743320f, 9.52855312f, 19.90690569f, 3.23348085f) * cos(
        fma(vec4(roughness),
            vec4(1.79537159f, 3.58608449f, 4.01505002f, 4.63841986f),
            vec4(0.636261278f, 3.60689811f, 3.55551139f, 3.25144230f)));
    zh += roughness * fma(
        inversesqrt(
            vec4(0.329615862f, 0.29109984f, 0.25094573f, 0.211655471f) +
            roughness*roughness),
        vec4(1.54054310f, 4.35171889f, 7.58146856f, 9.84410536f),
        vec4(-4.73179141e-04f, -3.58678416f, -6.47567145f, -8.76804538f));
    return zh;
}
```