Global Illumination on a Mobile Phone: Scalable Real-time Global Illumination using Sparse Radiance Probes

Joseph Bennett

2019

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the requirements for the degree of
Master of Science in Computer Science.

Abstract

Real-time global illumination that scales from low to high-end hardware is important for interactive applications so they can reach wider audiences. To do this, the real-time lighting algorithm used needs to have varying performance characteristics.

Sparse Radiance Probes (SRP) is a recent real-time global illumination algorithm that runs in under $5\,\mathrm{ms}$ per frame on a high-end Nvidia Titan X GPU. Its low per-frame timings suggest it could scale to low-end devices, but no prior work provides complete implementation details and evaluates its performance across devices with varying performance characteristics to prove this. Therefore, this thesis aims to fill this gap and determine if SRP is scalable across low to high-end devices. SRP is implemented with adjustable scaling parameters, and its performance is compared across three test devices. A low-end iPhone 7, a mid-range AMD Radeon 560 graphics card, and a high-end AMD RX Vega 56 graphics card. The implementation in this thesis ran above $60\,\mathrm{FPS}$ for simple scenes on the iPhone 7, and with a reasonable reduction in quality, it ran just above $30\,\mathrm{FPS}$ on more complex scenes like Crytek Sponza. These results show that SRP can scale to low-end devices. While the implementation in this thesis runs in real time, there are implementation optimisations that would make SRP run even faster across all the test devices without reducing quality.

Acknowledgements

First, I would like to thank my supervisors Taehyun Rhee and Andrew Chalmers. The advice and guidance you have given me over the past year has been invaluable.

I also want to thank Thomas Roughton. For both the countless technical discussions we have had, and for the support you have given me as a friend.

Lastly, I want to thank my parents and two brothers for supporting and encouraging me over the years I have had at University.

Contents

Chapter 1

Introduction

Imagine you are working on a video game. Your desktop computer with a high-end graphics card simulates the world beautifully, the lighting feels realistic, and the game runs at a smooth 60 frames per second. You have a problem though: the game needs to run on more devices than just your computer. Unfortunately, the global illumination (GI) algorithm that provides the realistic lighting does not scale to the limited memory and processing power of mobile devices.

A scalable GI algorithm needs to run well on low-end devices while not being so limited that it cannot take advantage of the extra computation and storage available on high-end devices. Scalability is especially important as more and more devices with rendering capabilities are becoming available, from low-end mobile devices up to high-end desktop computers with the latest graphics cards. Applications for scalable real-time GI are not limited to video games: for example, architects may need to show a realistic visualisation of a design to a client on a mobile device while on-site, and yet still have a higher quality version available on a desktop computer in their office.

There are many real-time GI methods in production use, each offering different levels of performance-quality tradeoffs. Screen-space based methods [1, 2] perform well on low-end devices but their inherent lack of off-screen information means the result is equally inaccurate on high-end devices. Voxel cone tracing [3] produces high-quality results, but it is too expensive to run on current low-end devices [4]. Precomputed lightmaps [5] scale well across all hardware but force geometry and lighting to remain static at runtime.

Recently, Silvennoinen and Lehtinen introduced Sparse Radiance Probes (SRP) [6], a real-time GI method which supports dynamic lighting, cameras, and diffuse materials. It efficiently reconstructs indirect illumination at a dense set of receiver points distributed over scene surfaces by using a sparse set of radiance probes and precomputed transport coefficients. In computer graphics, a method is generally accepted as real-time if it runs in under $33.3\,\mathrm{ms}$; on a high-end Nvidia Titan X graphics card, their method runs in under $5\,\mathrm{ms}$ per frame. As SRP runs well under the real-time cut-off on this high-end graphics card, it suggests that low-end devices could also run SRP in real time. Additionally, the sparse nature of SRP means low-end devices with limited memory can still store SRP's precomputed transport coefficients, and the precomputed data can be packaged into reasonably small downloads over mobile networks. Silvennoinen and Lehtinen reported that SRP used less than $100\,\mathrm{MB}$ of memory for a scene used in an unannounced "triple-A" video game.

Although these metrics suggest SRP could scale to lower-end devices, there is no previous work that implements and evaluates SRP on low-end devices such as mobile phones. Additionally, the authors of SRP did not provide complete implementation details for precomputing the transport coefficients. The aim of thesis is therefore to determine if SRP is a scalable real-time GI method for various platforms, including an iPhone 7: that is, to determine if it runs in real time on low-end devices while also scaling up to provide higher quality results on high-end hardware. I implemented my own version of SRP with adjustable scaling parameters, and analysed its performance, quality, and scalability across low, mid, and high-end hardware.

The main contributions of this thesis can be summarised as:

- An implementation of SRP which runs across three devices with varying performance characteristics: a high-end AMD RX Vega 56 graphics card, a mid-range AMD Radeon 560 graphics card, and a low-end iPhone 7.
- A pipeline and complete implementation details for precomputing SRP's transport coefficients on the graphics processing unit (GPU).
- A performance and visual quality evaluation of my SRP implementation across low, mid, and high-end hardware to determine its scalability.

The thesis is structured as follows. Chapter 2 provides background on topics necessary to understand the chapters which follow. Chapter 3 describes the related work in real-time scalable GI algorithms. Chapter 4 covers the theory of the SRP;

it describes how SRP encodes radiance at a sparse set of probes while correcting for parallax error and visibility, how SRP reconstructs irradiance at receiver points using the probes and transport coefficients, and lastly how SRP compresses its transport coefficients using clustered principal component analysis (CPCA). Chapter 5 describes my implementation of SRP; it covers the steps of my implementation and how data flows between them, how radiance at the probes is efficiently projected into spherical harmonics at runtime using compute shaders, and how the pipeline for precomputing the transport coefficients on the GPU works. Chapter 6 provides the results of my performance and visual quality evaluation across low, mid, and high-end hardware to determine the scalability of SRP. Lastly, Chapter 7 outlines the conclusions of this thesis, discusses its limitations, and provides suggestions for future work.

Chapter 2

Background

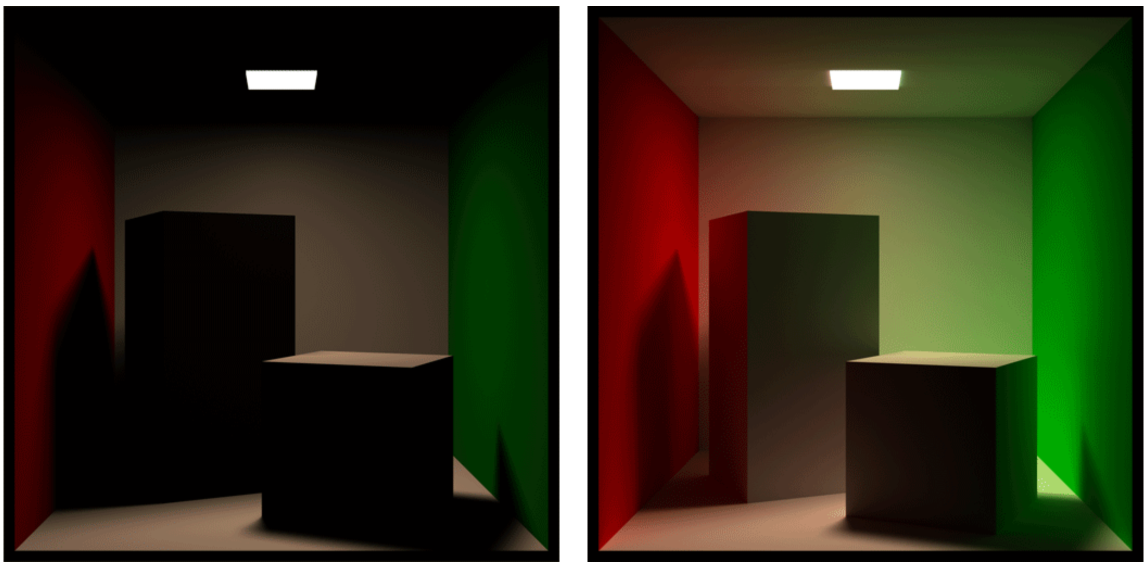


Figure 2.1: Direct vs Global Illumination. *"Cornell Box With and Without Radiosity Enabled"* by Paul Johnston is licensed under CC BY-SA 3.0 [7].

2.1 Global Illumination

Global illumination (GI), roughly speaking is the effect that other lit objects have on the illumination of a surface. A basic real-time rasteriser only considers local illumination. Local illumination assumes that the illumination of a point on a surface depends only on light sources in a scene; this means a rasteriser can compute illumination for each pixel independently of every other pixel. However, in the real world, the illumination of a point also depends how light interacts with other surfaces. This interdependency between surfaces makes global illumination expensive to simulate.

Shadows, reflection, and refraction are all examples of GI. These effects significantly increase the realism of a rendered images (Figure 2.1).

2.2 Physically Based Rendering Units

Physically based rendering (PBR) is a methodology that aims to simulate light as close as possible to how it behaves in the real world. One component of PBR is the use of physically based units from radiometry to measure light:

- *Radiant Flux*, Φ, is the flow of radiant energy over time. It is measured in watts (W).
- *Radiance* is the radiant flux per square metre per steradian.
- *Irradiance* is the density of radiant flux over an area - $\frac{d\Phi}{dA}$.

Radiance and irradiance are the units most commonly used in computer graphics. At a high level, radiance is a measure of an amount of light in a direction ω, and irradiance is the amount of light a point x receives from all directions.

2.3 The Rendering Equation

The aim of any global illumination method is to solve or approximate the rendering equation [8]. This equation describes the distribution of light, or more specifically, radiance in a scene:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n}) \mathrm{d}\omega_i \tag{2.1}$$

It states that the outgoing radiance from a point **x** in the direction ω_o is the sum of the radiance $\boldsymbol{x}$ emits in direction ω_o, plus the radiance $\boldsymbol{x}$ reflects in direction ω_o. The integral describes the amount of light $\boldsymbol{x}$ reflects. It gathers incoming radiance from all directions over the hemisphere Ω centred at the surface normal $\boldsymbol{n}$ and modulates it using the bi-directional reflectance distribution (BRDF) and Lambert's cosine law. The BRDF controls the amount of radiance from an incoming direction ω_i that $\boldsymbol{x}$ reflects in the outgoing direction ω_o.

The interdependence between surfaces can be seen through the recursion in this equation. The incoming radiance at a point $\boldsymbol{x}$ depends on the radiance it receives from all points it can see, and the incoming radiance at those points again depends on the incoming radiance from all the points they can see, ad infinitum.

2.4 Spherical Harmonics

Spherical harmonics are an infinite set of orthonormal basis functions defined over the unit sphere. Many computer graphics techniques use spherical basis functions to efficiently encode radiance or irradiance using a linear combination of coefficients and basis functions:

$$f(\omega) = \sum_{j=0}^{\infty} c_j B_j(\omega) \tag{2.2}$$

Each coefficient c_j conveys how similar a particular basis function B_j is to the original function $f(\omega)$. Finding these coefficients can be done by *projection*, and recreating the original function is called *reconstruction* (Equation 2.2). To project a function $f(\omega)$ onto an orthonormal basis[1], the inner product is taken with each basis function, yielding a coefficient c_j which represents how similar the basis function is to the original function:

$$c_j = \int f(\omega) B_j(\omega) d\omega \tag{2.3}$$

[1]While not applicable in this thesis, it is useful to know that non-orthonormal basis functions must take the extra step of multiplying by the inverse Gramm-matrix here. See Iwanaki and Sloan for more details [9].

Limiting the number of basis functions to n gives a band-limited approximation of the original function:

$$f(\omega) \approx \sum_{j=0}^{n} c_j B_j(\omega) \tag{2.4}$$

This approximation is useful for real-time global illumination methods as it provides a way to efficiently store and evaluate radiance and irradiance distributions. In fact, spherical harmonics can represent Lambertian diffuse irradiance using nine coefficients with only 1% error [10]. However, a band-limited approximation only captures low-frequency information. Spherical harmonics represent low-frequency diffuse irradiance well, but information is lost when encoding radiance or irradiance with higher-frequency information like visibility. Unfortunately, low-frequency representations are a necessary trade-off for real-time methods, as alternatives generally require expensive sampling of the rendering equation.

In computer graphics, we only use the real part of spherical harmonics. These are defined as follows:

$$Y_l^m(\theta,\phi) = \begin{cases} \sqrt{2}K_l^m \cos(m\phi)P_l^m(\cos\theta) & m > 0 \\ \sqrt{2}K_l^m \sin(-m\phi)P_l^{-m}(\cos\theta) & m < 0 \\ K_l^0 P_l^0(\cos\theta) & m = 0 \end{cases} \tag{2.5}$$

where P is the associated Legendre polynomial function, and K is a normalisation factor. *Spherical Harmonic Lighting: The Gritty Details* [11] includes definitions of the Legendre polynomials and the normalisation factor.

Spherical harmonic functions are parameterised by l and m. These parameters group the functions into *orders*, where l is the *order* of a function and m is the index of a function within that order. Orders can also be referred to as *bands*. While orders are indexed from zero, bands start at one (i.e. order-0 is the first band, order-1 is the second etc.). For given order l, the index m lies within the range $[-l, l]$ and it follows that each order has $2l + 1$ basis functions. As an example, an order-2 spherical harmonic (3rd row of Figure 2.2) has nine functions in total: the five functions from its band and the four from the bands before it.

The total number of functions for an order (including all lower bands) is $(l+1)^2$. For cleaner notation in this thesis, I refer to the direction using a single vector ω instead of a spherical coordinates (θ, ϕ), and I index each basis function with a single index j instead of l and m (i.e. $Y_j(\omega)$ instead of $Y_l^m(\theta, \phi)$).

Spherical harmonics are not the only spherical basis functions in use in rendering; non-linear wavelets [12], spherical radial basis functions (SRBFs) [13], and Ambient Dice [9] are some examples.

To calculate each coefficient c_j, I solve Equation 2.3 using Monte Carlo integration within a compute shader. The next section introduces Monte Carlo integration, followed by a section on compute shaders.

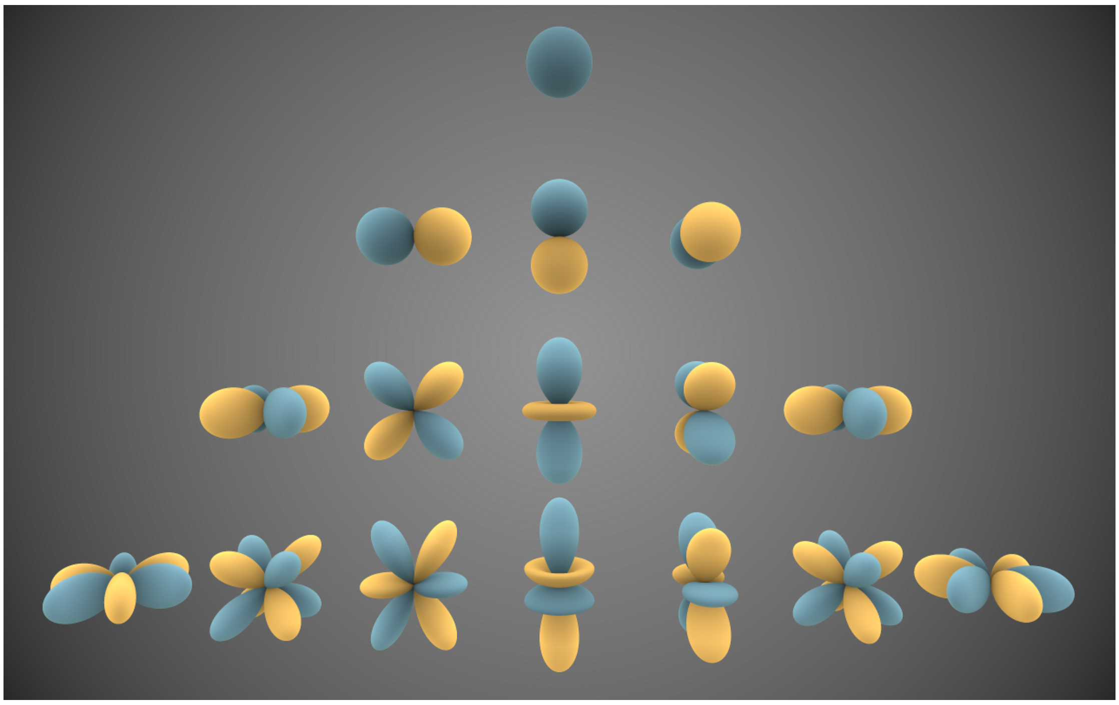


Figure 2.2: Visualisation of the first four spherical harmonic orders. Order-0 spherical harmonics are shown in the first row, order-1 in the second row and so forth. The higher the order, the higher the frequency of the functions. Blue shows where the function is positive, and yellow where it is negative. The above image *"Spherical Harmonics"* by Iñigo Quilez is licensed under CC BY-SA 3.0 [14].

2.5 Monte Carlo Integration

Monte Carlo integration is a numerical integration technique which uses random samples to estimate the value of an integral. It is especially useful for solving complex multi-dimensional integrals like the rendering equation because its convergence rate does not depend on the dimension of the integrand.

The basic Monte Carlo integration algorithm is simple: it takes N samples of a function and then divides the sum by the total number of sample N. For this to work, the samples must be uniformly distributed over the integral's domain. Given enough samples, the algorithm will converge to the true value of the integral.

For more information on Monte Carlo integration, Physically Based Rendering (PBRT) [15] provides a good introduction to the subject and why it works in the context of computer graphics.

2.6 Compute Shaders

Compute shaders are general purpose shaders which allow non-graphics specific functions to run on a graphics processing unit (GPU); they are a generalisation of vertex and fragment shaders. Each shader function is run in parallel over different data; in vertex and fragment shaders, this data is usually a single vertex or a pixel, but for a compute shader it can be anything.

Different shading languages use different terminology to describe how compute shaders operate. This thesis uses the terminology from the Metal Shading Language [16]. To transform data using a compute shader, it must be broken into chunks. Metal calls these chunks *threadgroups*. Within a threadgroup are individual *threads* which each run a *kernel* function. The threads within a threadgroup can run in parallel and can access shared threadgroup memory.

2.7 Principal Component Analysis and Singular Value Decomposition

Principal Component Analysis (PCA) is a technique used to find the axes in a data set with the most variation. These axes are called *principal components*.

Singular Value Decomposition (SVD) is one way to perform PCA. SVD decomposes matrices into the form:

$$M = U\Sigma V^T \tag{2.6}$$

where U is a $m \times m$ matrix, Σ is a $m \times n$ rectangular diagonal matrix, and V is an $n \times n$ matrix. SRP uses the SVD to implement Clustered Principal Component Analysis (CPCA) [17] for compressing its precomputed transport coefficients.

Chapter 3

Related Work

There is a great deal of research available on real-time global illumination. In this section, I cover the related work most relevant to scalable real-time global illumination. For more detailed background, Ritchell et al. provide an in-depth coverage of the state-of-the-art in interactive global illumination [18].

Monte Carlo path tracing [8] sets the bar in terms of the quality achievable by rendering algorithms. It is the industry standard for rendering VFX, animated films, commercials, and more [19]. Weta Digital's Manuka [20], Pixar's RenderMan [21], and Disney's Hyperion [22] are renderers from the top VFX and animation studios which are all based on path-tracing. However, path-tracing, and other ray tracing based methods [23, 24] are typically very expensive making them unsuitable for real-time applications.

While ray tracing can be expensive, recent advances in both GPU hardware and algorithms are making real-time ray tracing feasible [25]. For example, Weta Digital now uses real-time ray tracing as a pre-visualisation tool [26], and some video games such as Battlefield 5 use real-time ray tracing to render reflections and other global illumination effects [27].

The current problem with these advances is that they are only performant on recent high-end graphics cards specifically designed to improve the performance of ray tracing. Low-end devices like mobile phones present a more challenging environment for ray tracing: their small form factor means there is little room for cooling, and energy usage must be kept to a minimum to preserve battery life.

One way mobile GPUs reduce energy usage is through dynamic voltage and frequency scaling (DVFS). DVFS adjusts the voltage supplied to the GPU based on previous GPU usage. This typically saves energy at the cost of minor decreases in performance but unfortunately does not work well with global illumination algorithms [28]. There is research looking to improve this; Lee et al. proposed a novel GPU architecture in 2013 to improve ray tracing performance on mobile devices [29]. However, at publication, their work was in the simulation stage and they had not developed any hardware. Real-time ray tracing may be a viable solution in the future for scalable real-time global illumination, but hardware acceleration is not yet available on a wide enough range of devices.

One way to overcome the cost of ray tracing at runtime is *Lightmapping* [5]. Lightmapping stores precomputed illumination samples in a two-dimensional texture called a lightmap. In a manner similar to texture mapping, UV coordinates are assigned to surfaces in a scene which map to locations in a lightmap. Path tracing is commonly used to precompute the illumination samples. For diffuse Lambertian surfaces, a lightmap stores precomputed irradiance samples. At runtime, a forward or deferred pass queries the lightmap at the UV coordinates for the current surface the pass is shading. For normal-mapped or rough specular surfaces, it is infeasible to store samples for every view direction. Instead, a lightmap can store radiance samples encoded using a spherical basis such as spherical harmonics [30, 31].

Lightmapping has been proven to be scalable by many video games; for example, the puzzle game The Witness uses lightmapping and is available on mobile devices, consoles, and desktop computers [32, 33]. The primary limitation of lightmapping is that geometry and lighting must remain static at runtime. This is not an issue for applications like The Witness where there are no dynamic lights or geometry, but other techniques must be considered for applications for which these limitations are too restrictive.

Screen space global illumination techniques use the final rendered image and other auxiliary textures from the rasterisation process like the depth buffer to approximate global illumination effects. Examples include *Screen Space Ambient Occlusion (SSAO)* [1], which approximates the self-shadowing around edges of geometry; and *Screen Space Directional Occlusion (SSDO)* [2] which extends SSAO by adding directional shadows and one diffuse indirect bounce of light. Screen space techniques are highly performant, support both dynamic lighting and geometry, and need no precomputation. However, they lack accuracy as indirect light from offscreen sources cannot contribute.

Many Light Methods use many virtual point lights (VPLs) to approximate global illumination. *Instant Radiosity*[34] first introduced this idea. Paths of light are traced through a scene, and a VPL is placed at the intersections the paths make with geometry. Many light methods are biased (meaning their result is not exactly correct), but they produce images without noise and are scalable as their performance/quality can be tuned from plausible renders in real time to high-quality offline renders; *Lightcuts* [35] made many light methods scalable so they could produce high-quality offline renders, and *Bidirectional Lightcuts* [36] helped reduce the inherent bias. However, these extensions mainly improve offline applications.

Imperfect shadow maps by Ritchell et al. [37] is a many-light method which interactively renders GI with dynamic lighting and geometry. They later improved on their original imperfect shadow map implementation by adding support for more complex scenes [38]. While close, imperfect shadow maps does not run above 30 FPS on the dedicated graphics cards it was tested on, indicating that it may not be suitable for low-end devices. For more information on many light methods, Dachsbacher et al. provide a in-depth survey [39].

Voxel Cone Tracing (VCT) by Crassin [3] is a real-time global illumination technique which supports indirect diffuse and specular lighting, as well as dynamic lighting and geometry. At a high-level, VCT is similar to path tracing but with two approximations to improve performance. First, it approximates geometry by voxelising the scene, and second, it traces cones through the voxelised scene instead of firing many individual rays. Cascaded voxel cone tracing [40] offers improved performance by using dynamic resolution voxel grids. Similar to mip-maps, they use a high resolution grid near the camera, which progressively decreases in resolution as the distance from the camera increases.

VCT runs in real time on mainstream graphics cards but it is still too expensive for lower-end devices. Nvidia reported their implementation to run in 7.4 ms on the GTX770, and 28.1 ms on a then-mid-range GTX650 at resolution 1920x1080 and ″medium″ quality. Both of these graphics cards are considerably more performant than modern mobile graphics hardware. Wahleén′s master′s thesis found that VCT was too expensive to run in real time on a Samsung S7 Edge mobile phone [4], and Gaijin Entertainment also stated at GDC 2019 that voxel cone tracing was too expensive for their purposes on an Xbox One [41]; however, their GI budget is only a few milliseconds due to other GPU computation costs needed for their game. Instead of using VCT, they use their own method which also makes use of a voxelised scene. It runs in $0.7\,\mathrm{ms}$ for single bounce, and $1.6\,\mathrm{ms}$ for n-bounces on an Xbox One.

McGuire et al. introduce a new data structure and an algorithm called *light field probes* [42]. It supports real-time global illumination for static scenes with both static and dynamic objects.

Enlighten is a commercialised and proprietary solution for rendering global-illumination in real time [43, 44]. It runs on low-end mobile devices to high-end desktop GPUs. Unfortunately, there is limited information available on the details of their solution; however, at a high level, it uses lightmapping and light probes for diffuse indirect lighting. Their solution aims to minimise GPU time so that the GPU is free to perform other tasks, and as such they update their probes asynchronously on the central processing unit (CPU).

Precomputed radiance transfer is a method introduced by Sloan et al. for rendering indirect environment lighting in real time [45]. PRT splits the rendering equation into two parts: a lighting function and a transfer function. The lighting function describes incoming radiance, and the transfer function describes how that incoming radiance interacts with surfaces in a scene. This separation means PRT can reduce computation considerably by precomputing the transfer function. While this means geometry must remain static, it reduces runtime computation enough to make dynamic real-time indirect lighting possible. In addition to splitting the rendering equation, precomputed radiance transfer methods also band-limit the indirect environment lighting by projecting it into a spherical basis. Sloan et al. use spherical harmonics [46], but other bases like non-linear wavelets have also been used [12].

Early PRT methods were limited to environment lighting. Hasan et al. extend PRT adding support for local lights (e.g. point and spot lights) [47]. It is a direct-to-indirect method which computes indirect illumination by gathering from a large set of radiance/irradiance samples distributed in a scene called radiance probes.

Sparse Radiance Probes (SRP) is a real-time global illumination method based on PRT and other direct-to-indirect methods which supports dynamic lighting, cameras, and diffuse materials [6]. Additionally, it can support glossy materials for the final bounce towards the camera. Silvennoinen and Lehtinen, the authors of SRP, reported it ran in $2.5\,\mathrm{ms}$ on an a Nvidia Titan X for the test scene Sponza, and $4.87\,\mathrm{ms}$ for Brutalist Hall, a novel scene from a ”triple-A” video game. While these scenes were rendered on a high-end graphics card, the low render times suggest it could also scale to low-end devices. The next chapter of this thesis describes the theory of SRP in detail, followed by a chapter describing my SRP implementation.

Chapter 4

Sparse Radiance Probes: Theory

The Sparse Radiance Probes (SRP) technique aims to compute indirect illumination at a dense set of points called *receivers*. These receivers can be any set of points distributed over scene surfaces or within space. Computing the indirect illumination (or, more specifically, the outgoing indirect radiance) requires densely sampling the rendering equation [8] over directions ω_o for all receivers $\boldsymbol{x}$:

$$L_o(\mathbf{x}, \omega_o) = \int_\Omega f(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n}) \mathrm{d}\omega_i \tag{4.1}$$

Unfortunately, it is too expensive for current hardware to do this in real time. SRP first simplifies the problem by assuming all surfaces use the view-independent diffuse Lambertian BRDF[1]:

$$L_o(\mathbf{x}) = \frac{\rho}{\pi} \int_\Omega L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n}) \mathrm{d}\omega_i \tag{4.2}$$

Removing a dimension from the problem helps make it more tractable, but the incoming radiance at each receiver $L_i(\boldsymbol{x}, \omega_i)$ remains too expensive to sample in real time. Instead, SRP approximates $L_i(\boldsymbol{x}, \omega_i)$ by interpolating between radiance observed at a much sparser set of points called *radiance probes*. Section 4.1 covers how SRP performs this interpolation while accounting for parallax error and visibility.

[1]As an extension, SRP also approximates glossy view-dependent BRDFs, limited to the final bounce towards the camera. This approximation technique is not implemented in this thesis.

The interpolation also helps, but SRP still needs to compute the incoming radiance at each probe. On top of this, it must do this every frame to support dynamic lighting in real time. The problem is that even with only a few probes, it is still too costly to compute incoming radiance at each probe by sampling the rendering equation. To overcome this, SRP band-limits each probe's radiance by encoding it in a spherical basis (section 4.2). This encoding means SRP can efficiently store and evaluate radiance in real time using a small number of radiance coefficients. Dynamic lighting is possible as we can efficiently generate these coefficients in real time using a compute shader (section 5.2). The trade-off of this representation is that the encoding only captures low-frequency information, which can reduce the accuracy of indirect shadows and cause colour bleeding. Silvennoinen and Lehtinen [6], the authors of SRP, use spherical harmonics as the set of spherical basis functions.

While the interpolation and band-limited probes significantly reduce computation, performing the interpolation at every receiver in real time is still too expensive. The interpolation provides an approximation for the incoming radiance in a direction at a receiver, but to calculate outgoing radiance we must integrate the incoming radiance over a hemisphere at the receiver; since it requires densely sampling an integral, this operation is too expensive to perform in real time. To overcome this, SRP factors the radiance coefficients out of this integral and precomputes the resulting integral (section 4.3). This precomputation yields a set of transport coefficients for each receiver.

For simple scenes, a weighted sum of the radiance coefficients and transport coefficients for a receiver is enough to compute diffuse indirect irradiance in real time. However, this is not possible for more complex scenes, as storing the transport coefficients for every receiver can easily consume gigabytes of memory. The last component of SRP is compression. Section 4.5 covers how SRP uses clustered principal component analysis (CPCA) to compress its transport coefficients.

4.1 Sparse Interpolation

To avoid densely sampling the rendering equation (Equation 2.1) at every receiver, SRP builds upon the work of sparse interpolation techniques [48, 49]. Sparse interpolation techniques use a more accurate but more expensive method to sample radiance at a sparse set of points and then reconstruct radiance at a denser set of points by interpolating between those sparse samples. SRP calls the sparse sample points *radiance probes*, and the dense sample points *receivers*. The receivers interpolate between nearby probes to approximate their incoming radiance (Figure 4.1).

Ward et al. [48] introduced this idea with the following interpolator:

$$L(\boldsymbol{x}, \omega) \approx \frac{\sum_i^n w_i(\boldsymbol{x}) L(\boldsymbol{p}_i, \omega)}{\sum_k^n w_k(\boldsymbol{x})} \tag{4.3}$$

where $\boldsymbol{x}$ is a receiver, n is the probe count, $L(\boldsymbol{p_i}, \omega)$ is the incoming radiance for a probe $\boldsymbol{p_i}$ in direction ω, and $w_i(\boldsymbol{x})$ is a weight controlling how much of probe i's radiance contributes to the receiver's radiance.

SRP uses a weighting function based on the distance between $\boldsymbol{p}_i$ and the receiver $\boldsymbol{x}$ from Lehtinen et al. [50]:

$$\begin{cases} w(t) = 2t^3 - 3t^2 + 1 & 0 \geq t \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

where t is $\|\boldsymbol{x} - \boldsymbol{p_i}\|/r$ and r is a cut-off radius that defines the maximum distance at which a probe contributes to a receiver.

There are two main issues with the interpolator in Equation 4.3: first, it suffers from parallax error, and second, it does not account for visibility. The remainder of this section describes how SRP addresses these two issues and then defines SRP's interpolator.

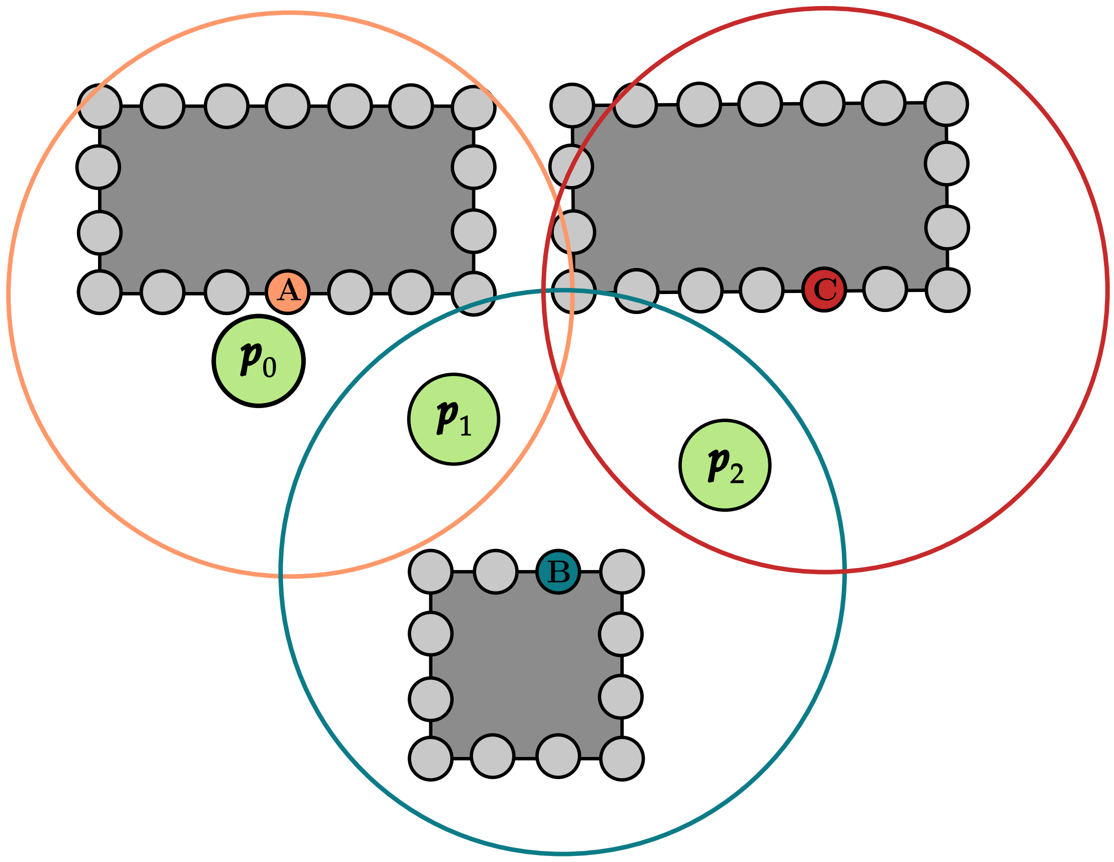
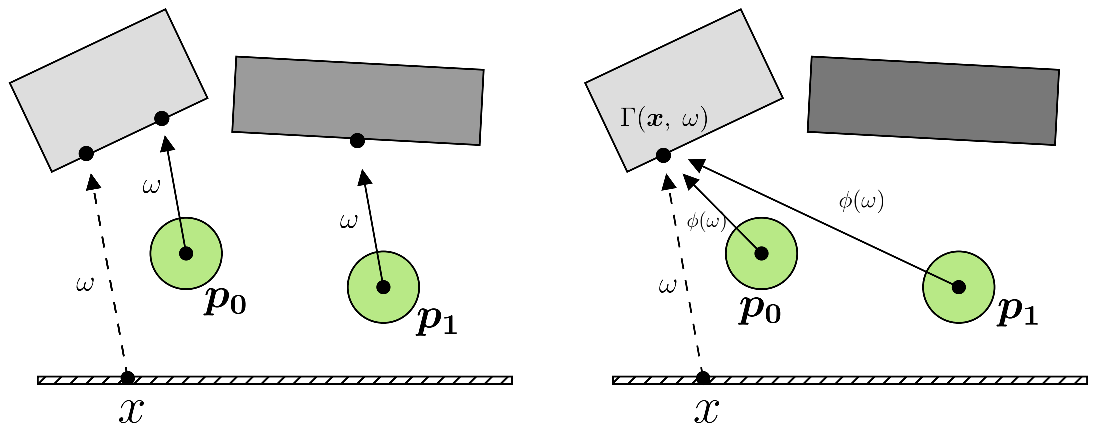


Figure 4.1: A top-down view of receivers and probes in a scene. The dark grey boxes represent geometry, the light grey circles distributed over the geometry represent the dense set of receiver points, and the larger green circles represent the sparse set of radiance probes. The coloured circles around the highlighted receivers represents their cut-off radius (i.e. the maximum distance at which a probe can contribute to a receiver). The receivers interpolate their radiance from nearby probes; for example, receiver A interpolates its radiance from probes $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$. Probe $\boldsymbol{p_1}$ contributes more to A than probe $\boldsymbol{p_0}$ as it is closer; receiver B interpolates from probes $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$; and receiver C interpolates from probe $\boldsymbol{p_2}$.

4.1.1 Parallax Error Correction

Parallax error occurs in Ward's interpolater (Equation 4.3) when the incoming radiance observed by a probe in a direction ω comes from a different location than the radiance observed by a receiver in that same direction ω. The only scenarios this does not occur in are when either a probe is in the same location as a receiver or the source of the radiance is at infinity (e.g. an environment map). Parallax error is an issue when the radiance observed differs between the probe and receiver (Figure 4.2a); it is less of an issue with low-frequency diffuse lighting but becomes more noticeable with higher frequency changes such as harsh shadows.



(a) Parallax error. Sampling in the same direction ω at probes $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ is incorrect due to parallax error. Probe $\boldsymbol{p}_1$ incorrectly samples the darker wall.

(b) Parallax error correction. Sampling in the corrected direction $\phi(\omega)$ means the probes correctly sample from the same surface point $\Gamma(\boldsymbol{x}, \omega)$ that $\boldsymbol{x}$ sees in direction ω.

Figure 4.2: Example of parallax error in (a), and its correction in (b).

To account for parallax error, we sample in a corrected direction $\phi(\omega)$, instead of the same sample direction ω used at a probe (Figure 4.2b). The corrected direction points from the probe towards the same point in the scene the receiver sees in the direction ω.

We calculate the corrected direction using the following equation:

$$\phi(\omega) = \phi(\boldsymbol{x}, \boldsymbol{p}_i, \omega) = \Gamma(\boldsymbol{x}, \omega) - \boldsymbol{p}_i \tag{4.5}$$

where $\Gamma(\boldsymbol{x}, \omega)$ is the ray-cast operator that returns the intersection point of a ray fired from $\boldsymbol{x}$ in direction ω.

4.1.2 Mutual Visibility

Equation 4.5 accounts for parallax error, but the observed radiance in the corrected direction may still be incorrect as it does not account for visibility. If an object $\boldsymbol{o}$ lies between $\boldsymbol{p}$ and the intersection point $\Gamma(\boldsymbol{x}, \omega)$, then the radiance from $\boldsymbol{o}$ will be incorrectly accounted for, instead of the radiance from $\Gamma(\boldsymbol{x}, \omega)$ (Figure 4.3).

Previous methods before SRP have accounted for visibility, but only by checking between the receiver point $\boldsymbol{x}$ and probe $\boldsymbol{p}$ [42]. This check is not enough to be correct. The surface point $\Gamma(x, \omega)$ must be mutually visible to the receiver and the probe. Both the receiver and the probe must see $\Gamma(x, \omega)$.

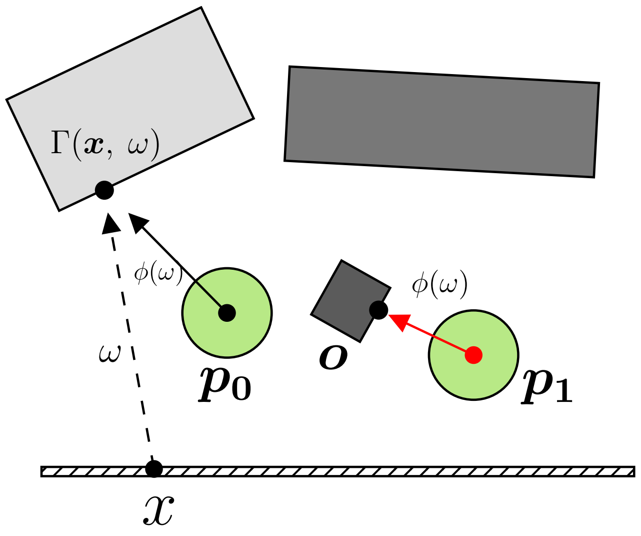


Figure 4.3: Mutual visibility. In this case, there is an object $\boldsymbol{o}$ between probe $\boldsymbol{p}_1$ and the surface point $\Gamma(\boldsymbol{x}, \omega)$. Without accounting for mutual visibility, probe $\boldsymbol{p}_1$ incorrectly samples from the object $\boldsymbol{o}$; instead of discarding the sample.

4.1.3 Interpolation Operator

SRP combines the parallax error correction and mutual visibility check into one interpolator:

$$L(\boldsymbol{x}, \omega) = \frac{\sum_i^n w_i(\boldsymbol{x}) V_i(\omega) L_p(\boldsymbol{p}_i, \phi(\omega))}{\sum_k^n w_k(\boldsymbol{x}) V_k(\omega)} \tag{4.6}$$

If all surfaces are diffuse, the interpolator (Equation 4.6) is exact given an infinite resolution sampling at the probes, and that the point $\Gamma(\boldsymbol{x}, \omega)$ can see at least one probe [6].

4.2 Band-Limited Probes

To make sparse interpolation viable for real-time use, SRP stills needs an efficient way to compute incoming radiance $L(\boldsymbol{p}_i, \omega)$ at each probe $\boldsymbol{p}_i$. Even though there are considerably fewer probes than receivers, a high-resolution sampling of the incoming radiance at each probe is still too expensive, both in terms of computation and in storage required. As a cheaper alternative, SRP band-limits the radiance at the probes by encoding it in a spherical harmonic basis:

$$L(\boldsymbol{p_i}, \omega) \approx \sum_j \lambda_{ij} Y_j(\omega) \tag{4.7}$$

where each radiance coefficient λ_{ij} is an element of the *probe radiance vector* $\boldsymbol{\lambda}$. This vector contains the radiance coefficients for all probes. For convenience, this thesis indexes $\boldsymbol{\lambda}$ with probe index i and the basis function index j.

Band-limiting radiance at the probes trades high-frequency information for much-needed execution speed and storage. For diffuse surfaces, the main source of high-frequency information comes from sharp changes in albedo or visibility. As an example, if an object casts a hard-edged shadow onto a surface, a low-frequency probe will not capture the sharp change in radiance it causes; this can cause areas to appear brighter than is correct. For spherical harmonics, it is possible to bring back high-frequency information by increasing the number of coefficients a probe uses; however, the cost increases quadratically, since each order n requires $(n+1)^2$ coefficients.

4.3 Precomputed Local Transport Operator

To combine the interpolation operator from Equation 4.6 with the band-limited probes from Equation 4.7, SRP defines a local transport operator $\mathcal{P}_x(\boldsymbol{\lambda})$ for every receiver $\boldsymbol{x}$. This operator transforms incoming radiance in the probe radiance vector $\boldsymbol{\lambda}$ into incoming indirect radiance at a receiver $\boldsymbol{x}$:

$$\mathcal{P}_x(\boldsymbol{\lambda}, \omega) = \frac{\sum_i^n w_i(\boldsymbol{x}) V_i(\omega) \sum_j \lambda_{ij} Y_j(\phi(\omega))}{\sum_k^n w_k(\boldsymbol{x}) V_k(\omega)} \tag{4.8}$$

While significantly cheaper than densely sampling the rendering equation at every receiver, the local transport operator is still too expensive to evaluate in real time. To see why this is, recall that the final goal of SRP is not to compute incoming indirect radiance at each receiver, but to compute outgoing indirect radiance towards the camera. To do this for diffuse surfaces, we must calculate irradiance over a hemisphere centred at $\boldsymbol{x}$:

$$I(\boldsymbol{x}) = \int_\Omega \mathcal{P}_x(\boldsymbol{\lambda}, \omega) \cos\theta d\omega \tag{4.9}$$

Again, we come up against an integral which is too expensive to sample in real time. As the mutual visibility and parallax-corrected radiance terms in the local transport operator depend on the variable we are integrating over ω, we must evaluate these terms many times to converge on the solution to the integral. To overcome this expense, SRP factors the probe radiance vector $\boldsymbol{\lambda}$ out of the integral and then precomputes the remaining components of the integral offline. This can be done because $\boldsymbol{\lambda}$ does not depend on the direction ω. Due to this precomputation, geometry must remain static, but lighting can still change at runtime. The next section covers the steps to factor $\boldsymbol{\lambda}$ out of Equation 4.9.

4.4 Irradiance Transport

This section describes how to factor the probe radiance vector $\boldsymbol{\lambda}$ out of the irradiance integral in Equation 4.9.

Starting with the irradiance transport integral from Equation 4.9:

$$\begin{aligned} I(\boldsymbol{x}) &= \int_\Omega \mathcal{P}_x(\boldsymbol{\lambda}, \omega) \cos\theta d\omega \\ &= \int_\Omega \frac{\sum_i^n w_i(\boldsymbol{x}) V_i(\omega) \sum_j \lambda_{ij} Y_j(\phi(\omega))}{\sum_k^n w_k(\boldsymbol{x}) V_k(\omega)} \cos\theta d\omega \end{aligned}$$

We bring the radiance coefficients out of the integral:

$$\begin{aligned} &= \int_\Omega \sum_i \sum_j \lambda_{ij} \frac{w_i(\boldsymbol{x}) V_i(\omega) Y_j(\phi(\omega))}{\sum_k w_k(\boldsymbol{x}) V_k(\omega)} \cos\theta d\omega \\ &= \sum_i \sum_j \lambda_{ij} \int_\Omega \frac{w_i(\boldsymbol{x}) V_i(\omega) Y_j(\phi(\omega))}{\sum_k^n w_k(\boldsymbol{x}) V_k(\omega)} \cos\theta d\omega \end{aligned}$$

Then, we define the transport kernel $K_{ij}(\boldsymbol{x}, \omega)$:

$$K_{ij}(\boldsymbol{x}, \omega) = \begin{cases} \frac{w_i(\boldsymbol{x}) V_i(\omega) Y_j(\phi(\omega))}{\sum_k^n w_k(\boldsymbol{x}) V_k(\omega)} & \text{if } \sum_k^n w_k(\boldsymbol{x}) V_k(\omega) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

Leaving us with:

$$I(\boldsymbol{x}) = \sum_i \sum_j \lambda_{ij} \int_\Omega K_{ij}(\boldsymbol{x}, \omega) \cos\theta d\omega \tag{4.11}$$

SRP precomputes the integral in Equation 4.11 yielding a transport coefficient α_{ij} for each probe i and basis function j at a receiver $\boldsymbol{x}$:

$$\alpha_{ij} = \int_\Omega K_{ij}(\boldsymbol{x}, \omega) \cos\theta d\omega \tag{4.12}$$

$$I(\boldsymbol{x}) = \sum_i \sum_j \lambda_{ij} \alpha_{ij} \tag{4.13}$$

Equation 4.13 provides an efficient way to transform the incoming irradiance at the probes into indirect irradiance at a receiver point. To convert to outgoing indirect radiance, we multiply at runtime by the Lambertian BRDF $\frac{\rho}{\pi}$.

4.5 Clustered Principal Component Analysis

Sections 4.1-4.3 show how SRP reduces evaluating indirect diffuse irradiance at a receiver $\boldsymbol{x}$ to a dot product between the probe radiance vector $\boldsymbol{\lambda}$ and a precomputed transport vector $\boldsymbol{\alpha}$ (Equation 4.13). This is efficient to evaluate but impractical as storing a transport vector $\boldsymbol{\alpha}$ per receiver requires a significant amount of memory. To reduce the amount of memory, SRP compresses the transport vectors using clustered principal component analysis (CPCA) [17].

CPCA is an important part of SRP's scalability. A medium-sized scene with around $200\,000$ receivers, 100 probes, and 192 coefficients (order-7 spherical harmonics) would use $28.61\,\mathrm{GB}$ if stored in 16-bit floating point. To put this in perspective, an iPhone 7 only has $2\,\mathrm{GB}$ of shared memory between the GPU and CPU, and the largest amount of memory available on current high-end GPUs like the AMD FirePro W1900 or NVIDIA Quadro GV100 is $32\,\mathrm{GB}$.

This section describes how CPCA compresses the transport vectors for SRP. CPCA works by dividing receivers into clusters. In each cluster, it uses principal component analysis (PCA) to find the axes which best describe the variation in the transport vectors. The top n_c axes are kept along with weights which describe how to reconstruct the transport vectors using the n_c axes. As n_c lowers, less memory is used, but the higher the error in the reconstructed transport vectors.

4.5.1 Cluster Compression

To compress the transport vectors within a cluster, we perform PCA on each cluster. To do this, we first define a *cluster transport matrix* $\boldsymbol{T}_c$ where the ith row of T_c contains the transport vector for the ith receiver in a cluster.

We then compute the SVD of T_c:

$$\underset{n\times n}{T_c} = \underset{n\times n}{U} \times \underset{n\times n_i n_j}{\Sigma} \times \underset{n_i n_j\times n_i n_j}{V^T} \tag{4.14}$$

The SVD decomposes T_c into three matrices: U, Σ, and V^T. U is a $n \times n$ matrix where n is the number of receivers in the cluster, Σ is a $n \times n_i n_j$ matrix, and V^T is a $n_i n_j \times n_i n_j$ matrix where n_i is the number of probes, and n_j is the number of basis functions used to represent radiance at a probe. The notation $n \times n_i n_j$ is short for $n \times (n_i \times n_j)$, where $n_i \times n_j$ is a scalar; in this case, n is the number of rows in the matrix, and $n_i \times n_j$ is the number of columns.

To compress T_c, we keep only the first n_c columns of the matrix U and the first n_c rows of the matrix Σ:

$$\underset{n\times n}{T_c} \approx \underset{n\times n_c}{U_c} \times \underset{n_c\times n_i n_j}{\Sigma_c} \times \underset{n_i n_j\times n_i n_j}{V^T} \tag{4.15}$$

To evaluate indirect irradiance at runtime, we save two matrices. The *cluster projection matrix* $\Sigma_c V^T$, which contains the top n_c principal axes; and the *receiver reconstruction matrix* U_c, which contains reconstruction weights for each receiver in the cluster. Section 4.5.2 describes how these matrices are used to evaluate indirect irradiance at runtime.

As the receivers in a cluster are near each other, each cluster only uses a small subset of the total probes in a scene. This means the cluster projection matrix $\Sigma_c V^T$ contains many zero-columns. Further compression is possible if only the non-zero columns and their offsets are stored.

4.5.2 Runtime Reconstruction

At runtime, SRP evaluates indirect illumination in two steps. First, each cluster projection matrix $\Sigma_c V^T$ is multiplied with the probe radiance vector $\boldsymbol{\lambda}$ to produce a n_c dimensional *light basis vector* $\boldsymbol{l}$ for each cluster. Second, the receiver reconstruction matrix U_c is multiplied with the light basis vector $\boldsymbol{l}$ to produce an n_k dimensional vector containing the diffuse indirect irradiance for each receiver in the cluster. To convert irradiance to outgoing radiance when shading, each element of the n_k dimensional vector is multiplied by the surface albedo and divided by π.

Chapter 5

Sparse Radiance Probes: Implementation

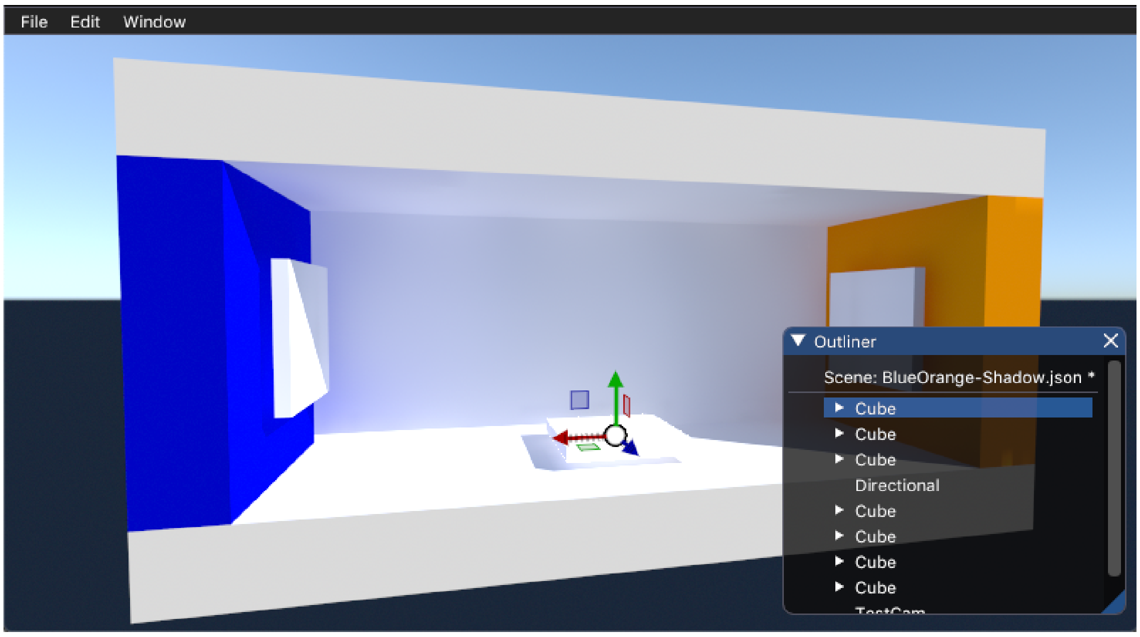


Figure 5.1: Sparse Radiance Probes implementation running in LlamaEngine.

This chapter covers my implementation of SRP (Figure 5.1). It describes the runtime implementation first in section 5.1, followed by the pipeline for precomputing the transport coefficients in section 5.6. I developed the implementation within LlamaEngine, a 3D rasteriser and rendering framework developed by myself and a fellow Master's student Thomas Roughton. Appendix A provides more details on LlamaEngine.

5.1 Runtime Overview

At runtime, the implementation is split across several passes on the GPU. These passes are listed below. Passes 2-5 reference sections which describe them in more detail. Figure 5.2 shows how data flows between these passes and where precomputed data is used.

1. **Lightmap Update**
   This pass rasterises the direct illumination in the scene to a lightmap.

   After the first loop of the algorithm, this pass also accounts for secondary indirect light bounces. It adds the contribution of the indirect illumination lightmap from the previous frame to the direct illumination lightmap. This creates a feedback loop, as the probes updated in the next pass now see the indirect lighting from the previous frames.

2. **Probe Update** (Section 5.2)
   This pass uses the lightmap from the previous step to update the probe radiance vector $\boldsymbol{\lambda}$.

3. **Cluster SVD Projection** (Section 5.3)
   This pass computes a light basis vector $\boldsymbol{l}$ for every cluster p by multiplying each precomputed cluster projection matrix by the probe radiance vector $\boldsymbol{\lambda}$.

4. **Irradiance Transport** (Section 5.3)
   This pass takes in the light basis vectors $[\boldsymbol{l}]_p$[1] from the previous pass and multiplies each one with the corresponding receiver reconstruction matrix $[U_c]_p$. This results in a n-dimensional vector which contains the reconstructed irradiance for the nth receiver in that cluster. The pass saves the reconstructed irradiance at each receiver's location in a new indirect illumination lightmap $\boldsymbol{\kappa}$.

5. **Forward Render Pass** (Section 5.4)
   Finally, the forward render pass samples the indirect irradiance from the indirect illumination lightmap $\boldsymbol{\kappa}$ and adds it the direct lighting. A dilation pass is also run before the forward pass on the lightmap to remove dark seams around geometry edges (section 5.5).

[1]Per cluster vectors and matrices are indexed with square brackets. e.g. $[\boldsymbol{l}]_0$ is the light basis vector for cluster 0.

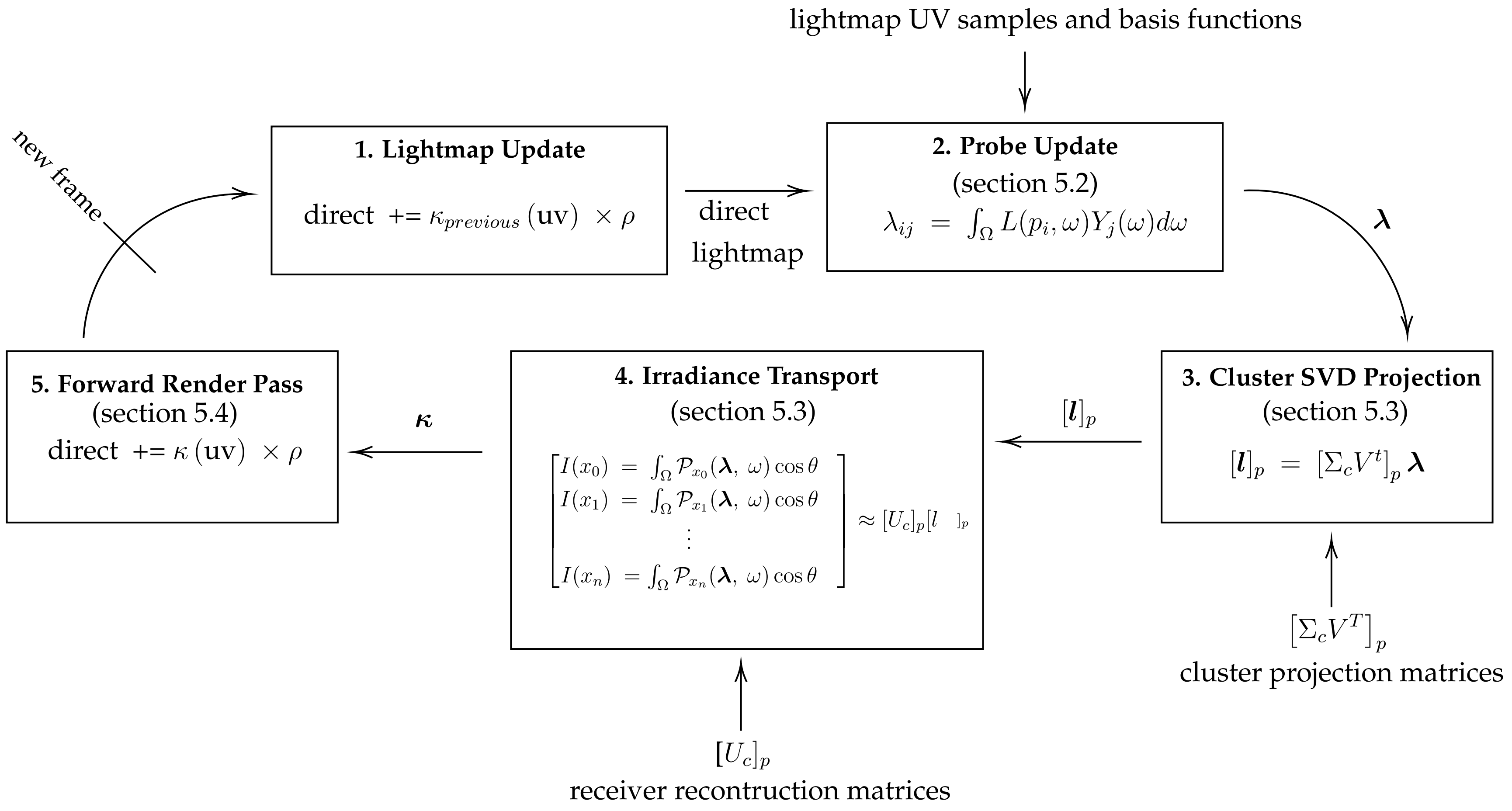

lightmap UV samples and basis functions
new frame
1. Lightmap Update
direct += $\kappa_{previous}$ (uv) $\times \rho$
direct lightmap
2. Probe Update
(section 5.2)
$\lambda_{ij} = \int_\Omega L(p_i,\omega)Y_j(\omega)d\omega$
$\boldsymbol{\lambda}$
3. Cluster SVD Projection
(section 5.3)
$[\boldsymbol{l}]_p = [\Sigma_c V^t]_p \boldsymbol{\lambda}$
$[\Sigma_c V^T]_p$
cluster projection matrices
$[\boldsymbol{l}]_p$
4. Irradiance Transport
(section 5.3)
$\begin{bmatrix} I(x_0) = \int_\Omega \mathcal{P}_{x_0}(\boldsymbol{\lambda}, \omega)\cos\theta \\ I(x_1) = \int_\Omega \mathcal{P}_{x_1}(\boldsymbol{\lambda}, \omega)\cos\theta \\ \vdots \\ I(x_n) = \int_\Omega \mathcal{P}_{x_n}(\boldsymbol{\lambda}, \omega)\cos\theta \end{bmatrix} \approx [U_c]_p[l\ \ ]_p$
$[U_c]_p$
receiver recontruction matrices
$\boldsymbol{\kappa}$
5. Forward Render Pass
(section 5.4)
direct += κ (uv) $\times \rho$

Figure 5.2: Flow diagram of the passes used in my SRP implementation. Each box represents a pass, and the arrows between the boxes show how data flows between passes. Arrows originating outside boxes represent precomputed data. Section 5.1 provides an overview for each pass. κ is an indirect illumination lightmap, ρ is the albedo from the Lambertian BRDF; $\boldsymbol{\lambda}$ is the probe radiance vector (section 4.2); U_c, Σ_c, and V^t are CPCA matrices used to reconstruct the transport vectors (section 4.5.1); and $\boldsymbol{l}$ is the light basis vector (section 4.5.2). As there is a set of CPCA matrices and a light basis vector for each cluster, each is indexed by the cluster index p (e.g. $[\boldsymbol{l}]_p$ is the light basis vector for cluster p).

5.2 Radiance Coefficient Generation

SRP transforms direct illumination stored in the probe radiance vector $\boldsymbol{\lambda}$ into indirect irradiance at a receiver by convolving it (which can be performed with a dot product) with a precomputed transport vector $\boldsymbol{\alpha}$. GPUs can perform this operation in real time, but we still must update $\boldsymbol{\lambda}$ every frame to support real-time dynamic changes in lighting. To update $\boldsymbol{\lambda}$, we must project the radiance at each probe into spherical harmonics using the following equation:

$$\lambda_{ij} = \int L(\boldsymbol{p}_i, \omega) Y_j(\omega) d\omega \tag{5.1}$$

My implementation solves this equation using Monte Carlo integration on the GPU. If needed, section 2.4 and 2.5 cover necessary background on spherical harmonics and Monte Carlo integration respectively. To describe my implementation, I start with a serial implementation on the CPU (Source Code 5.1) and then show how to convert it to a compute shader implementation on the GPU (Source Code 5.3). The serial implementation in Source Code 5.1 loops through all of the probes in a scene and computes the jth basis coefficient by summing each sample, $L(\boldsymbol{p}_i, \omega)Y_j(\omega)$, multiplied by a weight. As the samples are uniformly distributed over the unit sphere, the weight is $\frac{4\pi}{N}$ where N is the number of sample directions.

To evaluate the incoming radiance $L(\boldsymbol{p_i}, \omega)$ at a probe, the algorithm samples from either a direct illumination lightmap or an environment map. If a ray fired from a probe in a sample direction hits a surface, the algorithm samples from a direct illumination lightmap; if it misses a surface, it samples from an environment map. As casting a ray for every sample at runtime is too expensive, the samples are precomputed; for samples that hit scene surfaces, the lightmap UV coordinate of the hit location is saved, while the sample direction is saved for samples that miss. The basis functions are also precomputed in the directions of the precomputed samples. The only operations we must perform at runtime are sampling from the lightmap or environment map, multiplying by the appropriate precomputed basis function and weight, and summing the irradiance samples. Source Code 5.2 shows an updated serial algorithm which uses two precomputed sample buffers: one buffer of lightmap UVs for the samples that hit and another buffer of directions for samples that missed.

```
for i in 0..<p.count {
  for omega in p[i].sampleDirections {
    for j in 0..<coefficientsPerProbeCount {
      lambda[i][j] += L(p[i], omega) * Y(j, omega) * weight
    }
  }
}
```

Source Code 5.1: Serial probe projection code on the CPU. `p` is an array of probes, `p[i].sampleDirections` is an array of uniformly distributed sample directions over a unit sphere, `lambda` is the probe radiance vector, `L` is a function returning the incoming radiance at a probe in a direction, `Y(j, omega)` evaluates the jth spherical harmonics basis function in the direction `omega`. Lastly, `weight` is $\frac{4\pi}{N}$ (the surface area of the unit sphere, and the inverse of the sampling probability distribution function (PDF) for the rays) where N is the number of sample directions per probe.

```
for i in 0..<p.count {
  // Accumulate for samples that hit geometry.
  for uv in p[i].hitSampleDirections {
    let sample = lightmap.sample(uv)
    for j in 0..<coefficientsPerProbeCount {
      lambda[i][j] += sample * Y[i][j] //
    }
  }

  // Accumulate for samples that missed geometry.
  for omega in p[i].missSampleDirections {
    let sample = environmentMap.evaluate(omega)
    for j in 0..<m {
      lambda[i][j] += sample * Y[i][j] //
    }
  }
}
```

Source Code 5.2: Serial probe projection code on the CPU using precomputed sample data. Precomputed sample rays that hit surfaces sample from a lightmap using the UV coordinates of the hit location. Precomputed sample rays that miss sample from an environment map. `Y` is a buffer of basis functions evaluated in the precomputed sample directions and multiplied by the weight $\frac{4\pi}{N}$.

To implement this algorithm on the GPU, we must split the work into smaller chunks so it can execute the work in parallel. I perform the projection for all probes in a single compute pass. The pass assigns each probe to its own threadgroup [2] and then each thread within that threadgroup handles a subset of the total samples for a probe. More specifically, each thread performs the sums on lines 6 and 14 of the serial algorithm in Source Code 5.2 for the subset of samples it is assigned. Each thread saves the result of each sum in threadgroup memory and then waits at a threadgroup memory barrier. This barrier blocks the execution of a thread until all threads in its threadgroup have reached the barrier. The first j threads in each threadgroup then sum the sums stored in threadgroup memory to produce each coefficient λ_{ij} for a probe. Source Code 5.3 contains pseudocode for the compute kernel.

5.2.1 Sample Generation

For our use case, Monte Carlo integration requires uniformly distributed samples to produce correct results [15]. For each probe $\boldsymbol{p_i}$, I generate uniform samples over a sphere centred at the probe using *stratified sampling*.

Stratified sampling works in two steps. First, we evenly distribute n samples over a unit square. To do this, we divide the square into a $\sqrt{n} \times \sqrt{n}$ grid and then choose a random sample point (x, y) within each cell. Second, we map each sample point (x, y) to the surface of the unit sphere (θ, ϕ):

$$\begin{bmatrix} \theta \\ \phi \end{bmatrix} = \begin{bmatrix} 2\cos^{-1}(\sqrt{1-x}) \\ 2\pi y \end{bmatrix} \tag{5.2}$$

[2]I use *threadgroup* and other terminology from the Metal Shading Language to describe my implementation; see section 2.6 for definitions.

```
 1  kernel void projectProbes() {
 2    // Offset for threadgroupSums buffer.
 3    int o = threadIndexInThreadgroup * coefficientsPerProbeCount;
 4
 5    for(int i = 0; i < hitSamplesPerThread; i += 1) {
 6      float2 uv = hitSampleDirections[o + i];
 7      float3 s = lightmap.sample(uv);
 8      for(int j = 0; j < coefficientsPerProbeCount; j += 1) {
 9        threadgroupSums[o + j] += s * Y[threadIndex * i][j];
10      }
11    }
12
13    for(int i = 0; i < missSamplesPerThread; i += 1) {
14      float3 direction = missSamples[i];
15      float3 sample = environmentMap.evaluate(direction);
16      for(int j = 0; j < coefficientsPerProbeCount; j += 1) {
17        threadgroupSums[o + j] += s * Y[threadIndex * i][j];
18      }
19    }
20
21    threadgroup_barrier(mem_flags::mem_threadgroup);
22
23    if (threadgroupIndex < coefficientsPerProbeCount) {
24      lambda[threadgroupIndex] = sumAt(threadgroupIndex, threadgroupSums);
25    }
26  }
```

Source Code 5.3: Parallel probe projection code on the GPU. For clarity and to save space, arguments to the kernel function and offsets to account for multiple probes are omitted. Each thread performs the sums for a subset of the total samples for a probe (lines 5 to 19). All threads then wait until every thread has reached the threadgroup memory barrier (line 21). Lastly, the first j threads then sum the sums saved in threadgroup memory to produce each coefficient λ_{ij} using the `sumAt` function (lines 23 to 25). `Y` is a buffer of basis functions evaluated in the precomputed sample directions and multiplied by the weight $\frac{4\pi}{N}$.

5.3 Cluster SVD Projection and Irradiance Transport

To produce the indirect illumination lightmap, two matrix multiplications must be performed (section 4.5.2). I use one compute pass for each matrix multiplication.

The first compute pass, cluster SVD projection, multiplies the cluster projection matrix $\Sigma_c V^T$ by the probe radiance vector $\boldsymbol{\lambda}$ and outputs a n_c dimensional *light basis vector* $\boldsymbol{l}$ for each cluster. To do this, each thread in the pass multiplies one row of the cluster projection matrix by the probe radiance vector. Most of the columns of the cluster projection matrix contain only zeroes as probes which lie outside the cut-off radius for all receivers in a cluster make no contribution. These columns are not stored. Instead, the thread loops through the number of non-zero columns in the cluster projection matrix, and then uses a buffer to map each non-zero column index to its actual column index in the original matrix so that the result of the multiplication can be written to the correct element of the light basis vector.

The second pass, irradiance transport, reconstructs the irradiance at each receiver in a cluster. To do this, it multiplies the receiver reconstruction matrices with the corresponding light basis vector to produce the reconstructed indirect irradiance for each receiver in a cluster. Each thread performs the multiplication for a single receiver by multiplying a row of the receiver reconstruction matrix for a cluster by the light basis vector for that cluster. The multiplication produces the reconstructed indirect irradiance for that receiver which is then saved into an indirect illumination lightmap at the corresponding texel for that receiver.

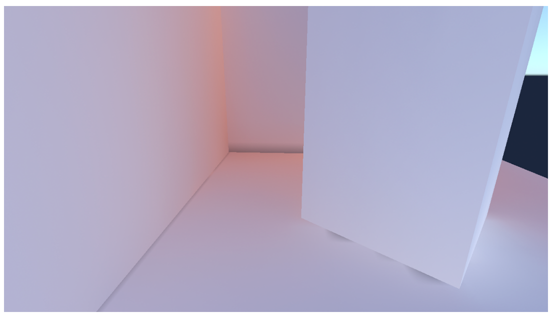
5.4 Forward Render Pass

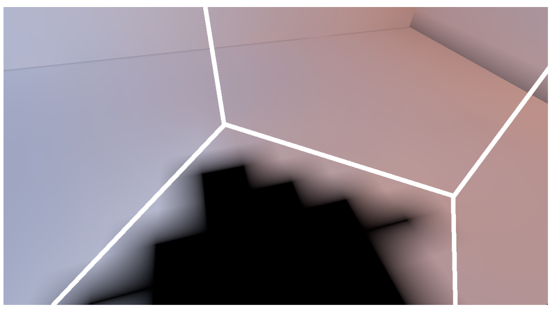
In my implementation, a forward render pass performs the last step of adding the contribution of the indirect lighting to the direct lighting by sampling from the indirect illumination lightmap. When sampling from the texture, it uses a linear texture filter to produce a smoothly blended result. While a forward renderer is used here, this could equally be performed in the geometry pass of a deferred lighting setup.

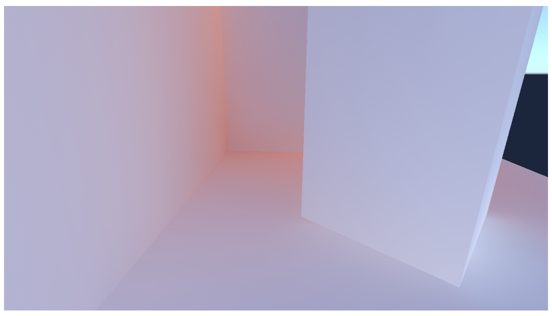
5.5 Lightmap Dilation

Dark seams can occur around the edges of geometry due to the limited resolution of lightmaps (Figures 5.3a and 5.3b). As texels inside of geometry are black, when the forward render pass samples the lightmap using a linear filter, it blends the black texels inside of geometry with the texels outside of geometry causing dark seams. To fix this, a dilation pass is run which extends valid texels that are beside invalid black texels out by a one texel radius so that there are no longer any black texels near the edges of geometry (Figures 5.3c and 5.3d).

I implement dilation in a compute shader pass, which runs over all texels in the lightmap. If a texel has an alpha of one, then the texel is valid and the shader leaves it as it is; however, if a texel has an alpha of zero, it is invalid, and so the mean contribution of its valid neighbouring texels is used instead.


(a) Without lightmap dilation.


(b) View from inside cuboid in (a)


(c) With lightmap dilation

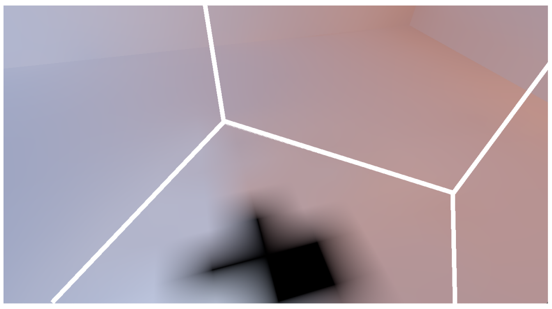

(d) View from inside cuboid in (c)

Figure 5.3: With and without lightmap dilation. Dark seams appear around the edges of geometry in 5.3a due to invalid black receivers inside geometry blending with valid receivers outside geometry. 5.3b shows these invalid texels from within the cubeoid in 5.3a. 5.3d shows how lightmap dilation removes the seams by extending valid texels so the invalid texels are no longer near edges.

5.6 Transport Coefficient Precomputation

This section describes my implementation of a pipeline for precomputing SRP's transport coefficients. Although the transport coefficient precomputation is run on a high-end device, it still plays a part in the scalability of SRP as parameters chosen for precomputation change SRP's quality and performance at runtime. The number of receivers and their positions, the number of probes and their positions, the number of basis coefficients per probe, and the the number of PCA vectors for compression are all parameters set at precomputation time which affect the quality and performance of SRP.

One challenge of a GPU implementation is the limited memory available. As discussed in section 4.5 on CPCA compression, storing a transport coefficient for every receiver and every basis consumes many gigabytes of memory. This means it is not possible to compute all transport coefficients at once. Fortunately, each transport coefficient can be solved for independently which means the precomputation can be performed in batches. Each batch computes the transport coefficients for a portion of the receivers. The method computes the transport coefficients for each batch on the GPU and then copies them to the CPU. It then performs CPCA compression for that batch while computing the transport coefficients for the next batch on the GPU at the same time. The process repeats until all batches are complete.

For irradiance transport, the goal is to compute a transfer coefficient vector $\boldsymbol{\alpha}$ for every receiver $\boldsymbol{x}$. To do this, we must solve the following integral (derived in section 4.4) for every probe i and basis function j:

$$\alpha_{ij} = \int_{\Omega} K_{ij}(\boldsymbol{x}, \omega) \cos\theta d\omega \tag{5.3}$$

To compute the transfer coefficients for each batch, I use a method which progressively solves the integral in equation 5.3, in a manner similar to how GPU path tracers like AMD's RadeonProRender [51] progressively solve the rendering equation. The method runs in passes, where one sample is taken per receiver per pass. The more passes, the more samples per receiver, and the better the estimate for each transport coefficient becomes.

Sections 5.6.1 to 5.6.5 cover how I compute the transport coefficients on the GPU, and section 5.6.6 describes how I compress the coefficients using CPCA on the CPU.

5.6.1 Overview

The method I use to compute the transport coefficients solves Equation 5.3 progressively in passes. The goal of each pass is to take one sample of the transport kernel K_{ij} for each receiver in the batch. In each pass, the following five steps are performed:

1. **Primary Ray Generation** (Section 5.6.2)

   Generate a ray for each lightmap texel (i.e. the receiver points $\boldsymbol{x}$). The direction of the ray is chosen from the hemisphere centred around the worldspace normal of the texel.

2. **Primary Ray Intersection** (Section 5.6.3)

   Run an intersection query for the generated rays.

3. **Mutual Visibility Ray Generation** (Section 5.6.4)

   For each primary ray, and for all the probes that are within the cut-off radius of the receiver the ray was fired from:

   If the primary ray hit, generate a ray from the probe to the intersection point. This ray is used in the next step to make sure the probe can see the hit point.

   If the primary ray missed, generate a ray from the probe in the direction of the primary ray. This ray is used in the next step to make sure the probe can see the environment lighting.

4. **Mutual Visibility Check** (Section 5.6.3)

   Run an occlusion query for rays generated in previous step to check visibility between the probe and intersection or the probe and the environment lighting.

5. **Accumulation** (Section 5.6.5)

   Steps 1-4 generate the data necessary to perform one step of Monte Carlo integration. To accumulate, for every receiver, sample each transport kernel $K_{ij}(\boldsymbol{x}, \omega)$ for each nearby probe i, and basis function j, and then accumulate each sample progressively using Welford's method [52].

5.6.2 Primary Ray Generation

The first step for each batch is generating the primary rays from the receiver points (Figure 5.4). To do this, my implementation uses a compute shader which runs over all valid lightmap texels [3]. The compute shader I used to do this was based on a shader written by Thomas Roughton for their Master's thesis on path-traced lightmaps [53].

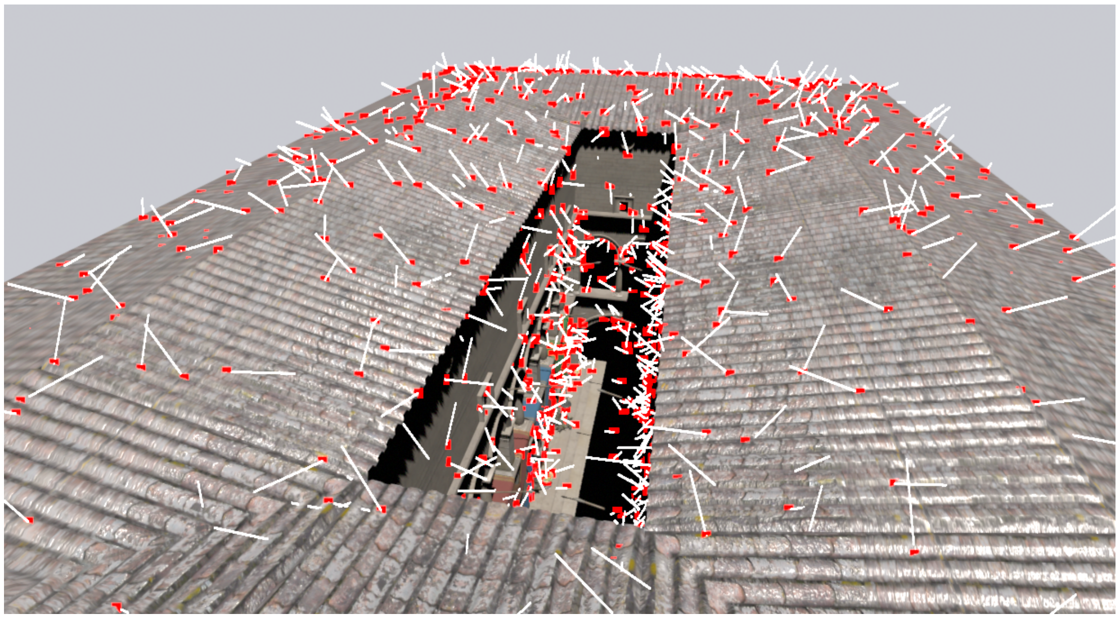


Figure 5.4: Visualisation of rays generated in the primary ray generation pass. The red dots are ray origins and the white lines are ray directions.

[3]Not all texels in a lightmap are valid. Lightmap packing algorithms aim to pack surfaces as tightly as possible into a two-dimensional texture but there will still be gaps. The receivers for SRP are texels which map to a valid surfaces.

Each thread in the compute shader generates a ray. The ray's origin is a point chosen randomly within the lightmap texel, and the ray's direction is chosen using cosine weighted sampling over a hemisphere centred at the normal of the receiver point.

Using cosine weighted samples has two benefits over uniform sampling. First, Monte Carlo integration converges faster on a solution if the samples are taken from a distribution similar to the integral it is solving [15]. Second, when sampling from a non-uniform distribution, each sample of the function being integrated must be divided by the inverse of the probability distribution function (PDF). For Equation 5.3, dividing by the inverse PDF cancels with cosine term in the integral we are solving. This means it is not necessary to multiply by cosine when accumulating the samples in step 5.

To show this mathematically, the estimator for the transport coefficient where p is the PDF is:

$$F_N = \frac{1}{N}\sum_{ij}\frac{K_{ij}(\boldsymbol{x},\omega)\cos\theta}{p(\omega)} \tag{5.4}$$

For cosine weighted samples, the PDF for sampling over a hemisphere is $\frac{cos\theta}{\pi}$:

$$F_N = \frac{1}{N}\sum_{ij}\frac{K_{ij}(\boldsymbol{x},\omega)\cos\theta}{\frac{cos\theta}{\pi}} \tag{5.5}$$

Which simplifies to:

$$F_N = \frac{\pi}{N}\sum_{ij}K_{ij}(\boldsymbol{x},\omega) \tag{5.6}$$

The factor of π can also be ignored if we compensate by not dividing the diffuse albedo by π at runtime.

5.6.3 Primary Ray Intersection and Mutual Visibility Check

Both the primary ray intersection and mutual visibility check steps require ray casting. LlamaEngine has integrated support for both Radeon Rays [54] and Metal Performance Shaders [55], which both provide compute kernels for accelerated ray intersection and occlusion queries. An intersection query returns information about the surface the ray made an intersection with, while an occlusion query only returns whether the ray hit or missed.

5.6.4 Mutual Visibility Ray Generation

This step generates the rays that the next step uses to check for mutual visibility.

For each primary ray, a compute shader generates a mutual visibility ray for every probe that contributes to the receiver the ray was fired from. A probe contributes if the weight term $w(\boldsymbol{x})$ in the transport kernel is non-zero. In my implementation, this means that the probe is within a certain radius of the receiver. Therefore, I refer to these probes as *nearby* probes.

If the primary ray hits a surface, the mutual visibility ray for each nearby probe points from the probe towards the intersection point. This makes sure both the receiver and the probe can see the intersection point. I refer to these rays as *probe-to-intersection* rays.

If the primary ray does not hit any surfaces, the mutual visibility ray for each nearby probe instead points in the same direction as the primary ray. This makes sure both the receiver and the probe can see the same point in the environment lighting. Parallax error is not an issue with environment lighting as it is assumed to be infinitely far away. I refer to these rays as *probe-to-environment* rays.

Probe-to-environment rays are important regardless of whether a scene includes environment lighting or not. Every transport coefficient still needs to encode the fact that there is no radiance coming from a direction. Figure 5.5 shows the difference before and after accounting for probe-to-environment rays.

The direction of the probe-to-intersection ray is the parallax-corrected direction $\phi(\omega)$. The direction of the probe-to-environment ray is also technically parallax corrected but is equal to the original direction ω. Step 5 uses the corrected direction when evaluating the basis function in each transport kernel.

Lastly, there are two validation checks for probe-to-intersection rays. These rays are marked as inactive if they do not pass these checks:

1. The probe must be in front of the intersection surface. Any probes behind the surface of an intersection cannot see the surface.

2. The receiver-to-intersection direction must point in the opposite direction to the intersection's surface normal. Otherwise, the surface is facing away from the receiver.

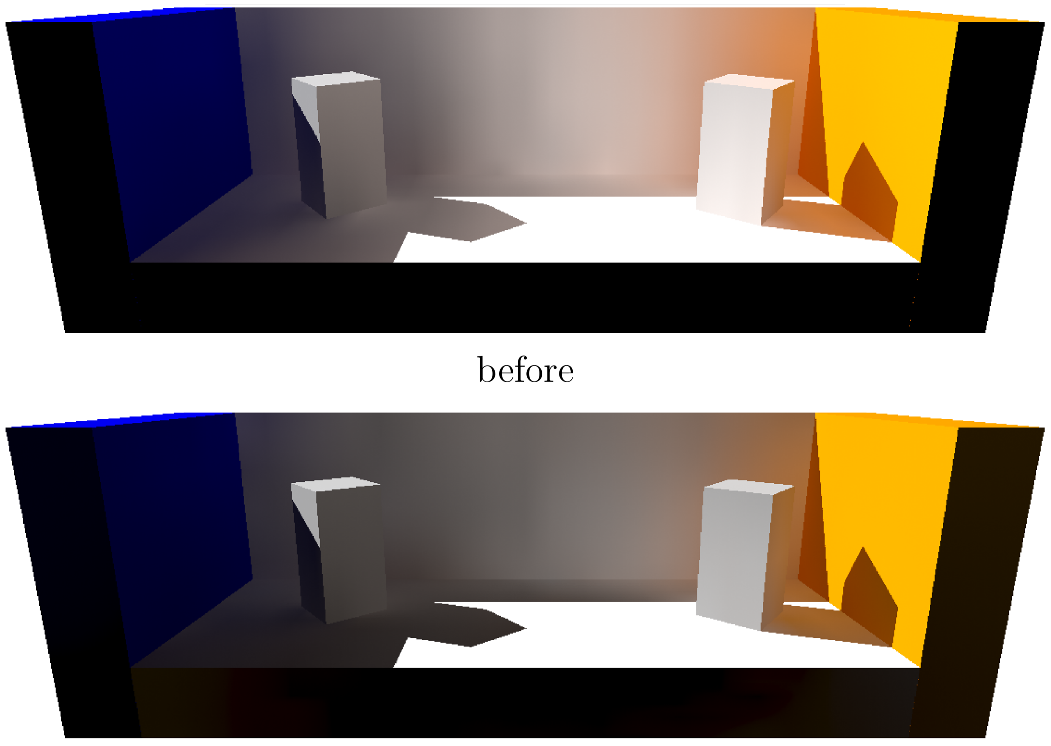


Figure 5.5: Before and after accounting for probe-to-environment rays. Both renders used one directional light and had no environment lighting. In the top render, probe-to-environment rays were not accounted for in the mutual visibility ray generation step. This results in the image appearing too bright as even though there is no environment lighting, the transport coefficients still need to encode the fact that there is no radiance in a direction. In the bottom render, the probe-to-environment rays have been correctly accounted for.

5.6.5 Accumulation

The last step in a pass accumulates one sample of the transport kernel $K_{ij}(\boldsymbol{x}, \omega)$ for every receiver $\boldsymbol{x}$, nearby probe i, and basis function j. Recall the transport kernel from Equation 4.10 in section 4.4:

$$K_{ij}(\boldsymbol{x}, \omega) = \begin{cases} \frac{w_i(\boldsymbol{x}) V_i(\omega) Y_j(\phi(\omega))}{\sum_k^n w_k(\boldsymbol{x}) V_k(\omega)} & \text{if } \sum_k^n w_k(\boldsymbol{x}) V_k(\omega) > 0 \\ 0 & \text{otherwise} \end{cases}$$

To perform the accumulation, a compute shader runs over all the receivers in the batch. Each thread of the compute shader first computes the denominator shared between all the transport kernels; i.e. the total weighted-visibility for all probes. To avoid looping through all probes, the thread looks up a buffer precomputed on the CPU that contains a list of nearby probes for the current receiver. The thread then sums the weights for each of the nearby probes which pass the mutual visibility test. Mutual visibility is checked by reading from a buffer generated by the occlusion pass in step 4.

If the total weighted-visibility is not zero, then for every probe that passes the mutual visibility test, the thread updates the current estimate for the corresponding transport coefficient a_{ij} by accumulating the contribution of the transport kernel K_{ij} for every basis function j in the corrected direction using Welford's method.

Welford's method [52] is a single-pass algorithm for computing the mean of a data set as samples become available. The method keeps track of the current mean and weight of all samples. A new sample is added by subtracting the mean from the new sample and multiplying by the weight.

If the total weighted-visibility is zero, then the sample is invalid. Ideally, the total-weighted visibility will never be zero as there should be at least one nearby probe for every sample that is mutually visible to a receiver and intersection point. However, it is hard to guarantee this if probes are placed manually like they are in my implementation. Additionally, an edge case occurs when a lightmap texel is both inside and outside geometry (Figure 5.6). As the origin of a primary ray can be offset anywhere within a texel, a sample ray's origin could be inside geometry in some passes and outside geometry in others. When it is inside

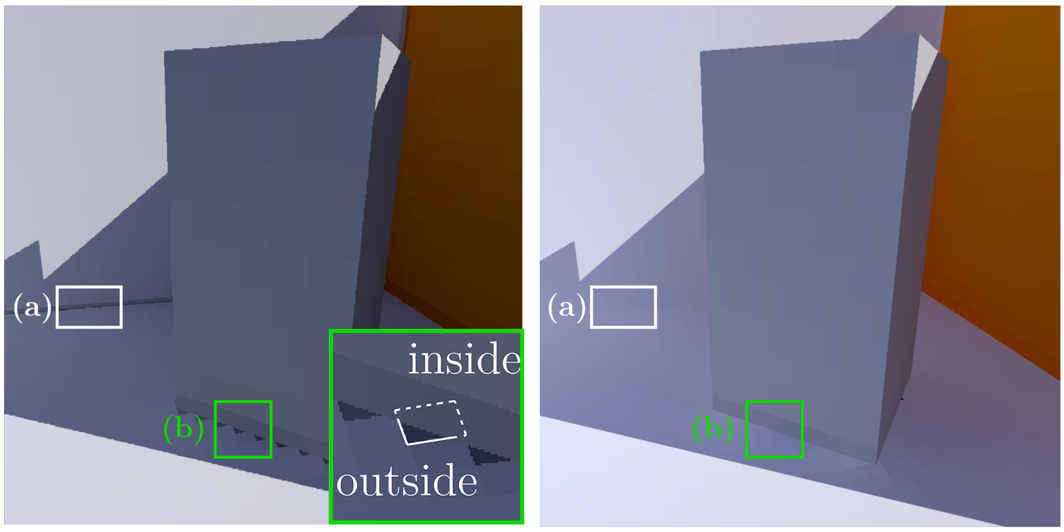


Figure 5.6: *Left:* Using an incorrect per sample weight. In general, the render appears too dark. Additionally, dark seams appear near edges in geometry (regions (a) and (b)). The green region (b) is enlarged to show a receiver inside and outside geometry. The outlined square in the enlarged green region is a single receiver. The dotted line is inside the geometry, and the solid line is outside. Samples inside the geometry incorrectly contribute zero radiance so when they are mixed with correct samples outside geometry the receiver appears darker than it should. *Right:* Using the adjusted sample weight correctly accounts for invalid samples.

geometry, the receiver cannot see any probes and so the total weighted-visibility is zero making the sample invalid.

The per sample weight must be adjusted to account for these invalid samples. If all samples were valid, then the per sample weight would be $\frac{1}{N}$ where N is the number of passes. However, if this per sample weight is used then areas will appear darker than they should (Figure 5.6 left) as invalid samples are accumulated as if no radiance came from the sample direction. Instead, the per sample weight is $\frac{1}{M}$ where M is the number of valid samples.

5.6.6 Clustered Principal Component Analysis

After the transport coefficients have been computed for a pass, they are compressed on the CPU using CPCA. My implementation is mainly based off the original paper by Sloan et al. [17] but I have only implemented the static version of CPCA without iterative or adaptive refinement.

For each cluster in a batch, I create a cluster transport matrix $\boldsymbol{T}_c$ where each row contains the transport coefficients for each receiver in a cluster. To compute the SVD of this matrix, I use the `dgesvd` routine from the LAPACK library [56].

The clusters CPCA compresses can be computed in multiple ways. Sloan et al. [17] use the Linde–Buzo–Gray algorithm [57] to perform the clustering but Silvennoinen and Lehtinen [6] instead use an axis-aligned bounding box tree (AABB) clustering technique as it produces smaller clusters. This speeds up compression as computing the SVD for large matrices is expensive.

To implement AABB clustering, I start with an AABB enclosing all the receivers at the root node, the AABB is divided in half by making a split orthogonal to its longest axis, and a child node is added for each half (Figure 5.7). I repeat this process recursively until the leaves of the AABB tree contain at most 1024 receivers.

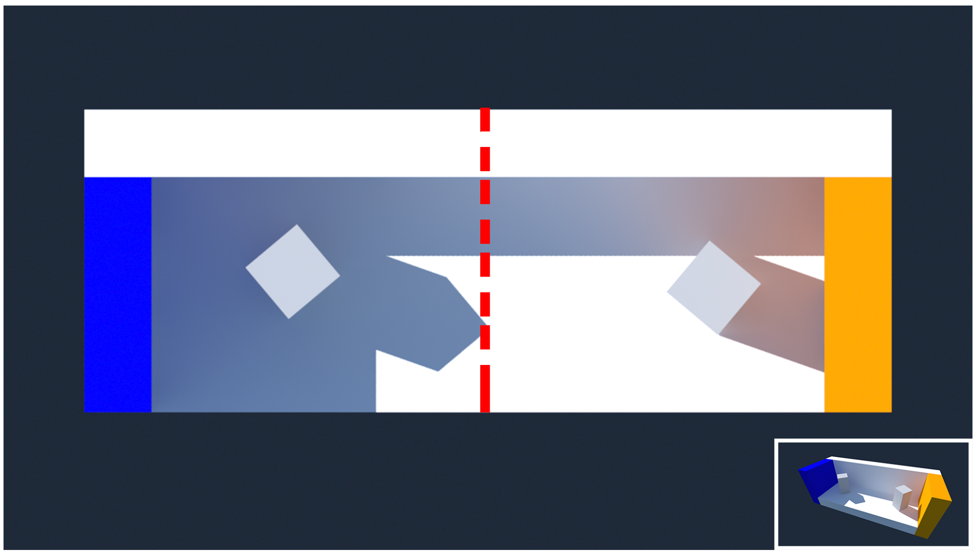
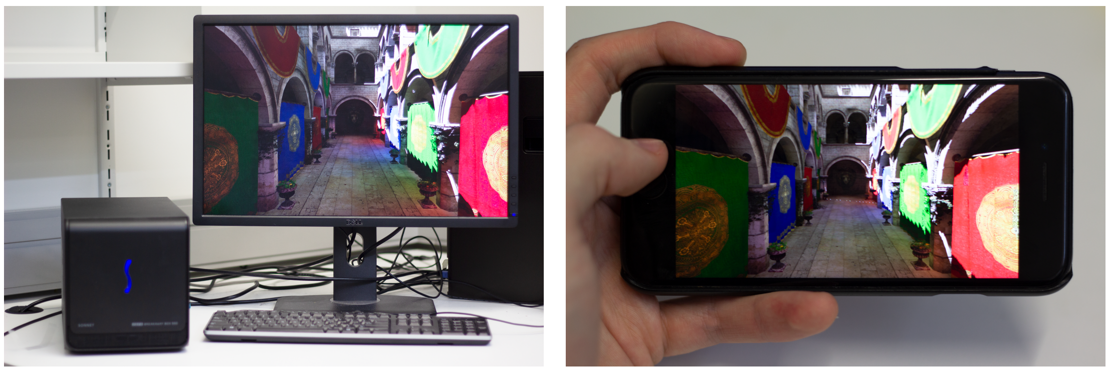


Figure 5.7: AABB Clustering Cut. The dotted red line shows the cut made orthogonal to the longest axis of a scene by the AABB clustering algorithm.

Chapter 6

Results



(a) AMD Vega 56: 50.2 FPS (19.9 ms) **(b)** iPhone 7: 31.5 FPS (31.7 ms)

Figure 6.1: SRP scaling from a dedicated graphics card down to a mobile phone. *Left*: 2017 Macbook Pro with an externally connected AMD RX Vega 56 graphics card. *Right*: Apple iPhone 7.

To determine whether SRP is scalable across a range of hardware, I performed three experiments which tested my implementation of SRP on a low-end, mid-range, and high-end device. The low-end device was an iPhone 7, the mid-range was a 2017 Macbook Pro with a Radeon 560 graphics card, and the high-end device was the same 2017 Macbook Pro with an AMD RX Vega 56 graphics card externally connected via a Sonnet eGFX Breakaway Box 550 (see Table 6.1 for more details). Four test scenes of varying complexity were used for the experiments. The complexity of a scene in this thesis is based on triangle count.

Section 6.1 provides details on the four test scenes. As a preview of the results, Figure 6.1 shows my SRP implementation running the most complex test scene used in this thesis *Crytek Sponza* in real time and scaling from a Macbook with a high-end graphics card down to an iPhone 7.

The first experiment was a visual quality (section 6.2) and performance evaluation (section 6.3). The aim of the visual quality evaluation was to show how my SRP implementation visually compared to a path-traced lightmap reference and to direct-only lighting. The aim of the performance evaluation was to show how my SRP implementation performed in terms of frame time over the four test scenes across the three test devices.

The second experiment was to see whether using higher order spherical harmonics is worth the extra computation cost on low-end devices (section 6.4). To do this, I compared an order-7 reference image of a scene to the same scene rendered with lower order spherical harmonics. As well as comparing the entire image, I also compared specific regions within the image to see if error decreased at different rates for high and low frequency areas.

The last experiment focuses on scalability (section 6.5). It shows how frame time changes as spherical harmonic order, probe count, and receiver count is increased. The aim of this experiment was to see if and how changing these parameters could help improve the performance of SRP and allow more complex scenes to scale down and run in real time on the iPhone 7.

	Device	**Graphics Card**	**Graphics Memory**	**32-Bit FLOPS**
Low-End	iPhone 7	Apple A10 Chip	2 GB Shared	250 GFLOPS*
Mid-Range	Macbook Pro 2017	AMD Radeon 560	4 GB Dedicated	2.611 TFLOPS
High-End	Macbook Pro 2017	AMD Radeon RX Vega 56	8 GB Dedicated	10.57 TFLOPS

Table 6.1: The three test devices. *Apple does not provide GFLOP specifications for its chips but it is estimated to be around 250 GFLOPS [58].

6.1 Test Scenes

For the visual quality and performance evaluations, two simple scenes and two complex scenes were used (Table 6.2). Complexity is based on triangle count. The two simple scenes were *Cornell Box*, a scene often used as a benchmark for global illumination algorithms [59]; and *Blue-Orange Box*, a scene I created which is similar to Cornell Box but longer in length and with the boxes closer to scene surfaces to show indirect shadowing. The complex test scenes were Modern Hall [60] and Crytek Sponza [61].

The precomputation for all the scenes was performed using 4096 sample rays per receiver, 1024 sample rays per probe, and 32 PCA vectors per cluster for CPCA compression. The probes were manually placed near scene surfaces, and all scenes were rendered at a resolution of 1334×750 on all devices. This resolution was chosen as it is the maximum possible on the iPhone 7; however, note that the cost of the algorithm depends mainly on the lightmap resolution, not the screen resolution, as the main overhead during the lighting pass is a single filtered texture sample of the runtime-generated lightmap.

6.2 Visual Quality Evaluation

To perform the visual quality evaluation, my SRP implementation is compared to a ground-truth reference and to direct-only lighting. As the receivers in my implementation are lightmap texels, path-traced lightmaps were used as the ground-truth reference. Path-traced lightmaps are accurate but significantly more expensive than SRP, taking seconds or even minutes to produce renders without noise. They were generated using a progressive lightmapper developed by Thomas Roughton for their Master's thesis [53]. All of the path-traced lightmap reference images were rendered on the AMD RX Vega 56 graphics card.

Figures 6.2 and 6.3 show rendered views of the four test scenes. These figures compare SRP order-7 spherical harmonics with the path-traced reference as well as direct-only lighting renders. The direct-only renders show the significant difference indirect lighting makes. All of the scenes rendered with SRP are visually comparable to the path-traced lightmap reference renders but there are some differences. To make these differences clearer, Figures 6.4 and 6.5 show the same renders with only the contribution of the indirect lighting and without multiplying the indirect lighting by the albedo in the forward pass.

Name	**Triangle Count**	**Probe Count**	**Receiver Count**
Cornell Box	94	27	40 564
Blue-Orange Box	96	20	50 784
Modern Hall	28862	108	103 605
Crytek Sponza	262267	160	437 557

Table 6.2: Test scenes used for the visual and performance evaluations.

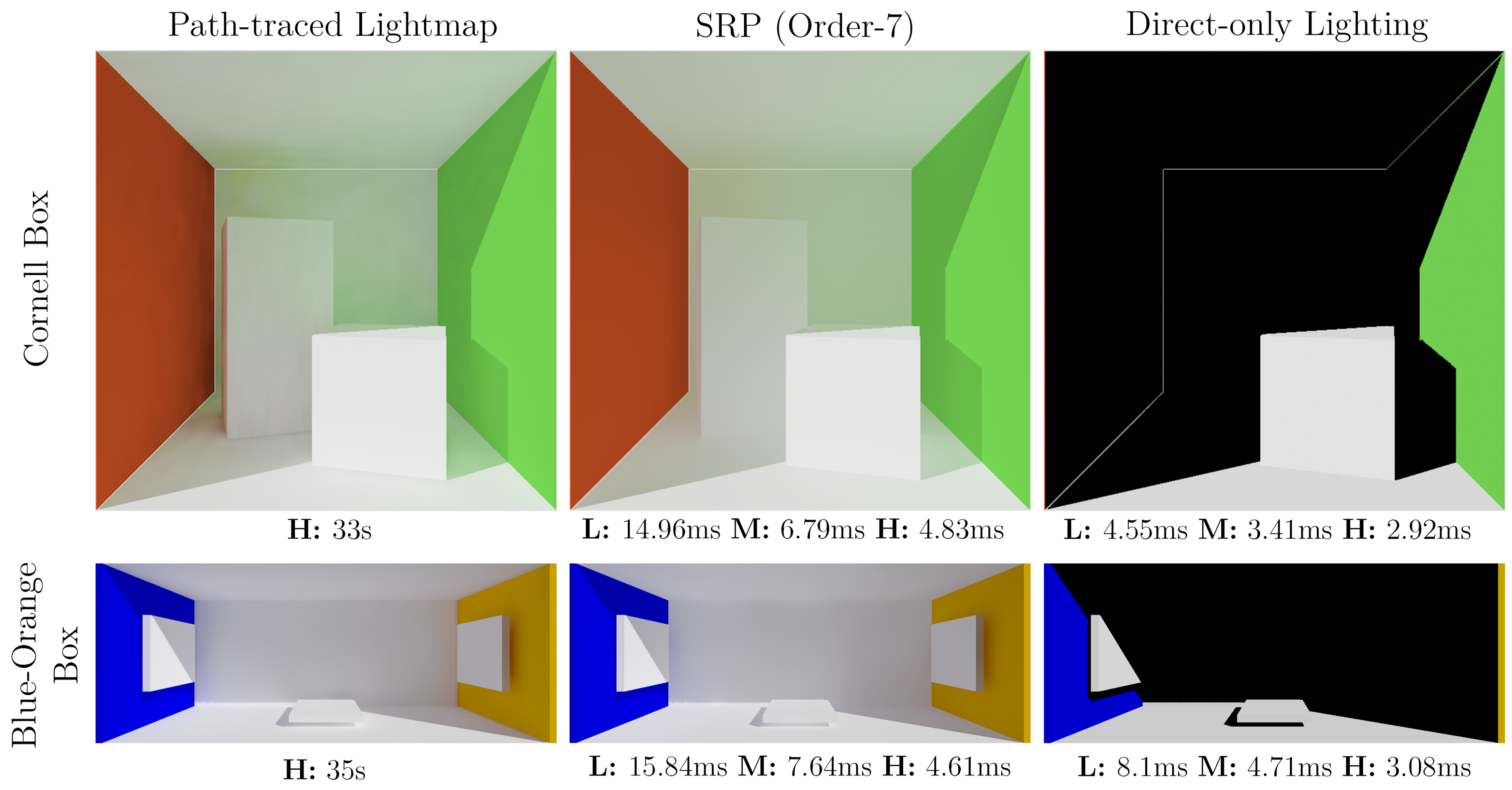

Path-traced Lightmap
SRP (Order-7)
Direct-only Lighting
Cornell Box
Blue-Orange Box
H: 33s
L: 14.96ms M: 6.79ms H: 4.83ms
L: 4.55ms M: 3.41ms H: 2.92ms
H: 35s
L: 15.84ms M: 7.64ms H: 4.61ms
L: 8.1ms M: 4.71ms H: 3.08ms

Figure 6.2: Visual comparison between path-traced lightmaps, SRP, and direct lighting for the two simple test scenes. Each image has its render time beneath it. The path-traced lightmaps have a single render time measured on the AMD RX Vega 56 graphics card. SRP and direct-only lighting have a time for the low-end iPhone 7 (L), the mid-range Radeon 560 (M), and the high-end Vega 56 (H).

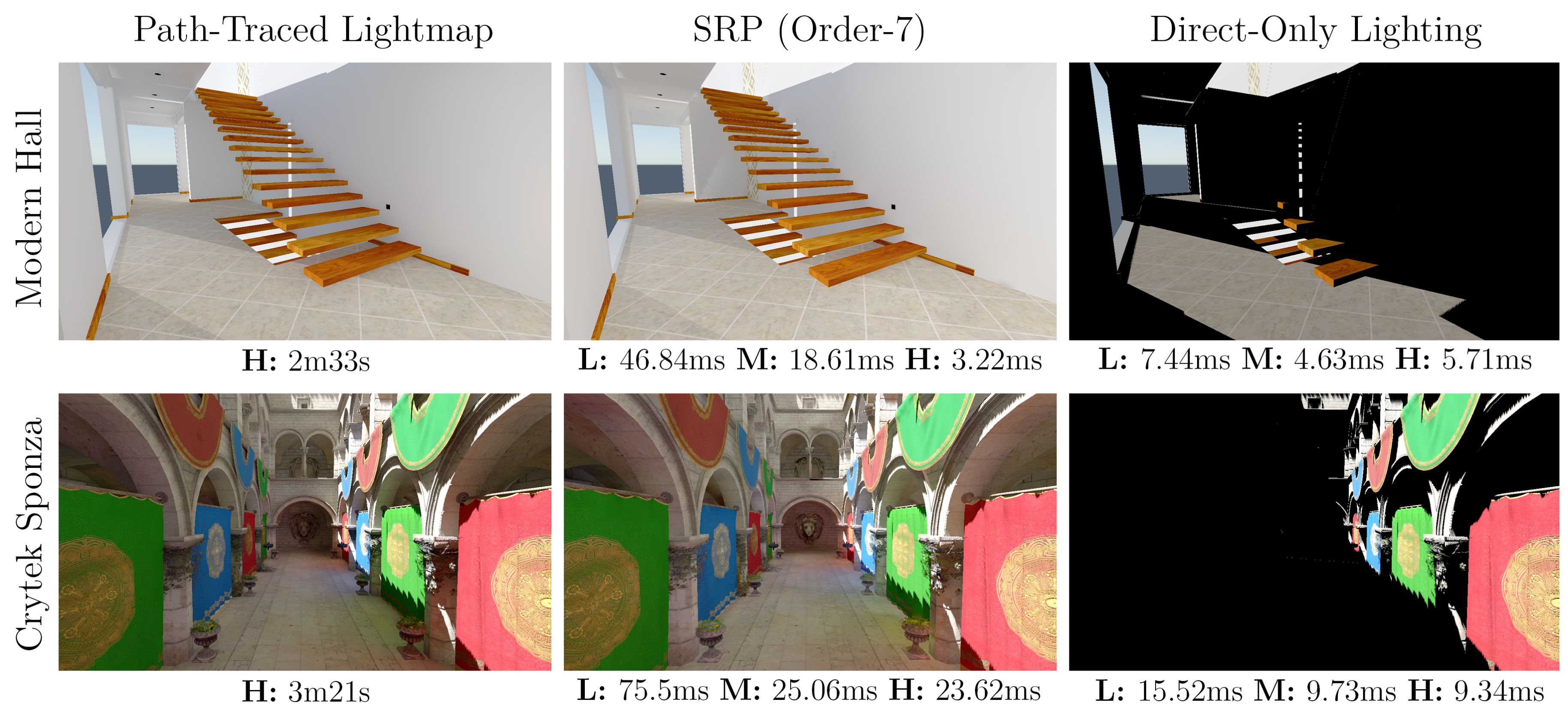

Path-Traced Lightmap
SRP (Order-7)
Direct-Only Lighting
Modern Hall
Crytek Sponza
H: 2m33s
L: 46.84ms M: 18.61ms H: 3.22ms
L: 7.44ms M: 4.63ms H: 5.71ms
H: 3m21s
L: 75.5ms M: 25.06ms H: 23.62ms
L: 15.52ms M: 9.73ms H: 9.34ms

Figure 6.3: Visual comparison between path-traced lightmaps, SRP, and direct lighting for the two complex test scenes. Each image has its render time beneath it. The path-traced lightmaps have a single render time measured on the AMD RX Vega 56 graphics card. SRP and direct-only lighting have a time for the low-end iPhone 7 (L), the mid-range Radeon 560 (M), and the high-end Vega 56 (H).

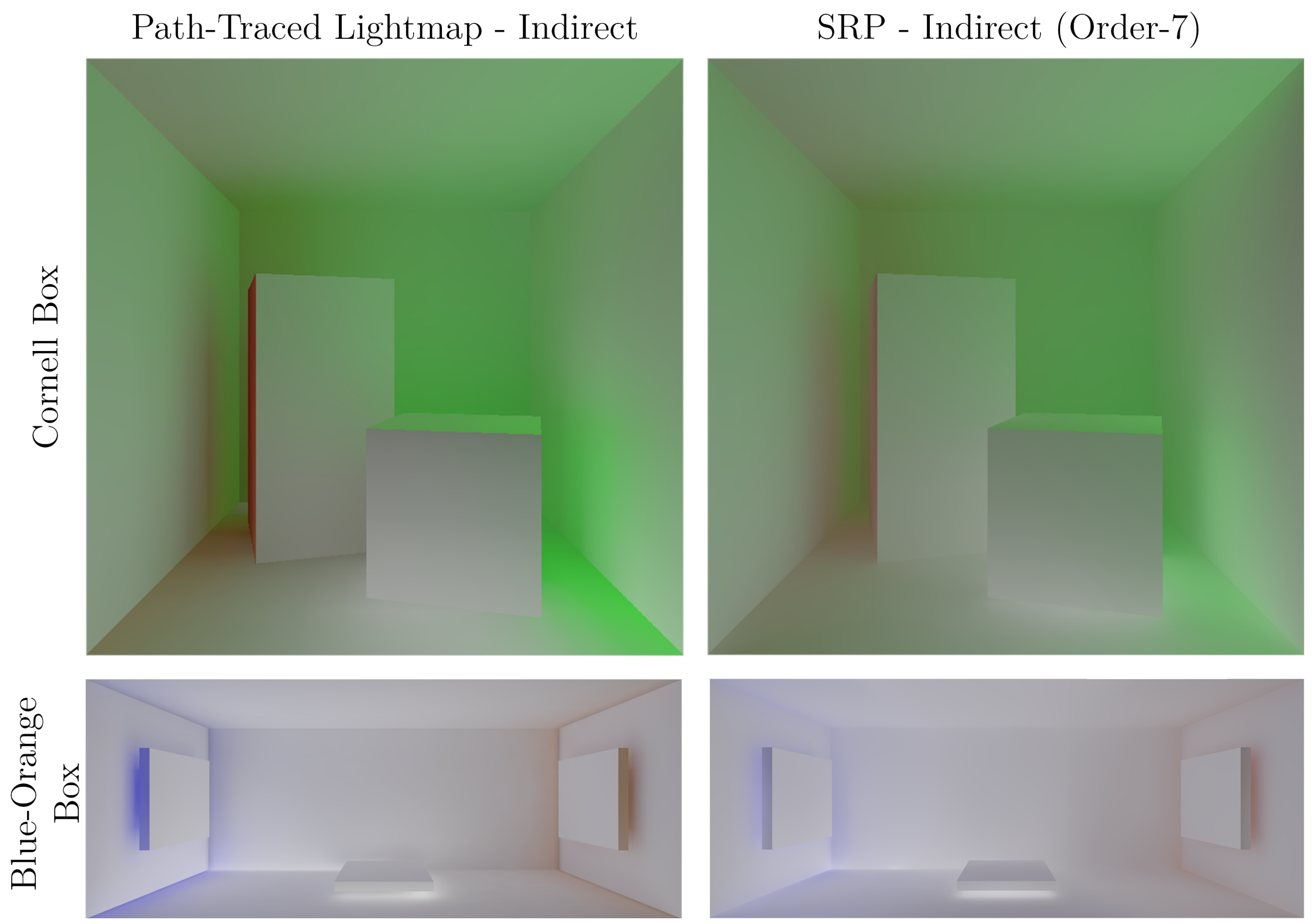


Figure 6.4: Indirect-only comparison between path-traced lightmaps and SRP for the two simple test scenes. Multiplication by the surface albedo is omitted in order to more clearly show the contribution of the indirect lighting.

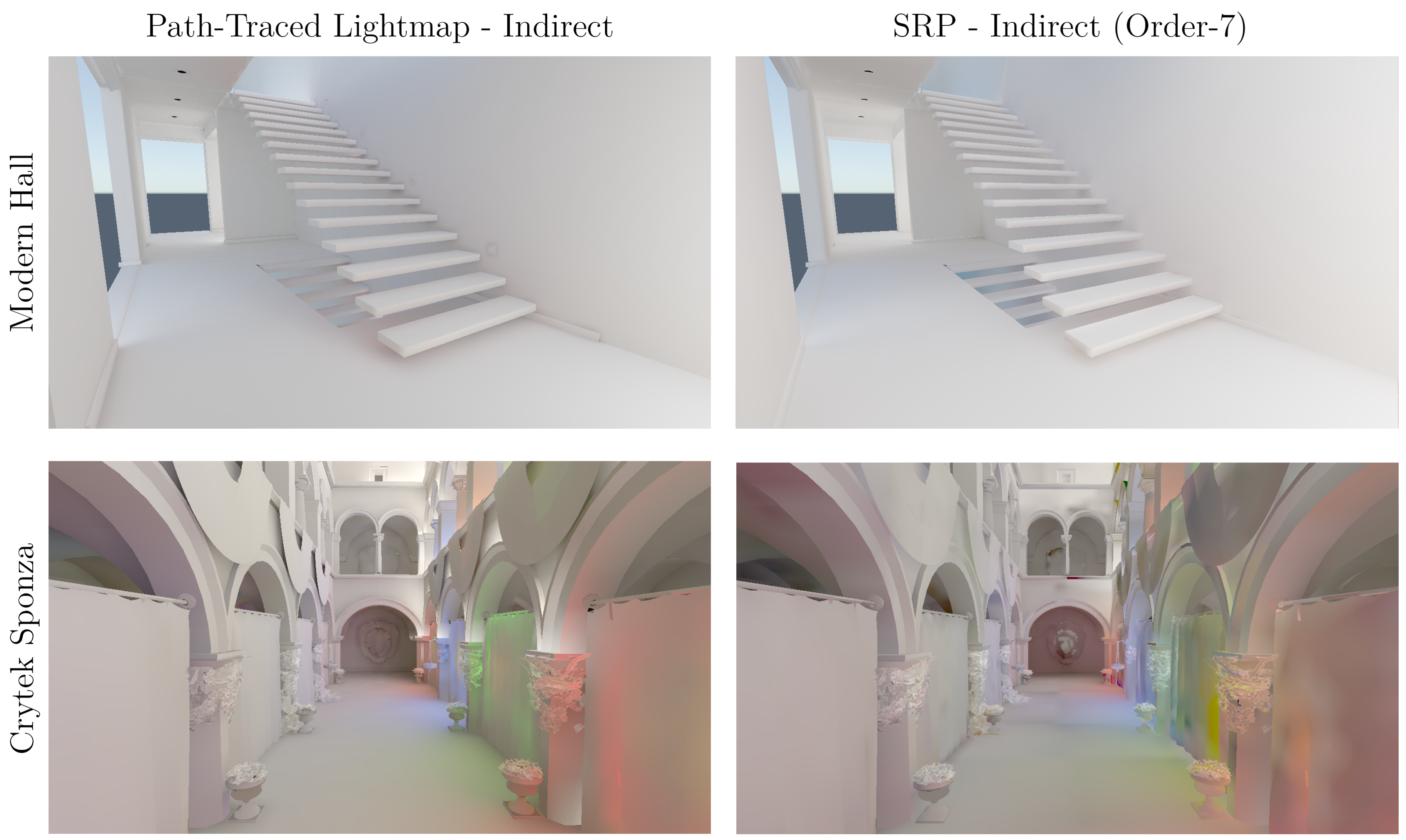


Figure 6.5: Indirect-only comparison between path-traced lightmaps and SRP for the two complex test scenes. Multiplication by the surface albedo is omitted in order to more clearly show the contribution of the indirect lighting.

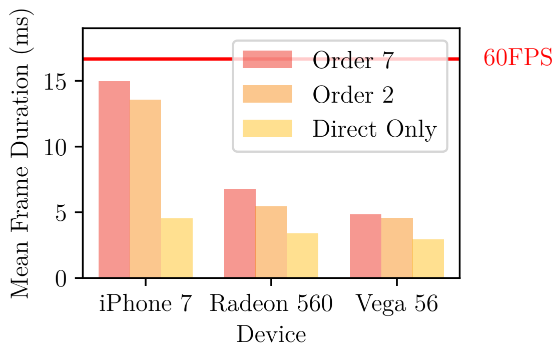
6.3 Performance Evaluation

For the performance evaluation, the mean render time per frame was recorded over 1000 frames. To show the performance difference between low and high-order spherical harmonics, the frame time of SRP was recorded for both order-7 and order-2 spherical harmonics. Additionally, the frame time of direct lighting was recorded to show the baseline cost of rendering a scene in LlamaEngine.

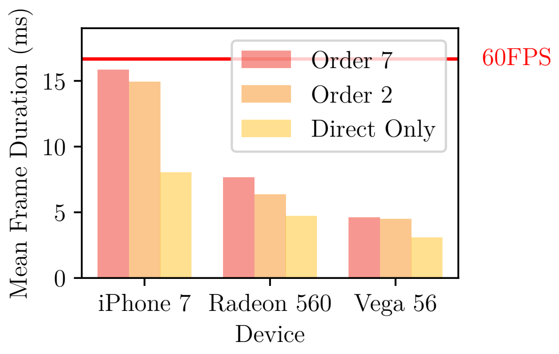
For the two simple test scenes, SRP ran in real time at a minimum of 60 FPS across all three test devices for both order-7 and order-2 spherical harmonics (Figure 6.6a-6.6b). SRP took more than twice as long to render on the iPhone 7 than on the two dedicated graphics cards. The performance gap between the iPhone 7 and the dedicated graphics cards is also larger for SRP than for direct lighting only. For example, for the Cornell Box scene, the difference between SRP order-2 spherical harmonics and direct lighting on the Radeon 560 was 2 ms yet on the iPhone 7 it was 9 ms.

The two complex test scenes ran in real time for the dedicated graphics cards. However, only Modern Hall ran in real time on the iPhone 7 at 33.67 FPS (29.7 ms per frame) for order-2 spherical harmonics. It ran slightly below real-time for order-7 spherical harmonics at 21.34 FPS (46.84 ms per frame) (Figure 6.6c). Crytek Sponza ran at an interactive frame rates of 16.36 FPS (61.1 ms per frame) for order-2 spherical harmonics and 13.24 FPS (75.5 ms per frame) for order-7 (Figure 6.6d).

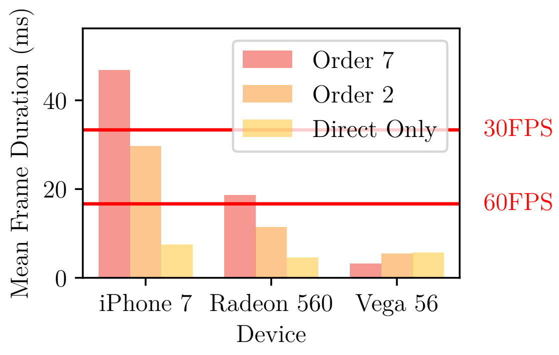
While these results show the complex test scenes do not run in real time on the iPhone 7, the scalability results in section 6.5 show how the performance of SRP can be scaled by reducing the receiver and probe count.



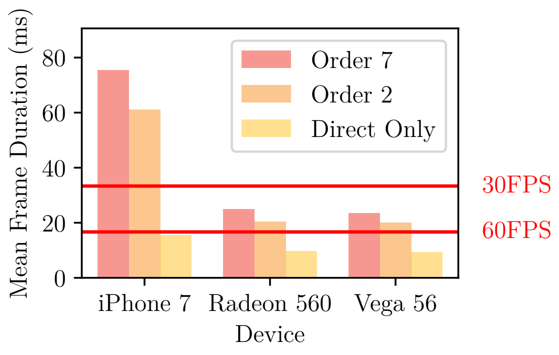
(a) Cornell Box



(b) Blue-Orange Box



(c) Modern Hall



(d) Crytek Sponza

Figure 6.6: Mean frame times for the four test scenes across the three test devices. Durations are shown for SRP with order-7 and order-2 spherical harmonics as well as for direct-only lighting. Direct-only lighting is included to show how much extra frame time is used by SRP.

6.4 Error vs Spherical Harmonic Order

The second experiment was to see whether using higher order spherical harmonics is worth the extra computation cost on low-end devices. I compared an order-7 indirect only render of a test scene (Figure 6.7 top left) to the same scene rendered with lower order spherical harmonics (Figure 6.7 top right). The comparison was done using the mean squared error (MSE) of the lightness channel (L) of the renders in the CIE L*a*b colour space [62]. I chose to perform the comparison in the CIE L*a*b colour space as the space is based on human perception of light and therefore gives a more intuitive error metric than a mean squared error in the RGB (red, green, blue) colour space.

To see if the error changed at different rates in different areas of the render, I also performed this comparison in four regions (Figure 6.7 top left). Regions a) and b) were placed over areas with higher frequency changes. The edge of the object by the left wall and the corner of the room. Regions c) and d) were placed over areas with low frequency changes. The back wall and the floor.

For the entire image, the MSE in lightness decreased as spherical harmonic order increased (grey line in Figure 6.7 bottom right). For the regions, the lightness MSE decreased faster for the low frequency regions than for the higher frequency regions as spherical harmonic order was increased. The low frequency regions dropped to a MSE below 0.01 by order-1, but it took the high frequency regions until order-4 to do the same. This is shown in both the bottom left and bottom right sub-figures within Figure 6.7. The bottom left figure shows the original render and a greyscale difference image between the order-7 render and each lower order render of the L channel in the L*a*b colour space.

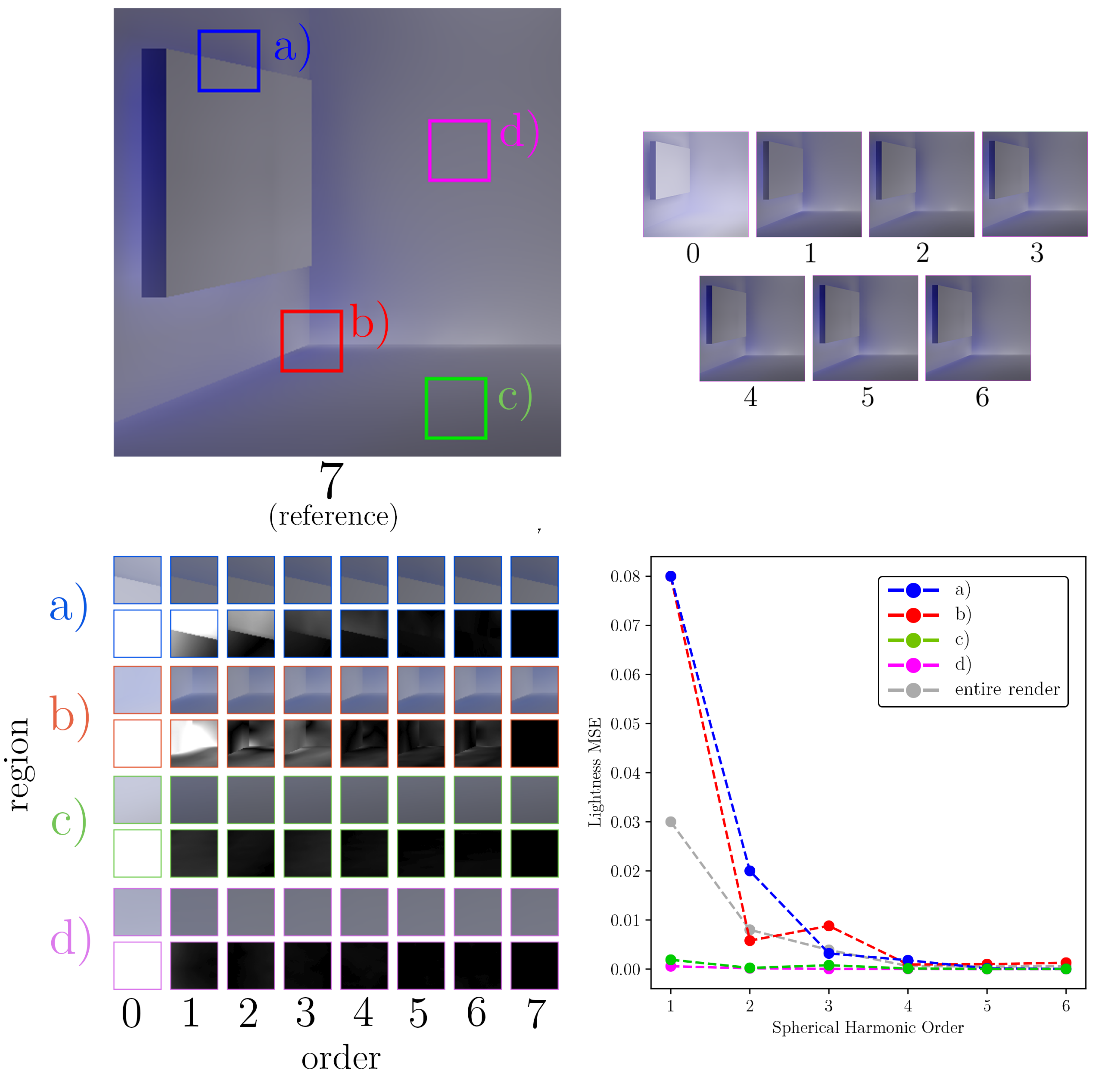


Figure 6.7: *Top Left:* Indirect-only SRP renders of the test scene using order-0 through to order-6 spherical harmonics. *Top Right:* Order-7 reference render which the other renders are compared to. The comparison regions are shown with coloured squares and labels. *Bottom Left:* Comparison regions as spherical harmonic order increases. The top row of each region is the original render. Bottom row is the difference in MSE from the reference render displayed as a greyscale render, where 0 is black and 0.01 is white. *Bottom Right:* Graph of the MSE in lightness for each region as spherical harmonic order increases. The graph omits order-0 in order to avoid skewing the Y-axis scale.

6.5 Scalability

SRP has scaling parameters which can be adjusted in the precomputation process to reduce the amount of computation required at runtime. These include spherical harmonic order, receiver count, and probe count [1]. To see how each of these parameters affected frame time as they increase, one parameter at a time was increased in increments and the mean frame time over 1000 frames was recorded for each increment on the Blue-Orange Box test scene.

6.5.1 Spherical Harmonic Order

Spherical harmonic order was increased from order-0 up to order-7. The probe count was fixed at 20 and receiver count at 50 784.

Interestingly, the increase in frame time was higher for the dedicated graphics cards. The difference between order-7 and order-2 spherical harmonics was 1.7 ms for the Radeon 560 and 2.3 ms for the Vega 56. Whereas on the iPhone 7 it was 1.1 ms.

6.5.2 Probe Count

Probe count was increased from 20 to 200. Spherical harmonic order was fixed at 2 and receiver count at 50 784.

The number of probes in a scene had a significantly larger impact on the iPhone 7. The difference in frame time between 200 probes and 20 probes for the iPhone 7 was 49.3 ms, 6.7 ms for the Radeon 560, and 2.25 ms for the Vega 56. For a receiver count of 50 784 with order-2 probes, around 100 probes was the limit for staying under the real-time cut-off of 33.3 ms for the iPhone 7.

6.5.3 Receiver Count

Receiver count was increased in increments from 50 784 - 1 000 000 receivers. Spherical harmonic order was fixed at 2 and probe count at 20.

[1]The number of PCA vectors for CPCA also has an effect but is not included in these results due to a lack of time. It is fixed at 32 PCA vectors for all scenes.

As with the probe count, increasing the receiver count had a significantly larger impact on the iPhone 7. The frame time difference between 1 000 000 and 50 784 receivers was 150.1 ms on the iPhone 7, 11.7 ms on the Radeon 560, and 1.9 ms on the Vega 56.

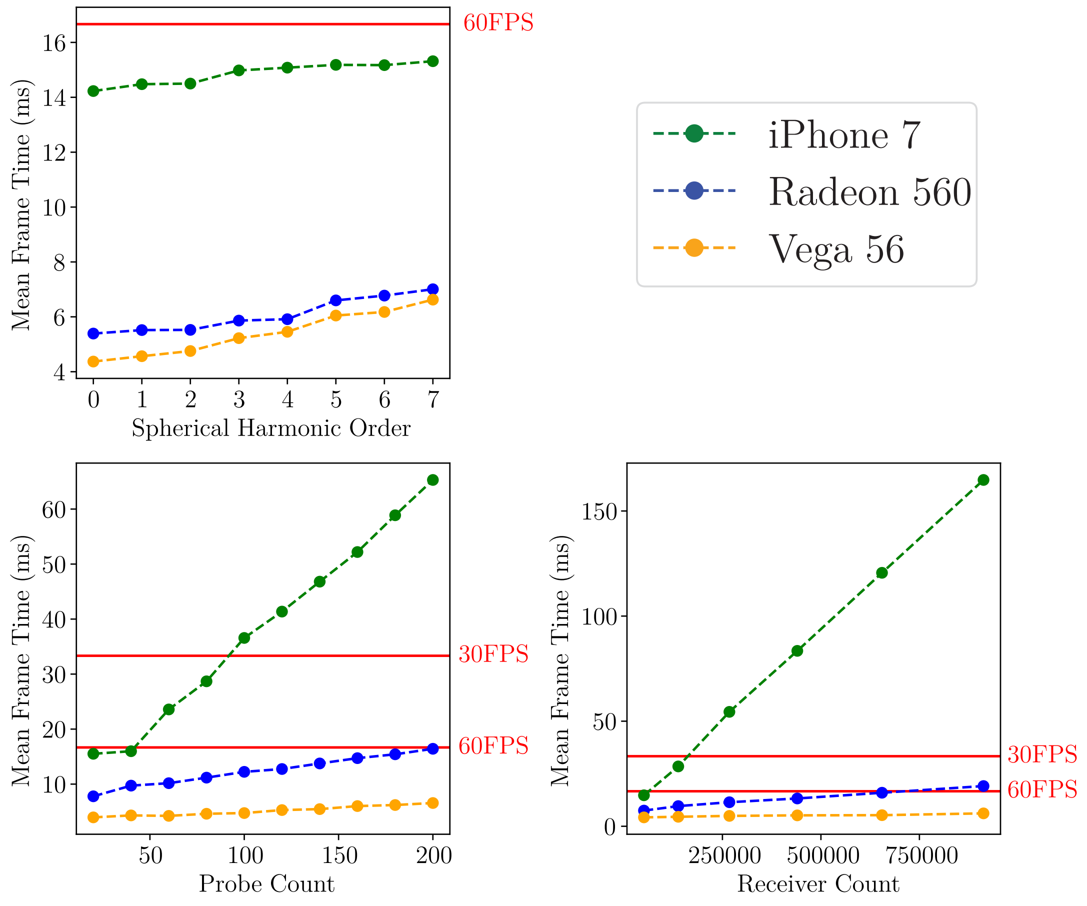


Figure 6.8: Graphs showing how the increase in spherical harmonic order, probe count, and receiver count affect frame time across the three test devices.

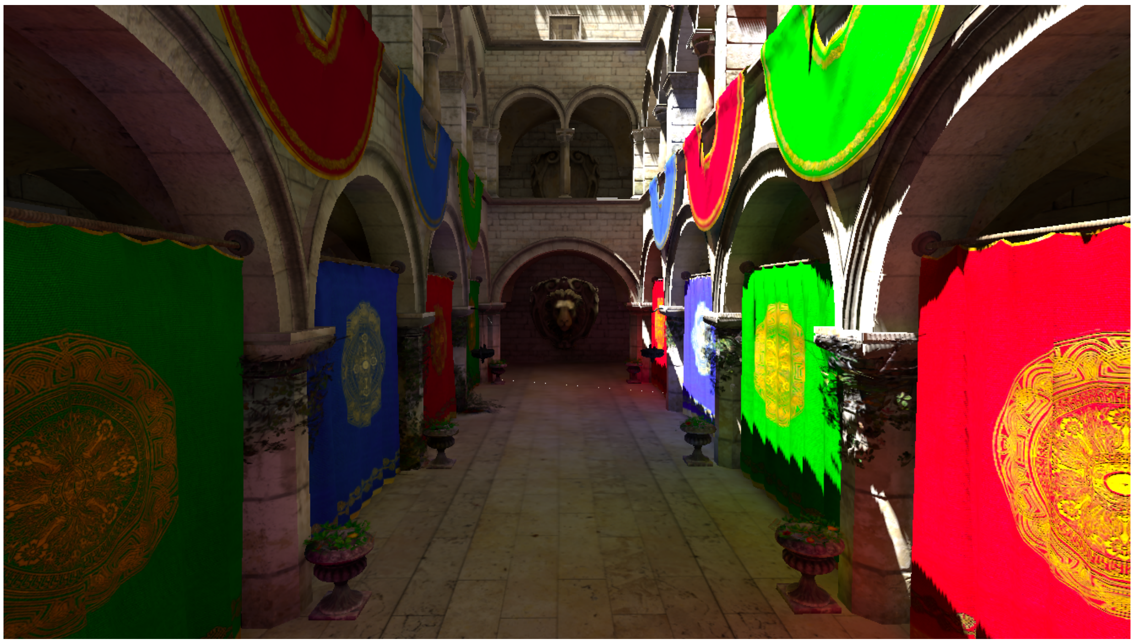
6.6 Reducing Receiver Counts to Improve Performance

The results from the previous section show that performance improves if any of the spherical harmonic order, probe count, or receiver count are reduced. To see if Crytek Sponza could run in real time on the iPhone 7 while still producing acceptable results, the receiver count was decreased in increments from 437 557 to 56 581 receivers. All of the renders in this section were performed on the iPhone 7 using order-2 spherical harmonics and 169 probes.

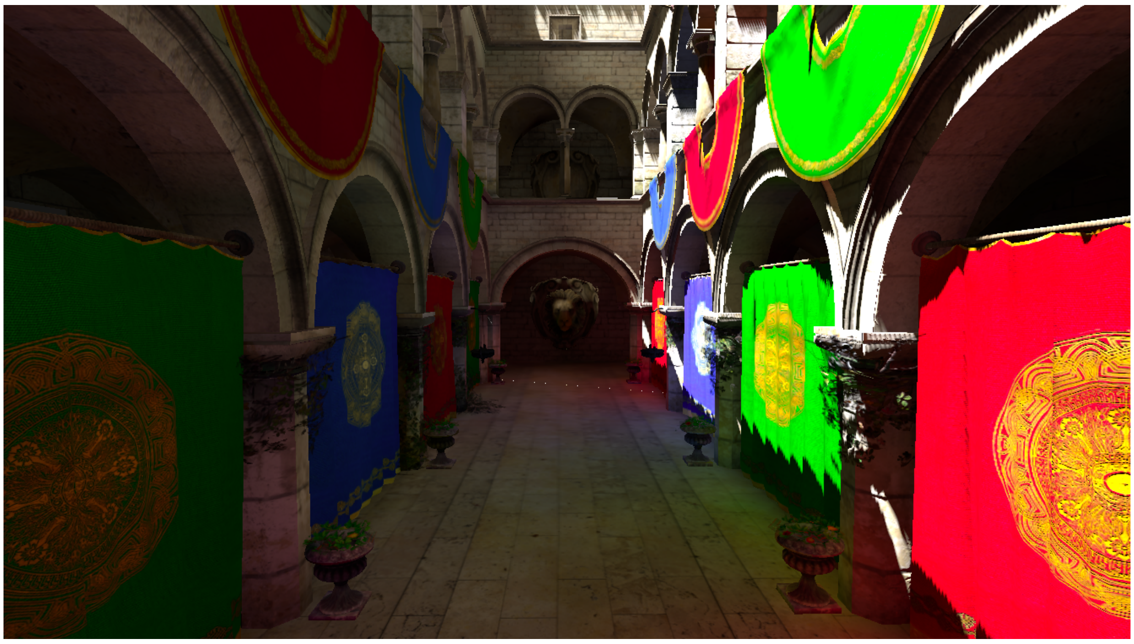
Crytek Sponza ran in real time at 31.5 FPS (31.7 ms per frame) when the receiver count was reduced to 120 479, but there are some quality differences when compared to the render with 437 557 receivers (Figure 6.9). Notably, the indirect lighting is slightly darker, the indirect shadows have less definition, and the indirect colour bleeding from the cloth curtains extends over a slightly larger area.

The quality differences are clearer when only indirect lighting is shown without surface albedo. Figure 6.10 show four renders of Crytek Sponza with only indirect lighting and without surface albedo. The renders progressively decrease in receiver count. At 56 581 receivers (Figure 6.10d), my SRP implementation ran even faster at 36.1 FPS (27.7 ms per frame), but the render is considerably darker, the indirect shadows are not visible, and the colour bleeding extends much further.

To a lesser extent, the same quality differences appear when reducing the receiver count of a path-traced lightmap reference render. Figure 6.11 shows a path-traced lightmap reference render of Crytek Sponza with 437 557 (6.11a) and 56 581 receivers (6.11b). The renders show only indirect lighting without surface albedo, and no linear filter is applied to clearly show the size of the receivers. At 56 581 receivers, the path-traced lightmap render also becomes darker, indirect shadows are no longer visible, and the indirect colour bleeding extends out slightly more. However, these quality differences are much more pronounced in same render using SRP (Figure 6.12b). Additionally, the path-traced lightmap reference render with 437 557 receivers has much more detail than SRP with 437 557 receivers. For example, the yellow seal on the curtains is visible in the path-traced reference (Figure 6.11a), but not in the SRP version (Figure 6.12a).

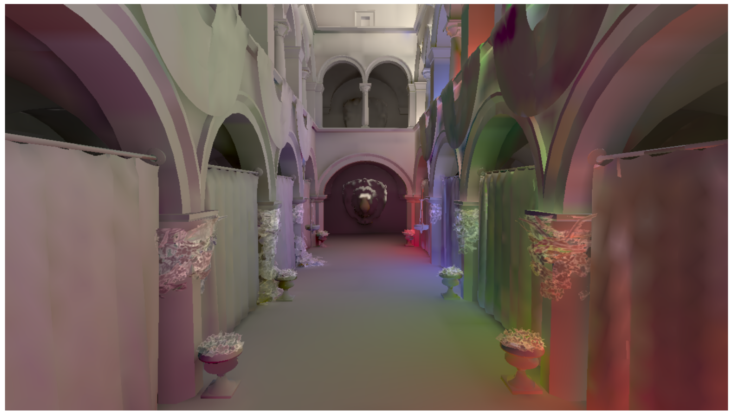


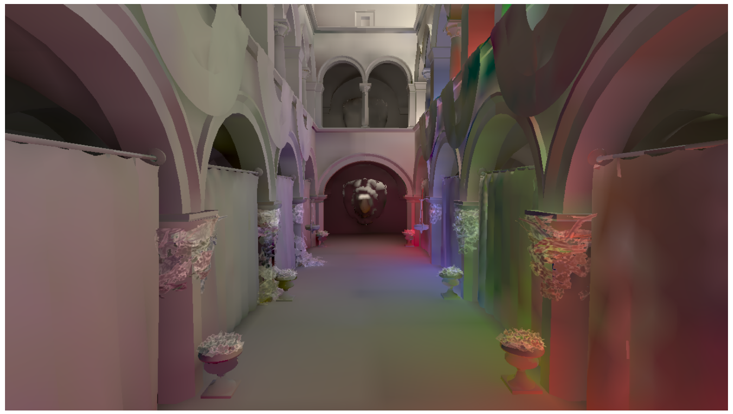
(a) 437 557 receivers — 16.4 FPS (61.1 ms)

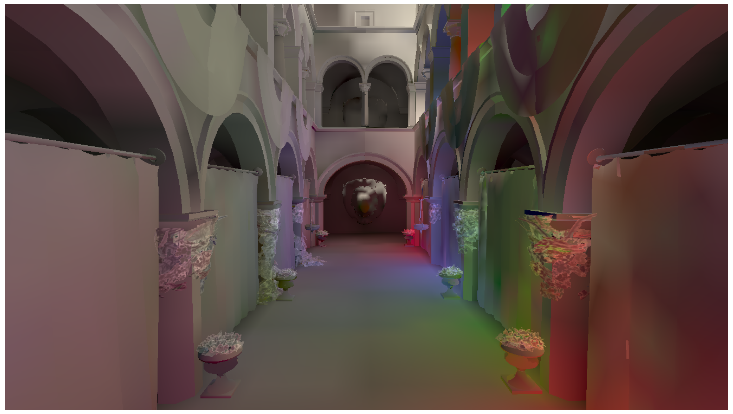


(b) 199 406 receivers — 31.5 FPS (31.7 ms)

Figure 6.9: Decreasing receiver count on Crytek Sponza from 437 557 to 120 479 receivers allowed it to run in real-time on the iPhone 7. There is a drop in quality though. The indirect lighting for SRP is darker, colour bleeds out more from the cloth curtains, and indirect shadows are less defined.


(a) 437 557 receivers — 16.4 FPS (61.1 ms)


(b) 199 406 receivers — 24.4 FPS (40.9 ms)


(c) 120 479 receivers — 31.5 FPS (31.7 ms)

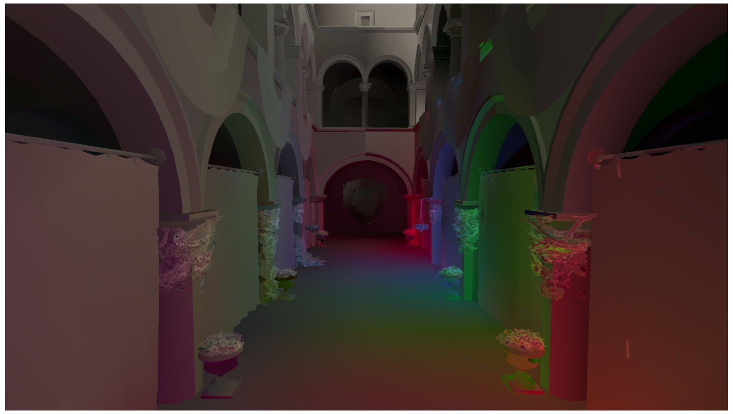
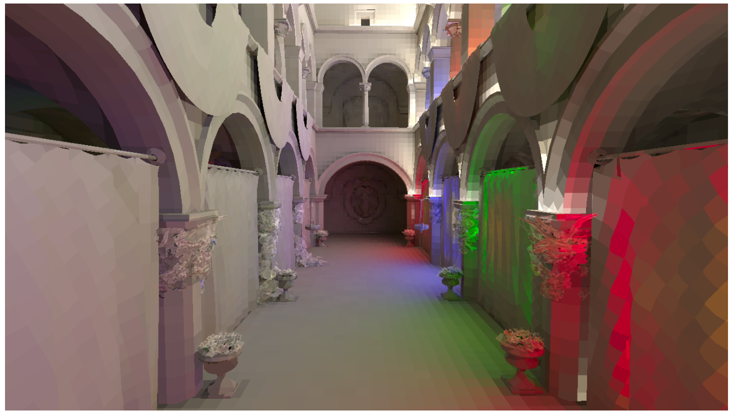

(d) 56 581 receivers — 36.1 FPS (27.7 ms)

Figure 6.10: Decreasing receiver count on Crytek Sponza. Renders only show contribution of indirect light without surface albedo. Performance metrics and renders are from the iPhone 7.


(a) 437 557 receivers

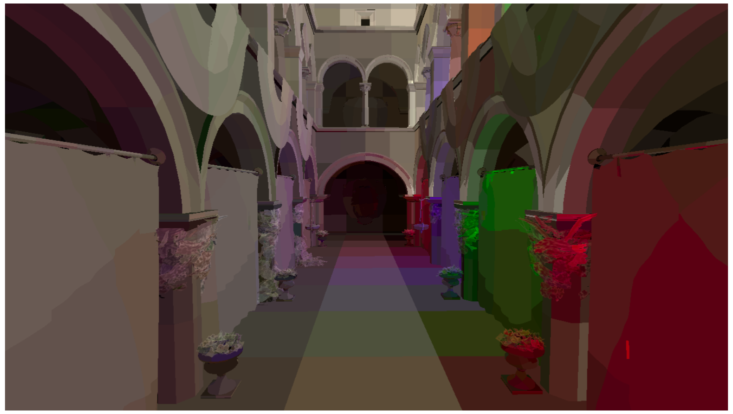
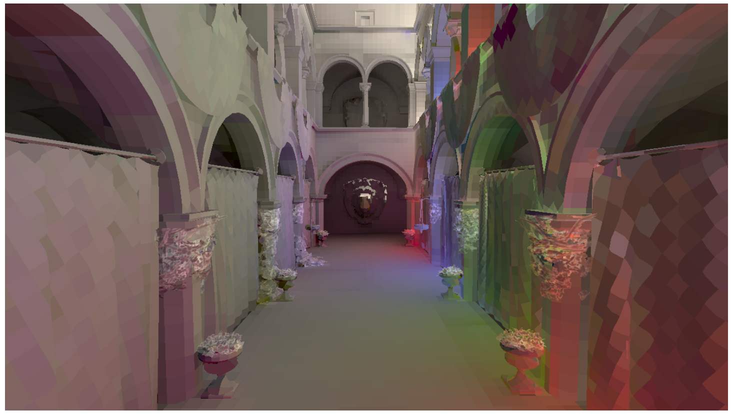

(b) 56 581 receivers

Figure 6.11: Reducing receiver count on Crytek Sponza for the path-traced lightmap reference.


(a) 437 557 receivers.

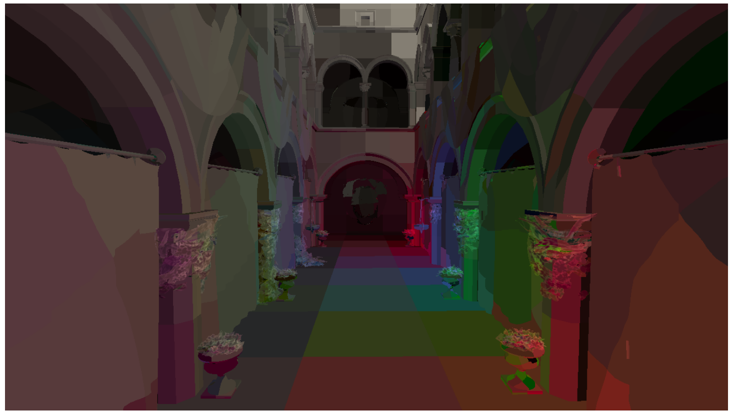

(b) 56 581 receivers.

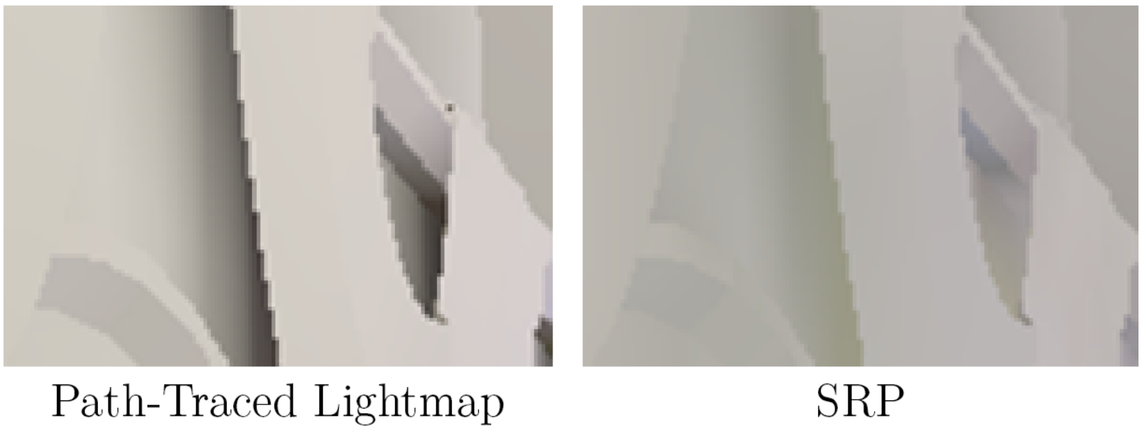
Figure 6.12: Reducing receiver count on Crytek Sponza for SRP.

6.7 Analysis

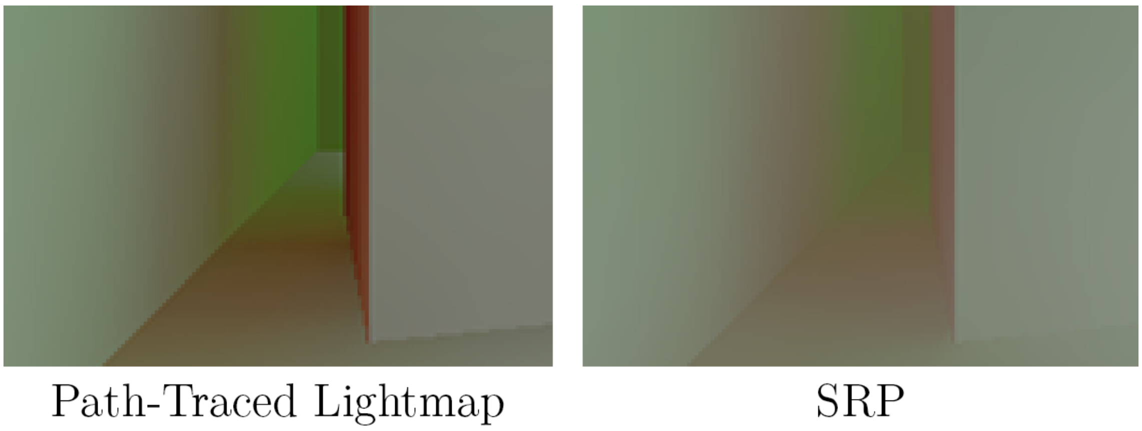
The performance evaluation results (section 6.3) showed that, at least for the simple scenes, it is possible for SRP to run in real time on an iPhone 7. The two simple test scenes ran on all devices at $60\,\mathrm{FPS}$ (Figures 6.6a and 6.6b), but for the two complex test scenes, SRP only ran in real time on the iPhone 7 for Modern Hall with order-2 spherical harmonics. For Crytek Sponza, SRP still ran interactively on the iPhone 7 at $16.4\,\mathrm{FPS}$ for order-2 spherical harmonics, but it did not run above the $30\,\mathrm{FPS}$ real-time threshold.

The visual evaluation results (section 6.2) show that my SRP implementation is comparable to a path-traced lightmap reference image. Although there are differences between them, SRP is considerably faster. For Crytek Sponza, the path-traced image took 3 minutes and 12 seconds to render compared to $75.5\,\mathrm{ms}$ on the iPhone 7. The main differences between SRP and the path-traced lightmaps were less defined indirect shadows and colour bleeding (Figure 6.13). This lack of definition is due to the band-limiting of the probes, the number of probes, and the probe positions. It is not because of the limited resolution of the receivers as the path-traced lightmap reference renders use the same receiver points as the SRP renders yet still have more definition.

The error vs order results (section 6.4) indicate that high-order spherical harmonics may not be worth the extra computation cost on low-end devices. For the low-frequency regions, the error is significantly reduced by order-2 spherical harmonics. This is expected as spherical harmonics can capture Lambertian diffuse irradiance signals with 1% error [10]. However, higher order spherical harmonics are still needed to better capture higher frequency information like visibility. For the Blue-Orange test scene with one directional light, the reduction in lightness MSE between order-3 and order-7 spherical harmonics was minimal for both the low-frequency and high-frequency regions, and not worth the $0.82\,\mathrm{ms}$ increased cost on the iPhone 7 between order-2 and order-7 harmonics. However, as the experiment only tested one scene, the results can only indicate and do not confirm that high-order spherical harmonics are not worth the extra computation. There may be some scenes where high-order spherical harmonics have a greater impact.



(a) Indirect shadows are visible but much less defined behind the banners in the Crytek Sponza test scene rendered using SRP (cut-out from Figure 6.5).



(b) Both the indirect shadows and colour bleeding from the left wall onto the box are less defined in the Cornell Box test scene rendered using SRP (cut-out from Figure 6.4).

Figure 6.13: Cut-outs showing examples of the less defined indirect shadows and colour bleeding in SRP.

The scalability results (section 6.5) show that the receiver count has a large impact on performance for the iPhone 7. Reducing the receiver count from 437 557 to 120 479 receivers for Crytek Sponza brought the per frame time down to 31.7 ms (section 6.6) – just under the real-time cut-off of 33.33 ms. The render with 120 479 receivers was still comparable to the render with 437 557 receivers, although there were slight reductions in quality: the render was slightly darker, indirect shadows were a little less defined, and the coloured indirect lighting from the curtains bled across the ground more. Once the receiver count was reduced to 56 581, however, the render was no longer comparable. It was significantly darker, indirect shadows were barely visible, and much more colour bleeding was apparent.

The darkening of the renders is mostly due to the receiver resolution. If half a receiver is lit by a light, then the whole receiver will be half the intensity of the lit half as each receiver stores the average irradiance of the patch it covers. This makes the half lit by the light appear too dark and the half in shadow appear too bright. With fewer receivers in a scene, the area each receiver covers is larger and so it is more likely that a receiver will cover an area which differs in intensity. A slightly darker image was expected when rendering Crytek Sponza with fewer receivers because some receivers were positioned underneath the curtains. These receivers were both in light and shadow at the same time. Figure 6.11b shows a path-traced lightmap reference render of Crytek Sponza with 56 581 receivers; as expected, the receivers underneath the curtains are considerably darker. However, the same render with SRP in Figure 6.12b is even darker still and there is considerably more colour bleeding over the receivers in the middle of the courtyard. These differences are likely due to the band-limited probes and their placement, but without further investigation, it is hard to tell if the differences are inherent to SRP or due to an implementation error.

Either way, the results show that receiver count is a point of scalability which can be reduced to improve performance at the cost of quality, and the renders are still visually comparable as long as the receiver count is not reduced too much. In this case, reducing the receiver count allowed Crytek Sponza to run in real time on the low-end iPhone 7.

Chapter 7

Conclusion

The research in this thesis presents a scalable real-time global illumination algorithm using SRP, and to the best of my knowledge, this thesis is the first to show that SRP runs in real time on mobile devices. The results showed that for simple scenes, my implementation ran above 60 FPS on an iPhone 7, and for complex test scenes, it ran above 30 FPS. The probe count, receiver count, and number of basis functions used are all points of scalability which can be adjusted to improve performance at the cost of quality. While a reduction in quality was necessary for the complex scenes to run in real time on the iPhone 7, the reduced-quality renders were still comparable to the high-quality ones.

However, while my SRP implementation runs in real time and scales from a low-end iPhone 7 to a high-end Vega 56, it is not yet fast enough for applications where graphics techniques need to run alongside other computation on the GPU. Graphics techniques in video games need to run well below the real-time cut-off of 33.33 ms. C. Barré-BriseBois, a rendering engineer at EA Games, stated that techniques in their games must typically run under a cut-off of 3.0 ms if a game needs to run at 30 FPS, or under 1.5 ms if a game needs to run at 60 FPS [63]. Even on the simple test scenes, my SRP implementation does not meet these requirements as it took at least 5 ms of GPU time.

I believe that this is a limitation of my implementation rather than SRP itself. Comparing the performance results from Silvennoinen and Lehtinen's implementation to mine, their most complex test scene *Brutalist Hall* took only

$4.87\,\mathrm{ms}$ of GPU time on a Nvidia Titan X, while the most complex test scene in this thesis took $14.26\,\mathrm{ms}$ on the AMD Vega 56 graphics card. This is a significant difference, especially given the specifications of the Vega 56 are generally better than the Titan X. On top of this, Brutalist Hall is more complex than Crytek Sponza. The triangle count for Brutalist Hall is not published, but it uses 115 more probes and has $129\,813$ more receivers. With more time, effort, and expertise spent on optimisation, I believe the performance cost of my SRP implementation could be considerably reduced even for low-end devices like the iPhone 7.

There are two main areas where future work could improve the performance of my implementation. Firstly, the compute shaders my implementation uses to perform the matrix multiplications needed to reconstruct irradiance at the receivers use a naïve approach (section 5.3). Each compute thread for both multiplications only multiplies a single row of the matrix; there is an overhead associated with spawning threads, so reducing the number of threads by increasing the number of rows each thread multiplies may improve performance. Additionally, as the cluster SVD projection multiplication is sparse, it would be worth investigating efficient algorithms available for performing sparse matrix multiplication on the GPU [64, 65]. Secondly, my implementation projects radiance at all probes in a scene into spherical harmonics every frame. It would be interesting to see if staggering probe updates across frames could reduce costs without a perceivable drop in quality; for example by updating half of the probes one frame, then the other half on the next, or by updating probes closer to the camera more often.

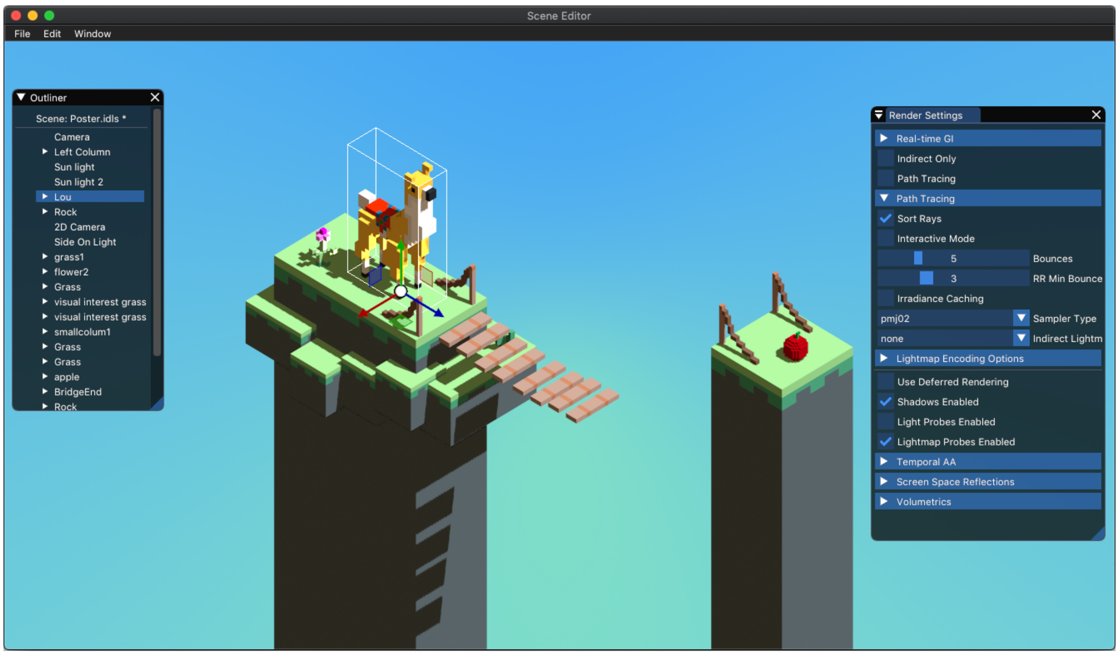
There are a few other areas which would be worth exploring in future work. My SRP implementation was limited to irradiance transport. Silvennoinen and Lehtinen support radiance transport in their implementation which allows for normal mapping and glossy BRDFs for the final bounce towards the camera. They had to reduce spherical harmonics to order-4 to maintain performance. It would be interesting to see if low-end devices like the iPhone 7 could handle radiance transfer in real time. Next, my thesis did not explore any automatic probe placement methods. Probes were placed manually for all results. This is not ideal as probe placement affects the quality of renders. Exploration of automatic probe placement methods to help with the scalability of SRP would also be worthwhile for future work. Fewer probes placed in better positions could provide higher quality results at a lower performance cost.

Real-time global illumination that scales from low to high-end devices means applications that require realistic lighting can reach wider audiences by running on more devices. This thesis shows that SRP scales from a high-end AMD RX Vega 56 graphics card down to an iPhone 7 while still being visually comparable to a path-traced lightmap reference. Additionally, an optimised implementation would run even faster making it viable for applications that need to run other computation alongside SRP.

Appendices

Appendix A

LlamaEngine



Indirect Only
Path Tracing
✓ Sort Rays
Interactive Mode
5 Bounces
3 RR Min Bounce
Irradiance Caching
pmj02 Sampler Type
none Indirect Lightm
Use Deferred Rendering
✓ Shadows Enabled
Light Probes Enabled
✓ Lightmap Probes Enabled

Figure A.1: Lou, the protagonist of Interdimensional Llama, within LlamaEngine.

LlamaEngine, named because of its planned use in the video game Interdimensional Llama (Figure A.1), is a 3D rasteriser and rendering framework developed by Thomas Roughton and myself. We began development on LlamaEngine in late 2016 and have continually developed and improved on it since then. It primarily uses the Swift programming language [66] on the CPU and the Metal Shading Language [16] on the GPU.

LlamaEngine supports both forward+ and deferred rendering using clustered shading [67], uses a rendering graph approach [68] for setting up rendering work, and uses industry-standard material models: either diffuse Lambertian, or Disney diffuse and GGX specular with Smith height-correlated visibility [69].

It also has a GPU-based path-tracer and progressive lightmapper developed by Thomas Roughton for their Master's thesis [53], which I use in this thesis as a ground-truth reference to compare the quality of my SRP implementation against.

Bibliography

[1] M. Mittring, "Finding Next Gen: CryEngine 2," in *ACM SIGGRAPH 2007 Courses*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007, pp. 97–121, event-place: San Diego, California. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281671

[2] T. Ritschel, T. Grosch, and H.-P. Seidel, "Approximating Dynamic Global Illumination in Image Space," in *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ser. I3D '09. New York, NY, USA: ACM, 2009, pp. 75–82, event-place: Boston, Massachusetts. [Online]. Available: http://doi.acm.org/10.1145/1507149.1507161

[3] Crassin Cyril, Neyret Fabrice, Sainz Miguel, Green Simon, and Eisemann Elmar, "Interactive Indirect Illumination Using Voxel Cone Tracing," *Computer Graphics Forum*, vol. 30, no. 7, pp. 1921–1930, Nov. 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2011.02063.x

[4] C. Wahlén, "Global Illumination in Real-Time using Voxel Cone Tracing on Mobile Devices," Master's thesis, Linköping University, 2016. [Online]. Available: https://liu.diva-portal.org/smash/get/diva2:1148572/FULLTEXT01.pdf

[5] Abrash, Michael, *Michael Abrash's Graphics Programming Black Book (Special Edition)*. Coriolis Group, Jul. 1997.

[6] A. Silvennoinen and J. Lehtinen, "Real-time Global Illumination by Precomputed Local Reconstruction from Sparse Radiance Probes," *ACM*

Trans. Graph., vol. 36, no. 6, pp. 230:1–230:13, Nov. 2017. [Online]. Available: http://doi.acm.org/10.1145/3130800.3130852

[7] P. Johnston, "Cornell Box With and Without Radiosity Enabled.gif." [Online]. Available: https://commons.wikimedia.org/wiki/File:Cornell_Box_With_and_Without_Radiosity_Enabled.gif

[8] J. T. Kajiya, "The Rendering Equation," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 143–150. [Online]. Available: http://doi.acm.org/10.1145/15922.15902

[9] M. Iwanicki and P.-P. Sloan, *Ambient Dice*. The Eurographics Association, 2017. [Online]. Available: https://diglib.eg.org:443/xmlui/handle/10.2312/sre20171191

[10] R. Ramamoorthi and P. Hanrahan, "An Efficient Representation for Irradiance Environment Maps," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 497–500. [Online]. Available: http://doi.acm.org/10.1145/383259.383317

[11] R. Green, "Spherical Harmonic Lighting: The Gritty Details," p. 69, Jan. 2003.

[12] R. Ng, R. Ramamoorthi, and P. Hanrahan, "All-frequency Shadows Using Non-linear Wavelet Lighting Approximation," in *ACM SIGGRAPH 2003 Papers*, ser. SIGGRAPH '03. New York, NY, USA: ACM, 2003, pp. 376–381. [Online]. Available: http://doi.acm.org/10.1145/1201775.882280

[13] Y.-T. Tsai and Z.-C. Shih, "All-Frequency Precomputed Radiance Transfer using Spherical Radial Basis Functions and Clustered Tensor Approximation," p. 10.

[14] Iñigo Quilez, "Spherical Harmonics." [Online]. Available: https://commons.wikimedia.org/wiki/File:Spherical_Harmonics.png

[15] M. Pharr and G. Humphreys, *Physically Based Rendering - 3rd Edition. From Theory to Implementation*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[16] Apple Inc, "Metal Shading Language Specification Version 2.1," p. 179, 2019. [Online]. Available: https://developer.apple.com/metal/Metal-Shading-Language-Specification.pdf

[17] P.-P. Sloan, J. Hall, J. Hart, and J. Snyder, "Clustered Principal Components for Precomputed Radiance Transfer," in *ACM SIGGRAPH 2003 Papers*, ser. SIGGRAPH '03. New York, NY, USA: ACM, 2003, pp. 382–391. [Online]. Available: http://doi.acm.org/10.1145/1201775.882281

[18] T. Ritschel, C. Dachsbacher, T. Grosch, and J. Kautz, "The State of the Art in Interactive Global Illumination," *Computer Graphics Forum*, vol. 31, no. 1, pp. 160–188, Feb. 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.02093.x

[19] A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. Christensen, J. Hanika, C. Eisenacher, and G. Nichols, "The path tracing revolution in the movie industry," in *ACM SIGGRAPH 2015 Courses on - SIGGRAPH '15*. Los Angeles, California: ACM Press, 2015, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2776880.2792699

[20] L. Fascione, J. Hanika, M. Leone, M. Droske, J. Schwarzhaupt, T. Davidovič, A. Weidlich, and J. Meng, "Manuka: A Batch-Shading Architecture for Spectral Path Tracing in Movie Production," *ACM Trans. Graph.*, vol. 37, no. 3, pp. 31:1–31:18, Aug. 2018. [Online]. Available: http://doi.acm.org/10.1145/3182161

[21] P. Christensen, A. Kensler, and C. Kilpatrick, "Progressive Multi-Jittered Sample Sequences: Supplemental Materials," p. 7, 2018.

[22] B. Burley, D. Adler, M. J.-Y. Chiang, H. Driskill, R. Habel, P. Kelly, P. Kutz, Y. K. Li, and D. Teece, "The Design and Evolution of Disney's Hyperion Renderer," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 3, p. 33,

Aug. 2018. [Online]. Available: http://dl.acm.org/citation.cfm?id=3243123.3182159

[23] H. W. Jensen, "Global Illumination using Photon Maps," in *Rendering Techniques '96*, W. Hansmann, W. T. Hewitt, W. Purgathofer, X. Pueyo, and P. Schröder, Eds. Vienna: Springer Vienna, 1996, pp. 21–30. [Online]. Available: http://www.springerlink.com/index/10.1007/978-3-7091-7484-5_3

[24] E. Veach and L. J. Guibas, "Metropolis light transport," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*. Not Known: ACM Press, 1997, pp. 65–76. [Online]. Available: http://portal.acm.org/citation.cfm?doid=258734.258775

[25] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, and P. Shirley, "State of the Art in Ray Tracing Animated Scenes," *Computer Graphics Forum*, vol. 28, no. 6, pp. 1691–1722, Sep. 2009. [Online]. Available: http://doi.wiley.com/10.1111/j.1467-8659.2008.01313.x

[26] G. Estes, "World's Top Graphics Software Companies Are Already Adopting NVIDIA RTX Capabilities. Here's Why." Aug. 2018. [Online]. Available: https://blogs.nvidia.com/blog/2018/08/13/turing-industry-support/

[27] A. Burnes, "Battlefield V DXR Real-Time Ray Tracing Available Now," Nov. 2018. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/battlefield-v-rtx-ray-tracing-out-now/

[28] S. Kim, T. Harada, and Y. J. Kim, "Energy-efficient global illumination algorithms for mobile devices using dynamic voltage and frequency scaling," *Computers & Graphics*, vol. 70, pp. 198–205, Feb. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0097849317301103

[29] W. Lee, Y. Shin, J. Lee, J. Kim, J. Nah, H. Park, S. Jung, and S. Lee, "A novel mobile GPU architecture based on ray tracing," in *2013 IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2013, pp. 21–22.

[30] O. Good and Z. Taylor, “Optimized photon tracing using spherical harmonic light maps,” in *ACM SIGGRAPH 2005 Sketches on - SIGGRAPH ’05*. Los Angeles, California: ACM Press, 2005, p. 53. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1187112.1187175

[31] H. Chen and X. Liu, “Lighting and Material of Halo 3,” in *ACM SIGGRAPH 2008 Games*, ser. SIGGRAPH ’08. New York, NY, USA: ACM, 2008, pp. 1–22, event-place: Los Angeles, California. [Online]. Available: http://doi.acm.org/10.1145/1404435.1404437

[32] Jonathan Blow, “Graphics Tech: Precomputed Lighting,” Mar. 2010. [Online]. Available: http://the-witness.net/news/2010/03/graphics-tech-precomputed-lighting/

[33] Ignacio Castaño, “Lightmap Parameterization,” Mar. 2010. [Online]. Available: http://the-witness.net/news/2010/03/graphics-tech-texture-parameterization/

[34] A. Keller, “Instant radiosity,” in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’97*. Not Known: ACM Press, 1997, pp. 49–56. [Online]. Available: http://portal.acm.org/citation.cfm?doid=258734.258769

[35] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg, “Lightcuts: A Scalable Approach to Illumination,” p. 10, 2005.

[36] B. Walter, P. Khungurn, and K. Bala, “Bidirectional Lightcuts,” *ACM Trans. Graph.*, vol. 31, no. 4, pp. 59:1–59:11, Jul. 2012. [Online]. Available: http://doi.acm.org/10.1145/2185520.2185555

[37] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz, “Imperfect Shadow Maps for Efficient Computation of Indirect Illumination,” in *ACM SIGGRAPH Asia 2008 Papers*, ser. SIGGRAPH Asia ’08. New York, NY, USA: ACM, 2008, pp. 129:1–129:8, event-place: Singapore. [Online]. Available: http://doi.acm.org/10.1145/1457515.1409082

[38] T. Ritschel, E. Eisemann, I. Ha, J. D. K. Kim, and H.-P. Seidel, "Making Imperfect Shadow Maps View-Adaptive: High-Quality Global Illumination in Large Dynamic Scenes," *Computer Graphics Forum*, vol. 30, no. 8, pp. 2258–2269, Dec. 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-8659.2011.01998.x

[39] C. Dachsbacher, J. Křivánek, M. Hašan, A. Arbree, B. Walter, and J. Novák, "Scalable Realistic Rendering with Many-Light Methods: Scalable Realistic Rendering with Many-Light Methods," *Computer Graphics Forum*, vol. 33, no. 1, pp. 88–104, Feb. 2014. [Online]. Available: http://doi.wiley.com/10.1111/cgf.12256

[40] J. McLaren, "The Technology of the Tomorrow Children," GDC 2015, 2015. [Online]. Available: https://web.archive.org/web/20150315020546/http://fumufumu.q-games.com/archives/TheTechnologyOfTomorrowsChildrenFinal.pdf

[41] A. Yudintsev, "Scalable Real-Time Global Illumination for Large Scenes," Mar. 2019. [Online]. Available: https://www.gdcvault.com/play/1026469/Scalable-Real-Time-Global-Illumination

[42] M. McGuire, M. Mara, D. Nowrouzezahrai, and D. Luebke, "Real-time Global Illumination Using Precomputed Light Field Probes," in *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ser. I3D '17. New York, NY, USA: ACM, 2017, pp. 2:1–2:11. [Online]. Available: http://doi.acm.org/10.1145/3023368.3023378

[43] S. Martin, "A Real Time Radiosity Architecture for Video Games. Advances in Real-Time Rendering Course." SIGGRAPH 2010, 2010. [Online]. Available: http://advances.realtimerendering.com/s2010/Martin-Einarsson-RadiosityArchitecture(SIGGRAPH%202010%20Advanced%20RealTime%20Rendering%20Course).pdf

[44] W. Joseph, "Global Illumination That Scales," 2018.

[45] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed Radiance Transfer for Real-time Rendering in Dynamic, Low-frequency Lighting Environments," in

Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, ser. SIGGRAPH '02. New York, NY, USA: ACM, 2002, pp. 527–536. [Online]. Available: http://doi.acm.org/10.1145/566570.566612

[46] J. Kautz, P.-P. Sloan, and J. Snyder, "Fast, Arbitrary BRDF Shading for Low-frequency Lighting Using Spherical Harmonics," in *Proceedings of the 13th Eurographics Workshop on Rendering*, ser. EGRW '02. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2002, pp. 291–296. [Online]. Available: http://dl.acm.org/citation.cfm?id=581896.581934

[47] M. Hasan, F. Pellacini, and K. Bala, "Direct-to-Indirect Transfer for Cinematic Relighting," p. 9, 2006.

[48] G. J. Ward, F. M. Rubinstein, and R. D. Clear, "A Ray Tracing Solution for Diffuse Interreflection," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '88. New York, NY, USA: ACM, 1988, pp. 85–92. [Online]. Available: http://doi.acm.org/10.1145/54852.378490

[49] J. Krivanek, P. Gautron, S. Pattanaik, and K. Bouatouch, "Radiance caching for efficient global illumination computation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 5, pp. 550–561, Sep. 2005.

[50] J. Lehtinen, M. Zwicker, E. Turquin, and J. Kontkanen, "A Meshless Hierarchical Representation for Light Transport," p. 9, 2008.

[51] AMD, "Baikal - Real-time path-tracer." [Online]. Available: https://github.com/GPUOpen-LibrariesAndSDKs/RadeonProRender-Baikal

[52] B. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products: Technometrics: Vol 4, No 3," Aug. 1965. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00401706.1962.10490022

[53] T. Rougthon, "Interactive Generation of Path-Traced Lightmaps (Pending Examination)," Master's thesis, Victoria University of Wellington, 2019.

[54] AMD, "Radeon Rays." [Online]. Available: https://gpuopen.com/gaming-product/radeon-rays/

[55] Apple, "Metal Performance Shaders." [Online]. Available: https://developer.apple.com/documentation/metalperformanceshaders

[56] University of Tennessee, University of California, Berkely, University Of Colorado Denver, and NAG Ltd., "LAPACK." [Online]. Available: http://performance.netlib.org/lapack/

[57] Y. Linde, A. Buzo, and R. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Transactions on Communications*, vol. 28, no. 1, pp. 84–95, Jan. 1980.

[58] GSMArena, "Counterclockwise: console-quality graphics on phones," Nov. 2017. [Online]. Available: https://www.gsmarena.com/counterclockwise_consolequality_graphics_on_phones-news-28080.php

[59] C. University, *Cornell Box*. [Online]. Available: http://www.graphics.cornell.edu/online/box/data.html

[60] B. Bitterli, *Rendering resources.*, 2016. [Online]. Available: https://benedikt-bitterli.me/resources/

[61] M. McGuire, *Computer Graphics Archive*, Jul. 2017. [Online]. Available: https://casual-effects.com/data

[62] International Commision on Illumination, "COLORIMETRY — PART 4: CIE 1976 L*A*B* COLOUR SPACE," 1976. [Online]. Available: http://www.cie.co.at/publications/colorimetry-part-4-cie-1976-lab-colour-space

[63] C. Barré-Brisebois, "A Certain Slant of Light: Past, Present and Future Challenges of Global Illumination in Games," SIGGRAPH 2017, Aug. 2017. [Online]. Available: https://www.slideshare.net/colinbb/past-present-and-future-challenges-of-global-illumination-in-games

[64] M. Deveci, C. Trott, and S. Rajamanickam, "Multithreaded sparse matrix-matrix multiplication for many-core and GPU architectures,"

Parallel Computing, vol. 78, pp. 33–46, Oct. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167819118301923

[65] J. Demouth, "Sparse Matrix-Matrix Multiplication on the GPU," GPU Technology Conference, 2012. [Online]. Available: https://on-demand.gputechconf.com/gtc/2012/presentations/S0285-Optimization-of-Sparse-Matrix-Matrix-Multiplication-on-GPU.pdf

[66] Apple, "Swift." [Online]. Available: https://swift.org

[67] O. Olsson, M. Billeter, and U. Assarsson, "Clustered Deferred and Forward Shading," p. 10, 2012.

[68] E. Arts, "FrameGraph: Extensible Rendering Architecture in Frostbite - Frostbite," Mar. 2017. [Online]. Available: https://www.ea.com/frostbite/news/framegraph-extensible-rendering-architecture-in-frostbite

[69] E. Heitz, "Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs," vol. 3, no. 2, p. 60, 2014.