Evaluation of Function Bases for Spherical Light Signals

Rasmus Rønn Nielsen (jpm687)

Bachelor Thesis

Supervisor: Prof. Jon Sporring

UNIVERSITY OF COPENHAGEN

Department of Computer Science

Januar 11, 2019

Abstract

In video game development, it is generally infeasible to fully simulate the interaction between light and matter. To solve this problem, games usually sample the light environment by precomputing so-called light probes at selected points in space. These probes are represented mathematically as spherical functions often referred to as signals. Computer graphics literature offers several techniques to encode such spherical signals. Most published works focus on just a single encoding technique. This thesis contributes by *comparing* a selection of techniques.

To enable this comparison, an evaluation method is developed and implemented. A selected set of spherical models are then evaluated using this method. This leads to a set of measurement data, which are analysed to gain insight into the benefits and drawbacks of each encoding technique. The result of the analysis is a set of recommendations about which encoding techniques to use for particular types of spherical signals (irradiance, radiance, and self-occlusion).

Additionally, this thesis serves as an introduction to spherical function bases for the uninitiated reader with interest in this subject.

Contents

Chapter 1

Introduction

Rendering a 3D scene is fundamentally about simulating interactions between light and matter. The light that ends up hitting our eyes' retinas (or a camera sensor) determines what we see. The first decade of real-time 3D video games did not do any real light simulation, however. The required computational resources were simply not available. Instead, game developers of the time pushed the available hardware to its limits via clever tricks that somehow created the illusion of 3D worlds.

The phrase *physically based rendering* is often used to refer to a family of rendering techniques based on the principles of physics. This idea was popularized by Matt Pharr, Greg Humphreys, and Pat Hanrahan in 2004 with the release of their seminal book of the same name [PHW10]. Among the adopters was the video game industry. This new trend accelerated the race toward photorealism in games as show-cased in figure 1.1. Today, full simulation of photons is still not possible in real-time. However, many of the most dominant visual phenomena of light can now be approximated in real-time: Fresnel-based reflection, translucency/transparency, conservation of energy, metallicity, etc. Some games even simulate real-time global illumination, e.g. by using libraries like Enlighten [Sil].

Physically based rendering is an extensive subject and beyond the scope of this thesis. For a full treatment see [PHW10] and [AMHH18, Chapter 9].

In order to present the problem statement of this thesis, I must first provide some context.

(a) Wolfenstein 3D (1992), Copyright id Software.

(b) Red Dead Redemption 2 (2018), Copyright Rockstar Games.

Figure 1.1: Evolution of 3D video game graphics.

It is necessary to cover the basic quantities of radiometry (section 1.1) since these will play a big role later. I will also introduce the concept of light probes (section 1.2) and explain why they are an important part of modern video games. Equipped with this background information, I will complete this introduction by presenting the problem statement (section 1.3).

1.1 Radiometric Quantities

An important part of migrating to physically based rendering models is adopting the units and quantities conventionally used within the field of physics. *Radiometry* is a branch of physics that deals with the objective measurement of electromagnetic radiation, including human-visible light (this is opposed to *photometry* that deals with the subjective human perception of radiation). A full coverage of radiometry is beyond the scope of this thesis. However, I will briefly introduce a few of the fundamental radiometric quantities since these will be used extensively throughout the rest of this thesis.

Probably the most important quantity is *radiance*. Informally, it describes the power carried by a ray of light. It is defined as the radiant flux emitted or received by a given surface, per unit solid angle, per unit projected area. To aid in the understanding of this definition, figure 1.2 shows the geometry of radiance. One way to get intuition about this quantity is to imagine a device that measures the number of photons per second coming from a small cone $\mathrm{d}\omega$ of directions centred around $\boldsymbol{\omega}$. The device's sensor is a small area $\mathrm{d}A^{\perp}$ that is oriented perpendicularly to $\boldsymbol{\omega}$. The device outputs the number of detected photons divided by the solid angle of the cone, divided by the size of the sensor area. As the sensor area and cone get smaller this value tends towards a certain number, and in the limit you get radiance.

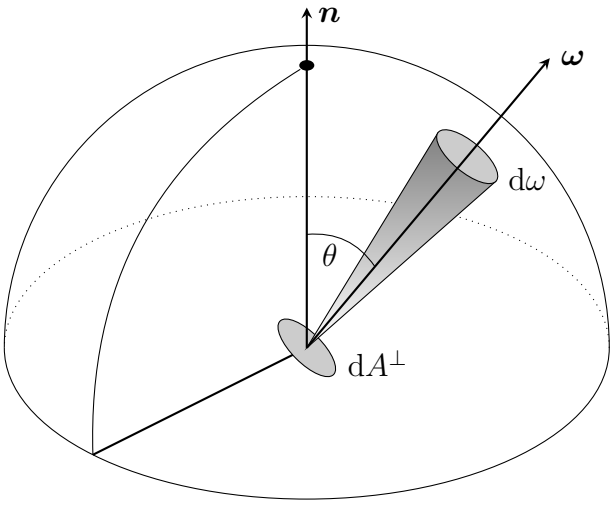


Figure 1.2: The geometry of radiance. Radiance is the amount of infinitesimal flux, per infinitesimal solid angle $\mathrm{d}\omega$, per infinitesimal projected area $\mathrm{d}A^{\perp}$. θ is the angle between a light ray and the surface normal $\boldsymbol{n}$.

The other important quantity is *irradiance* which is defined as radiant flux per unit area. Irradiance E at a point with surface normal $\boldsymbol{n}$ can be expressed in terms of incoming

radiance $L(\boldsymbol{\omega})$ as

$$E(\boldsymbol{n}) = \int_{\Omega(\boldsymbol{n})} L(\boldsymbol{\omega}) \cos\theta \, \mathrm{d}\omega \tag{1.1}$$

where $\Omega(\boldsymbol{n})$ is the hemisphere of directions above $\boldsymbol{n}$, $\mathrm{d}\omega$ is infinitesimal solid angle, and θ is the angle between ω and $\boldsymbol{n}$ (see figure 1.2). Intuitively this expression is just summing over all incoming light. The cosine term is there to account for the fact that light coming from a shallow angle is spread over more area.

1.2 Light Probes

In order to render a 3D object, we need a description of the surrounding light environment. Due to the intricate nature of light, this is not trivial to compute. Techniques such as ray tracing solve the problem by tracing paths of light as they bounce around the scene. Depending on the scene's surface properties, this may be too computationally expensive to do in real-time, even with modern techniques and hardware.

Instead of recalculating the light on demand for each pixel, we can instead sample the light environment at selected positions in the scene and store the result for later use. This sampling is often performed offline, i.e. before the game is running. Such samples are referred to as *light probes*. Each light probe is thus a spherical function that describes incoming light for all directions, for a particular point in the scene. We can approximate the light information at an arbitrary point and direction by interpolating between close-by light probes [Cup12]. Figure 1.3 shows an example of light probes placed in a 3D scene.

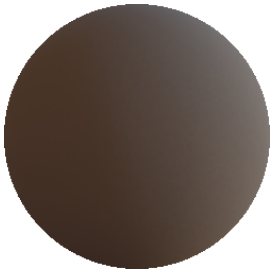
Figure 1.3: Light probes are placed in the scene to sample the light environment.

There are several ways to visualize light probes (and spherical functions in general) as shown in figure 1.4. Each projection comes with different benefits and drawbacks. The equirectangular projection shows the entire light probe contents but has distortions (figures 1.4a and 1.4b). Perspective projection leverages our human intuition but it reveals only part of the contents of the spherical function (figures 1.4c and 1.4d).

(a) Radiance, equirectangular projection.

(b) Irradiance, Equirectangular projection.

(c) Radiance, perspective projection.



(d) Irradiance, perspective projection.

Figure 1.4: Light probe examples. © by [Vog10].

In practice, you need to decide which quantity you want to store in your light probes. This choice depends on what kinds of light phenomena you want to simulate, but often light probes hold irradiance. Figure 1.4 shows two light probes containing radiance and irradiance respectively. Since irradiance $E(\boldsymbol{n})$ is essentially the sum of incoming radiance (see equation 1.1), it varies slowly as $\boldsymbol{n}$ changes.

1.3 Quality vs. Storage

Ideally, we would like to place infinitely many light probes everywhere in our game world. However, hardware resources are finite and hence there is a limit to how many light probes a video game can handle at once. Therefore we are interested in efficient ways to encode our light probes. To increase the number of light probes, modern video games use lossy compression techniques to reduce the storage requirements of each probe. But this comes at the cost of reduced probe quality. This means that we have a trade-off between probe quality and storage requirements. This begs the question:

> How should we represent and store light probes such that quality is maximized and storage requirements are minimized?

The problem of light probe encoding is the main theme of this thesis.

Efficient encoding of light probes is a hot topic among researchers [SKS02] [RH01] [WGS+07] and video game industry professionals [ON] [Pet16] alike. Most published works focus on just a single encoding method. In this thesis, I want to instead review several popular methods and compare them by measuring their approximation errors. Based on these measurements, I aim to synthesize a set of recommendations about which

encoding techniques to use for particular types of light probes.

1.4 Thesis Outline

For the convenience of the reader, I here provide a brief structural overview of this thesis.

- **Chapter 1: Introduction**
  Sets the scene for the thesis.
- **Chapter 2: Spherical Function Bases**
  Presentation of spherical function bases including introduction to spherical harmonics and spherical Gaussians. This establishes the theoretical foundation on which the thesis is based.
- **Chapter 3: Evaluation Method**
  Here I explain what spherical models I wanted to evaluate and *how* I evaluated them. This includes descriptions of error metrics, test signals, and fitting techniques.
- **Chapter 4: Analysis**
  My evaluation method generates test data, and this data is analysed in this chapter. Based on the analysis I derive a handful of key findings.
- **Chapter 5: Discussion**
  Here I put my findings into perspective and consider future work.
- **Chapter 6: Conclusion**
  Concludes the thesis by summing up the process and results.

Chapter 2

Spherical Function Bases

In this chapter I establish the theoretical foundations on which the rest of the thesis is based. I will start with a review of spherical coordinates and functions (sections 2.1 and 2.2), and then move onto function spaces (section 2.3). Finally I will cover three spherical bases that are used in the field of 3D graphics: ambient cube, spherical harmonics, and spherical Gaussians (sections 2.4, 2.5, and 2.6).

2.1 Spherical Coordinates

The 3-dimensional Cartesian coordinate system is capable of describing spherical objects. For example, you could describe points on the unit sphere by the equation $x^2+y^2+z^2 = 1$. However, sometimes it can be inconvenient to work with spherical objects in Cartesian coordinates. *The spherical coordinate system* is an alternative way to represent objects in 3-dimensional space. In this system the unit sphere is described by the equation $r = 1$ which is undeniably simpler than the Cartesian equivalent mentioned above.

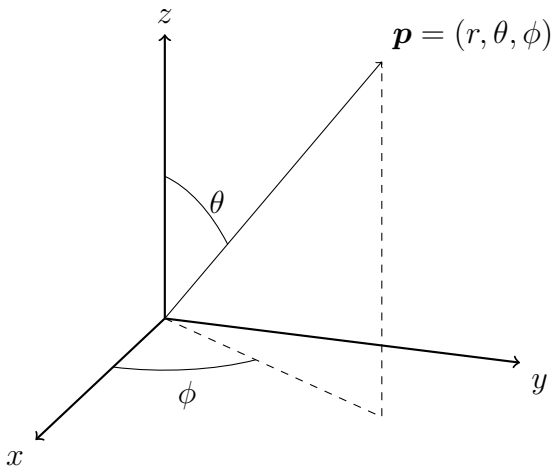
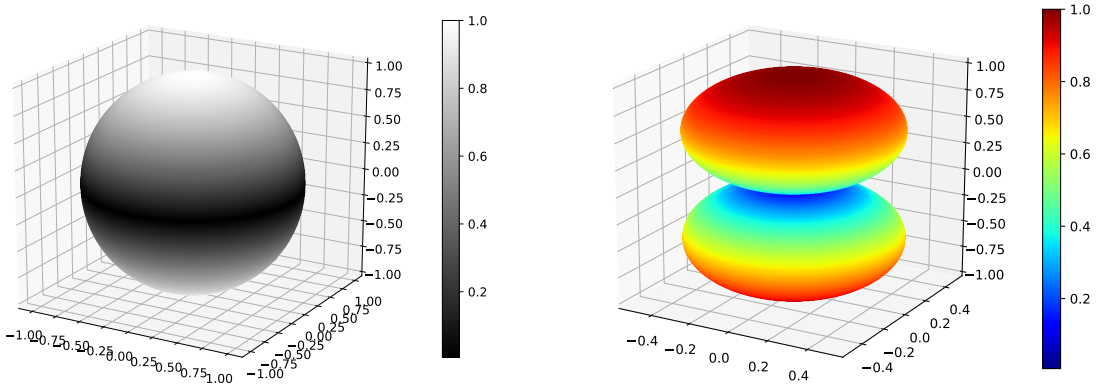


Figure 2.1: The spherical coordinate system used in this thesis. $r \in [0, \infty[$ is the distance from the origin. $\theta \in [0, \pi]$ is the *polar angle*, i.e. the angle between the vector $\boldsymbol{p}$ and the z-axis. $\phi \in [0, 2\pi]$ is the *azimuthal angle*, i.e. the angle between the x-axis and the point's projection onto the xy-plane.



(a) Constant radii.

(b) Varying radii.

Figure 2.2: Two equivalent visualizations of the spherical function $|\cos\theta|$.

There are several ways to specify a spherical coordinate system. In this thesis I will adopt the convention specified in figure 2.1. With this convention, we can represent points in $\mathbb{R}^3$ as triplets (r, θ, ϕ). However, here I am only interested in 3D *directions* which can be thought of as points on the unit 2-sphere, i.e. $\boldsymbol{p} \in S^2$ and $r = 1$. For this reason, I will often omit r and specify directions as 2-tuples, e.g. (θ, ϕ).

Using elementary trigonometry we can show the following relation between Cartesian coordinates (x, y, z) and spherical coordinates (r, θ, ϕ):

$$\begin{aligned} x &= r \sin\theta \cos\phi, \\ y &= r \sin\theta \sin\phi, \\ z &= r \cos\theta. \end{aligned} \tag{2.1}$$

This relation allows us to convert back and forth between Cartesian and spherical coordinates.

2.2 Spherical Functions

A spherical function is simply a function that maps a direction (θ, ϕ) to some set, e.g. $\mathbb{R}$. If a function conveys information about a physical phenomenon, it can also be referred to as a *signal* [Wik19]. In this thesis, I will use the word "signal" to refer to functions that describe light.

There are several ways to visualize such functions and each of them has benefits and drawbacks. In addition to the equirectangular projection and the perspective projection I introduced in chapter 1, we may also use the visualizations exemplified in figure 2.2b. This visualization draws a 3-dimensional shape where the distance between any surface point and the origin equals the function value in the corresponding direction. This projection appeals to our in-born intuition about spatiality to convey the structure of the spherical function.

As with regular function defined on the real line, spherical functions are integrable. We can understand spherical integration by thinking about the sphere surface as being divided

into a lot of tiny flat rectangles. If we make these rectangles smaller and smaller, the sum of their areas converges to the area of the sphere. Figure 2.3 shows geometrically that the area of each surface element is equal to $r^2 \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi$. We can add up all element areas by integrating over all θ and ϕ. Using these observations and still assuming that $r = 1$ we get

$$\int_{S^2} f(\omega) \, \mathrm{d}\omega = \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\pi} f(\theta, \phi) \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi.$$

This definition of sphere integration will come in useful throughout this thesis.

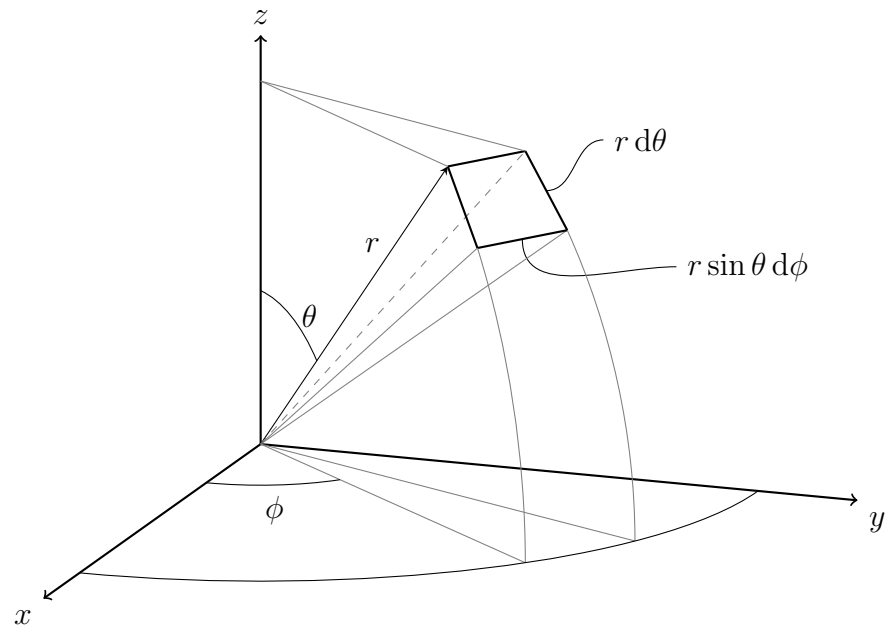


Figure 2.3: The area of the infinitesimal spherical surface element is $r^2 \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi$.

2.3 Function Spaces

One may think of vectors as arrows in euclidean space. However, the definition of vectors and vector spaces allow for a much richer interpretation and application. Every type of mathematical object that can be scaled and added can be considered a vector. This includes functions since

$$(f + g)(x) = f(x) + g(x), \qquad (2f)(x) = 2 \cdot f(x).$$

A function space is a vector space in which the basis vectors are functions. Points in a function space are themselves functions, since they are just linear combinations of the basis functions. The dot product defined in $\mathbb{R}^n$ is actually a specialization of the more general concept of *an inner product*. There are several ways to define an inner product for a given function space, but one natural definition is

$$\langle f, g \rangle = \int_D f(x) g(x) \, \mathrm{d}x \tag{2.2}$$

where D is the domain of f and g. In euclidean space, angles between vectors and vector lengths can be calculated using the dot product. The inner product induces the same quantities for functions. We can even use the inner product to project functions onto other functions by applying regular projection techniques from elementary linear algebra.

The set of possible function bases is infinite. In this thesis I will look only at basis functions defined on the sphere. In the following section I will introduce various spherical function bases and examine their properties.

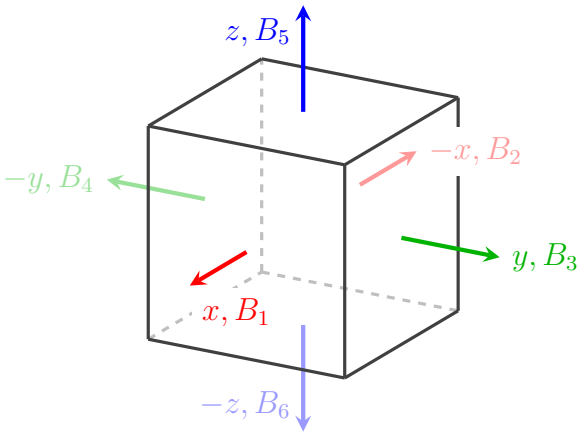


Figure 2.4: Ambient cube.

2.4 Ambient Cube

One of the first successful implementations of spatially variant indirect lighting in a major game production was Valve's source engine. Among other innovations presented in [MMG06], they introduced the *ambient cube* basis. It consists of six basis functions $\{B_1, \dots B_6\}$:

$$
\begin{aligned}
B_1(x,y,z) &= \begin{cases} x^2 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} & B_2(x,y,z) &= \begin{cases} 0 & \text{if } x \geq 0 \\ x^2 & \text{if } x < 0 \end{cases} \\
B_3(x,y,z) &= \begin{cases} y^2 & \text{if } y \geq 0 \\ 0 & \text{if } y < 0 \end{cases} & B_4(x,y,z) &= \begin{cases} 0 & \text{if } y \geq 0 \\ y^2 & \text{if } y < 0 \end{cases} \\
B_5(x,y,z) &= \begin{cases} z^2 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} & B_6(x,y,z) &= \begin{cases} 0 & \text{if } z \geq 0 \\ z^2 & \text{if } z < 0 \end{cases}
\end{aligned}
\tag{2.3}
$$

Figure 2.4 shows the association between axial directions and basis functions. With this definition each basis function covers one of the six directions. For example, $B_1(x,y,z) = x^2$ if $x \geq 0$ and 0 otherwise. I have visualized basis functions B_1 and B_5 in figure 2.5.

The full model can then be written out as a linear combination of these basis functions,

$$
f(\boldsymbol{v}) = \sum_{i=1}^{6} a_i B_i(\boldsymbol{v}), \quad \boldsymbol{v} \in \mathbb{R}^3. \tag{2.4}
$$

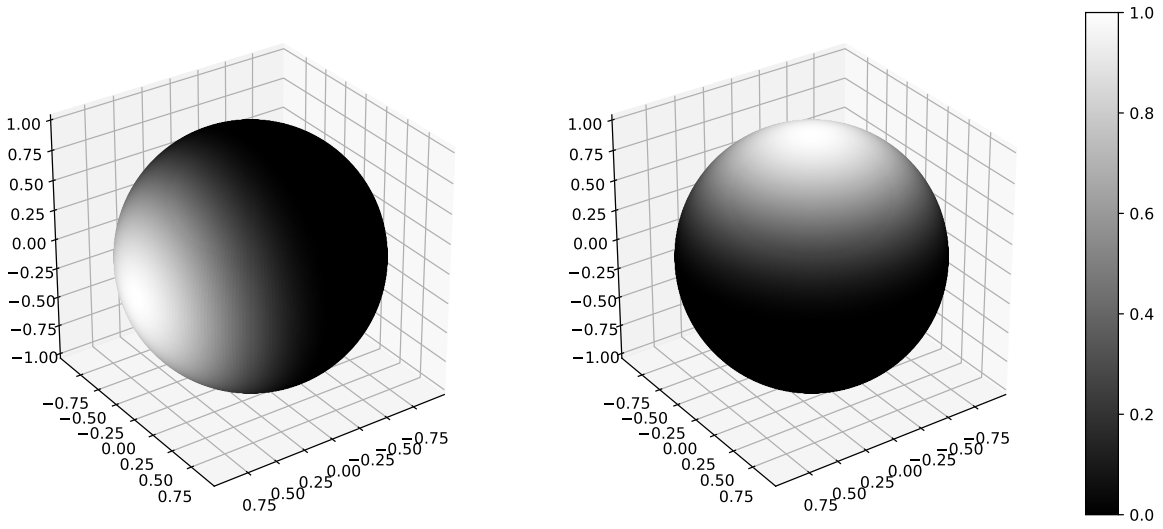


Figure 2.5: Basis functions B_1 (left) and B_5 (right) of the ambient cube model.

To find the coefficients a_i it is useful to know whether this basis is orthogonal, i.e. whether it is true that

$$\langle B_i, B_j \rangle = \int_{S^2} B_i B_j \, \mathrm{d}\omega = 0 \quad \text{when } i \neq j.$$

Due to the cases in the definition in equation 2.3 there is no straightforward way to evaluate this integral. However, from the definition we can deduce:

- $B_i(\boldsymbol{v}) \geq 0$ because it is either a square or 0.
- The support of some basis functions overlap, i.e. there exists some $\boldsymbol{v}$ such that $B_i(\boldsymbol{v})B_j(\boldsymbol{v}) > 0$ for some $i, j \in \{1, \ldots, 6\}$. This is apparent from the visualization in figure 2.5.

These observations tell us that $\langle B_i, B_j \rangle \neq 0$ for some i, j and therefore the basis is not orthogonal. Unfortunately, this means that we cannot easily use projection for finding the coefficients. Since the model is linear in the coefficients we can instead use linear least squares fitting. I will describe this technique in section 3.3.1.

2.5 Spherical Harmonics

In the 19th century the French mathematician Joseph Fourier showed that you can represent an arbitrary function $f : \mathbb{R} \to \mathbb{R}$ with period L as a infinite linear combination of sinusoids (under particular conditions). Using Eulers formula this can be expressed as

$$f(x) = \sum_{n=-\infty}^{\infty} c_n \exp\left(\frac{2\pi n \, xi}{L}\right)$$

where c_n are some constant coefficients. Each sinusoid is orthogonal to each other. A set of these therefore comprise an orthogonal function basis from which you can construct arbitrary functions. Spherical harmonics is the same idea applied to functions defined on the sphere rather than the number line. In this section I will introduce spherical harmonics and their properties.

2.5.1 Derivation

The complete derivation of the SH functions is complicated and beyond the scope of this thesis. However, instead of just citing their definition without explanation, I will in this section provide a high-level overview of how they can be derived. The following is based on [Wik18].

Laplace's equation is a famous partial differential equation in which the divergence of the gradient of a scalar field $f(\boldsymbol{x})$ is set to zero,

$$\nabla \cdot \nabla f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} = 0,$$

where x_i are components of vector $\boldsymbol{x} \in \mathbb{C}^n$. Informally this equation means that $f(\boldsymbol{x})$ is equal to the average of its neighbours. In spherical coordinates this equation becomes

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial f}{\partial r}\right) + \frac{1}{r^2 \sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta \frac{\partial f}{\partial \theta}\right) + \frac{1}{r^2 \sin^2\theta}\frac{\partial^2 f}{\partial \phi^2} = 0.$$

A common method for solving partial differential equations is *separation of variables*. If we suppose $f(r, \theta\, \phi) = R(r)Y(\theta, \phi)$ and apply this method we get two new equations involving R and Y respectively. The latter is given by

$$\frac{1}{Y}\frac{1}{\sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta \frac{\partial Y}{\partial \theta}\right) + \frac{1}{Y}\frac{1}{\sin^2\theta}\frac{\partial^2 Y}{\partial \phi^2} = -\lambda$$

where λ is the ratio constant used during separation of variables. If we apply separation variables again by supposing $Y(\theta, \phi) = P(\phi)T(\theta)$ we get two ordinary differential equations:

$$\frac{1}{P(\phi)}\frac{\partial^2 P}{\partial \phi^2} = -m^2 \qquad \text{and} \qquad \lambda \sin^2\theta + \frac{\sin\theta}{T(\theta)}\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\sin\theta \frac{\mathrm{d}T}{\mathrm{d}\theta}\right) = m^2.$$

The first equation is fairly simple and its solutions are linear combinations of $e^{\pm im\phi}$ with the constraint that m is an integer. This constraint arises from the fact that P has period 2π. The second equation is more involved but it can be shown that the solutions are multiples of $P_l^m(\cos\theta)$ for some non-negative $l \geq |m|$ where P_l^m are the associated Legendre polynomials. If we now combine these two results using our supposition $Y(\theta, \phi) = T(\theta)P(\phi)$ we arrive at the definition of the spherical harmonics:

$$Y_m^l(\theta, \phi) = K_l^m e^{im\phi} P_l^m(\cos\theta) \quad m, l \in \mathbb{N}, l \geq |m|. \tag{2.5}$$

Note that the arbitrary coefficients of $T(\theta)$ and $P(\phi)$ have here been merged into one constant K_l^m. This constant is discussed in section 2.5.3.

In summary this means that Y_m^l is the set of solutions to the angular part of Laplace's equation when expressed in spherical coordinates. It turns out that this set of functions has particularly interesting properties.

Degree l	New basis functions	Total basis functions
0	1	$1 = 1^2$
1	3	$1 + 3 = 2^2$
2	5	$4 + 5 = 3^2$
...	...	...
k	2k + 1	$(k + 1)^2$

Table 2.1: The progression of the number of spherical harmonics basis functions.

2.5.2 Family of Basis Functions

From equation 2.5 we can see that spherical harmonics are comprised of a family of functions parametrized by the degree l and order m under the constraint $l \geq |m|$. Below I show the first few basis functions for $l \leq 1$ (arguments θ and ϕ omitted for brevity):

$$Y_0^0 = \frac{1}{2}\sqrt{\frac{1}{\pi}}, \tag{2.6}$$

$$Y_1^{-1} = \frac{1}{2}\sqrt{\frac{3}{2\pi}}e^{-i\phi}\sin\theta, \qquad Y_1^0 = \frac{1}{2}\sqrt{\frac{3}{2\pi}}\cos\theta, \qquad Y_1^1 = -\frac{1}{2}\sqrt{\frac{3}{2\pi}}e^{i\phi}\sin\theta. \tag{2.7}$$

These expressions are computed directly from the definition in equation 2.5. Everytime we increase the degree l by 1, we get $2(l+1)+1$ additional basis functions with order m ranging from $-(l+1)$ to $l+1$. For example, if $l = 0$ then we get a single basis function and if $l \leq 1$ then we get $1 + 3 = 4$ basis functions (shown in equation 2.7).

Generally, the total number of basis functions for a family is given by $(k+1)^2$ where $l \leq k$ for some constant k. I exemplify this pattern in table 2.1.

2.5.3 Orthonormality

A key property of the spherical harmonics is that they are orthogonal to each other with respect to the standard inner product over the sphere. That is

$$\int_{S^2} Y_m^l(\theta,\phi)Y_{m'}^{l'}(\theta,\phi)\,\mathrm{d}\omega = \begin{cases} c_m^l & \text{if } m = m' \text{ and } l = l' \\ 0 & \text{otherwise} \end{cases}$$

where c_m^l is some constant. As mentioned in section 2.5.1 we are allowed to choose the coefficient K_l^m as we please. We can use this fact to normalize the basis. Normalization can be achieved by dividing $Y_m^l(\theta,\phi)$ with its own inner product akin to how you normalize a vector in $\mathbb{R}^3$. It can be shown that this is effectively equivalent to choosing

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

for the free coefficient in the spherical harmonics definition shown in equation 2.5. With this choice of K_l^m the basis becomes orthonormal, i.e.

$$\int_{S^2} Y_m^l(\theta,\phi)Y_{m'}^{l'}(\theta,\phi)\,\mathrm{d}\omega = \begin{cases} 1 & \text{if } m = m' \text{ and } l = l' \\ 0 & \text{otherwise} \end{cases}.$$

Orthonormal bases are particularly useful, for example with regards to function projection. Therefore I will adopt this definition of K_l^m.

2.5.4 Real Spherical Harmonics

The general spherical harmonics functions $Y_l^m(\theta, \phi)$ are complex-valued. In the context of computer graphics though, we usually restrict ourselves to a real-valued formulation. It would be tempting to define these as simply $\operatorname{Re} Y_l^m(\theta, \phi)$. However, this would strip out the sine terms of $e^{im\phi}$ effectively making the basis worse at matching odd functions. Ideally, we would like to have an equal number of sine and cosine terms. The common definition of real spherical harmonics achieves this like so:

$$y_l^m = (\theta, \phi) = \begin{cases} \sqrt{2}\operatorname{Im} Y_l^m) & \text{for } m < 0 \\ Y_l^0 & \text{for } m = 0 \\ \sqrt{2}\operatorname{Re}(Y_l^m) & \text{for } m > 0 \end{cases} = \begin{cases} \sqrt{2} K_l^m \sin(-m\phi) P_l^m(\cos\theta) & \text{for } m < 0 \\ K_l^0 P_l^0(\cos\theta) & \text{for } m = 0 \\ \sqrt{2} K_l^m \cos(m\phi) P_l^m(\cos\theta) & \text{for } m > 0 \end{cases} \tag{2.8}$$

We multiply by $\sqrt{2}$ to maintain the orthonormality described in section 2.5.3. Note that the real spherical harmonics are denoted with a lowercase y_l^m. For $m < 0$ we extract the imaginary part which contains the sines, and for $m > 0$ we extract the real part which contains the cosines. This means that the resulting real basis holds a balanced mix of sines and cosines as desired.

Using the relation between Cartesian and spherical coordinates (equation 2.1) and the definition in equation 2.8 above, we can write the first real basis functions like polynomials:

$$\begin{array}{ccccccc} & & & & y_0^0 = \dfrac{1}{2\sqrt{\pi}}, & & \\ & & y_1^{-1} = -\dfrac{1}{2\pi}\sqrt{3}y, & & y_1^0 = \dfrac{1}{2\pi}\sqrt{3}z, & & y_1^1 = -\dfrac{1}{2\pi}\sqrt{3}x, \\ y_2^{-2} = \ldots, & & y_2^{-1} = \dfrac{\sqrt{15}yz}{2\pi}, & & y_2^0 = \dfrac{\sqrt{5}(3z^2-1)}{4\pi}, & & y_2^1 = -\dfrac{\sqrt{15}xz}{2\pi}, \qquad y_2^2 = \ldots \\ & & & & \vdots & & \end{array}$$

Figure 2.6 shows the first 9 spherical harmonics functions. The functions generally become more high-frequency as the degree l increases, analogous to the behaviour of Fourier series. For example, figure 2.7 uses the spatial visualization to convey the shape of y_4^0 and y_4^2.

A useful observation is that the first basis function y_0^0 is constant and therefore it encodes signal averages. To see why this is true, suppose we want to project a spherical function $f : S^2 \to \mathbb{R}^n$ into a spherical harmonics basis. Since the basis is orthogonal we can find the first coefficient c_0 using the inner product:

$$c_0 = \langle y_0^0, f \rangle = \int_{S^2} y_0^0(\omega) f(\omega)\, \mathrm{d}\omega = \int_{S^2} \frac{1}{2\sqrt{\pi}} f(\omega)\, \mathrm{d}\omega = \frac{1}{2\sqrt{\pi}} \int_{S^2} f(\omega)\, \mathrm{d}\omega$$

Hence c_0 is proportional to the average value of f over the sphere.

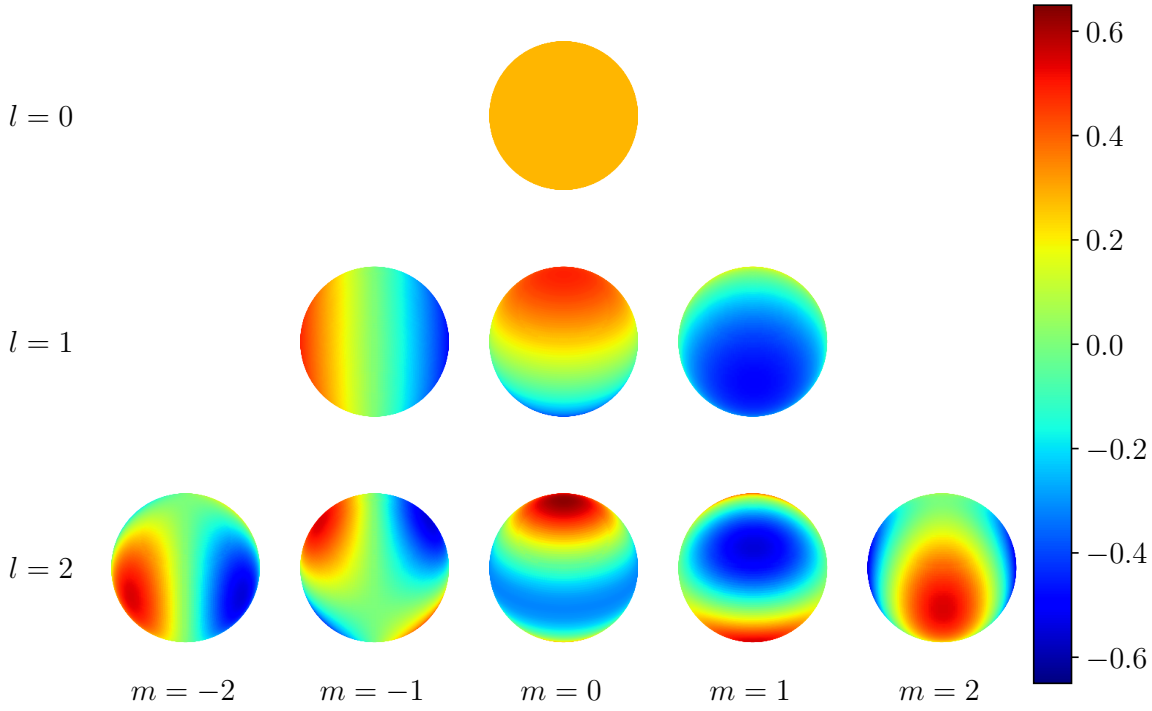


Figure 2.6: The first 9 spherical harmonics functions.

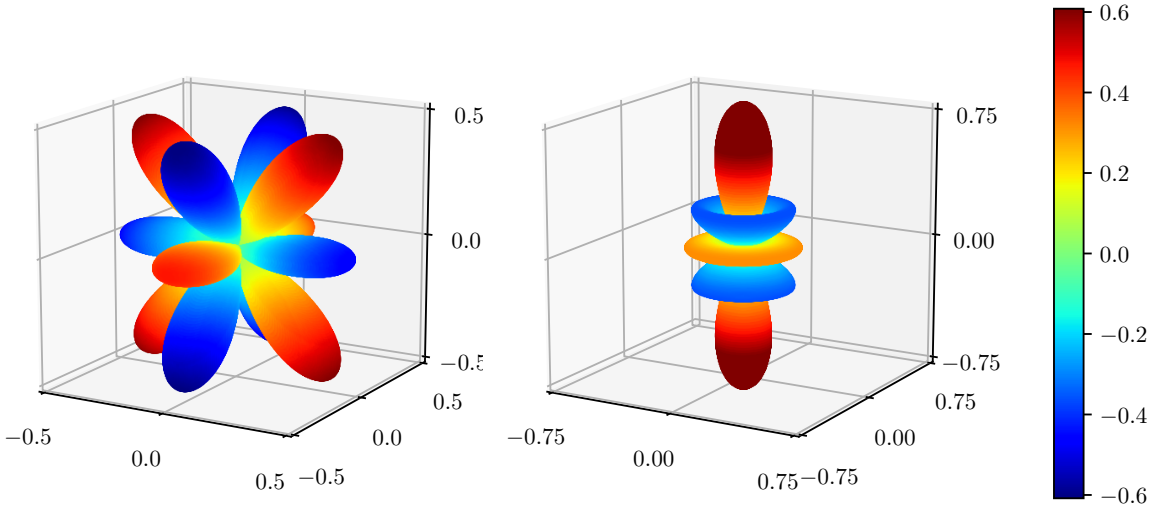


Figure 2.7: Spatial visualization of basis functions y_4^0 (left) and y_4^2 (right).

2.6 Spherical Gaussians

A multivariate Gaussian is a function of the form

$$f(\boldsymbol{x}; a, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = a \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{2.9}$$

where $\boldsymbol{\mu} \in \mathbb{R}^n$ is the mean vector and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ is the covariance matrix. I show an example of a 2-dimensional Gaussian with zero mean and $\boldsymbol{\Sigma} = \boldsymbol{I}$ in figure 2.8.

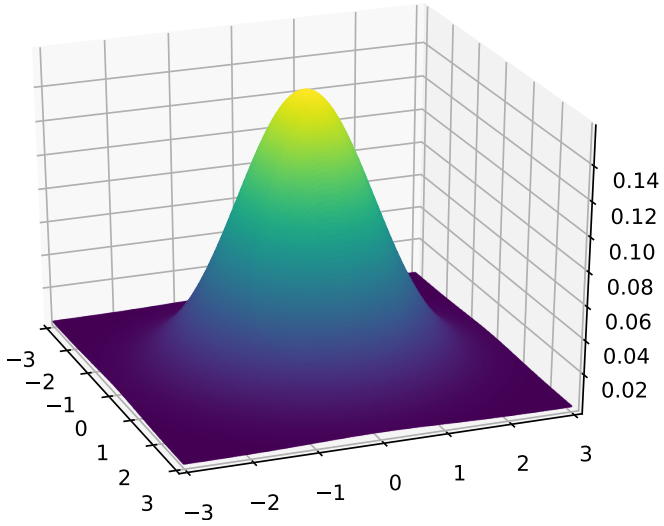


Figure 2.8: A 2-dimensional Gaussian with $\boldsymbol{\mu} = \boldsymbol{0}$ and $\boldsymbol{\Sigma} = \boldsymbol{I}$.

The parameter a controls the overall height of the curve. In the special case where

$$a = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}}$$

the Gaussian becomes a distribution with unit integral. The mean vector $\boldsymbol{\mu}$ controls the centre point of the curve, i.e.

$$\operatorname*{argmax}_x f(\boldsymbol{x}; a, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \boldsymbol{\mu}.$$

Finally, the covariance matrix $\boldsymbol{\Sigma}$ controls the spread of the Gaussian for each dimension. As $\boldsymbol{\Sigma}$ increases the bump generally gets thinner.

A spherical Gaussian is similar to the regular Gaussian except that it is defined on the sphere rather than the number line. I show a visualization of this in figure 2.9. In the plot, colour intensity corresponds to function value magnitude, e.g. higher values are brighter.

In the regular Gaussian function (equation 2.9) the exponent controls the falloff. The exponent is 0 when $\boldsymbol{x} = \boldsymbol{\mu}$ and becomes increasingly more negative as $\|\boldsymbol{x} - \boldsymbol{\mu}\|$ increases. However, this distance metric is not suitable on the sphere since we here only deal with directions rather than arbitrary points in space. Instead, spherical Gaussians use the cosine of the angle between two directions to quantify direction similarity. This particular

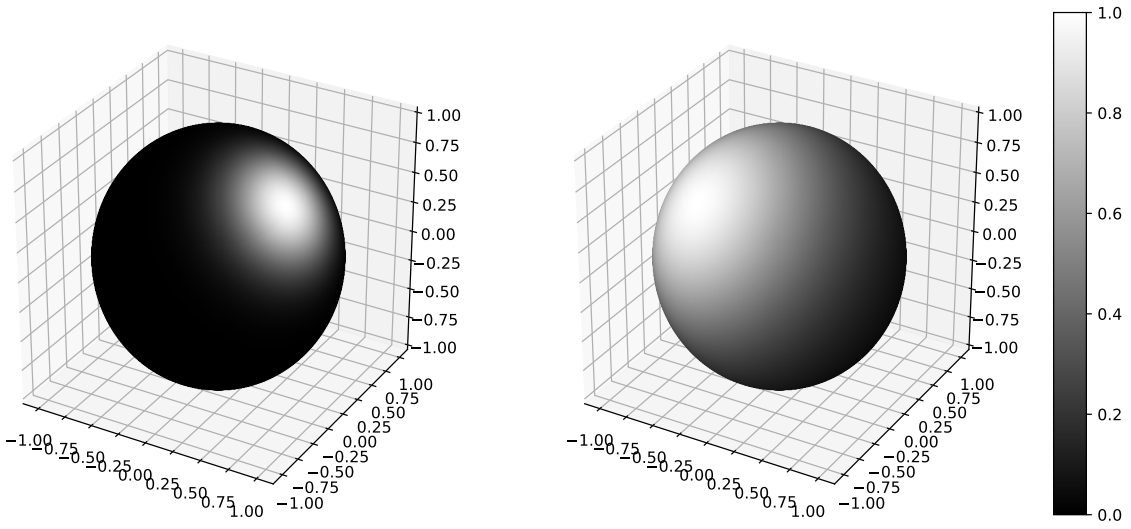


Figure 2.9: Visualization of spherical Gaussians with different parameters.

choice of distance metric has several nice properties. As shown in figure 2.10 the cosine is smooth across the sphere and its range is conveniently $[-1, 1]$. Additionally, it can be calculated efficiently using the Euclidian dot product, i.e. $\cos\theta = \boldsymbol{v_1}^T \boldsymbol{v_2}$ where $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ are direction vectors and θ is the angle between them.

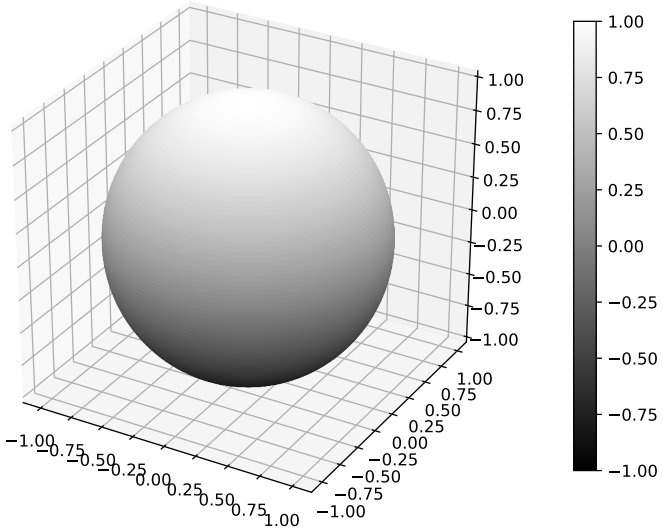


Figure 2.10: Visualization of the spherical function $\cos\theta = \boldsymbol{x}^T[0\,0\,1]$.

The full spherical Gaussian formula is defined in [WGS+07] as

$$G(\boldsymbol{v}; \boldsymbol{p}, \lambda, \mu) = \mu \exp\Big(\lambda(\boldsymbol{v}^T\boldsymbol{p} - 1)\Big). \tag{2.10}$$

The reason we subtract 1 is to make sure the exponent is never positive (to match the behaviour of the regular Gaussian). G has three parameters:

- $\boldsymbol{p} \in \mathbb{R}^3$ is the dominant direction (or symmetric axis). Since this vector represents a direction it has unit length, i.e. $\|\boldsymbol{p}\| = 1$.

- λ controls the sharpness/spread of the lobe. This is similar to the regular Gaussian's $\boldsymbol{\Sigma}$ parameter.
- μ is the amplitude. It controls the overall height of the lobe similar to the regular Gaussian's a parameter. Here μ is a scalar, but in computer graphics it is often a 3D vector corresponding to an RGB value.

These parameters enable us to rotate and shape a spherical Gaussian function, as shown in figure 2.9. This makes spherical Gaussians more versatile than the basis functions of the ambient cube (section 2.4) and spherical harmonics (section 2.5).

If we sum several spherical Gaussians each with different parameters, we can create arbitrarily complex functions:

$$f(\boldsymbol{v}; \boldsymbol{P}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \sum_{i=1}^{n} G(\boldsymbol{v}; \boldsymbol{p}_i, \lambda_i, \mu_i), \quad \boldsymbol{P} \in R^{n\times 3}, \boldsymbol{\lambda}, \boldsymbol{\mu} \in R^n.$$

Each term in the sum corresponds to a distinct spherical Gaussian. Note that this means that the dimensionality of all parameters has increased, e.g. $\boldsymbol{\mu}$ is now a vector rather than a scalar. The larger n, the more complex functions we can construct. For example, in figure 2.11 I show a sum of three spherical Gaussians with different directions $\boldsymbol{p_i}$, sharpnesses λ_i, and amplitudes μ_i.

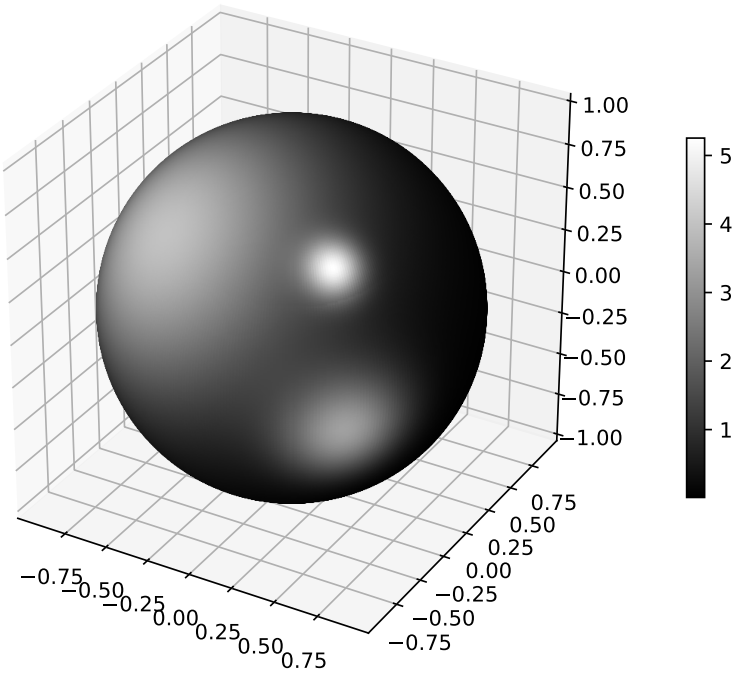


Figure 2.11: A sum of three spherical Gaussians with different parameters.

2.6.1 Properties

One reason that regular Gaussians are heavily used is that they have certain useful mathematical properties. Spherical Gaussians inherit these and I will mention a few of them here.

There exists a closed form expression for the integral of a spherical Gaussian:

$$\int_{\Omega} G(\boldsymbol{v})\,\mathrm{d}\boldsymbol{v} = \frac{2\pi\mu}{\lambda}\left(1 - e^{-2\lambda}\right).$$

Notice that the integral does not depend on the dominant direction $\boldsymbol{p}$. This is expected because we should be able to rotate a spherical Gaussian without its integral changing. The fact that we can integrate a spherical Gaussian allows us to normalize it, which make it feasible as a distribution function (just like the regular Gaussian).

The inner product of two spherical Gaussians $G_1(\boldsymbol{v}; \boldsymbol{p_1}, \lambda_1, \mu_1)$ and $G_2(\boldsymbol{v}; \boldsymbol{p_2}, \lambda_2, \mu_2)$ is derived in [TS06] and is given by

$$\langle G_1, G_2 \rangle = \int_{\Omega} G_1(\boldsymbol{v}) G_2(\boldsymbol{v})\,\mathrm{d}\boldsymbol{v} = \frac{4\pi\mu_1\mu_2}{e^{\lambda_1+\lambda_2}} \frac{\sinh d}{d} \quad \text{where} \quad d = \|\lambda_1 \boldsymbol{p_1} + \lambda_2 \boldsymbol{p_2}\|. \tag{2.11}$$

This property can be useful for spherical convolution. If two functions f and g can be approximated as spherical Gaussians, then we can efficiently calculate an approximation of $(f * g)(\boldsymbol{v})$ using equation 2.11.

Together these properties make spherical Gaussias particularly interesting in areas such as real-time graphics where computation time is of high priority and approximations are often acceptable.

Chapter 3

Evaluation Method

In this chapter I will address how I constructed a list of models, and how I subsequently evaluated these models. First, I will explain the test input signals that I used (section 3.1). With this set of test signals in mind, I will next describe the particular models (section 3.2) that I constructed based on the theoretical foundations presented in chapter 2. Next, I will explain the mathematical techniques I used to fit the test signals onto each model type (section 3.3). To complete the description of my method, I will also discuss the particular metrics (section 3.4) that I used to measure the reconstruction quality of each model. Finally, I will briefly explain how I implemented this evaluation method.

3.1 Test Signals

Before diving into the actual models it makes sense to discuss the test signals that I use for evaluation. I have included three signal types: radiance, irradiance, and self-occlusion.

Since radiance and irradiance are physical quantities that cannot be negative, their theoretical range is $[0, \infty[$. For this reason, these signals cannot be stored using the traditional image formats such as JPG, PNG, etc. They are instead encoded using the Radiance RGBE image format (`.hdr`) also known as the Picture file format [Lar].

3.1.1 Radiance

As explained in the introduction (section 1.1), radiance is a radiometric quantity that describes radiant flux, per solid angle, per square meter. Radiance signals are usually high-frequency as exemplified in figure 3.1. My test set includes 17 radiance probes downloaded from the Internet [Vog10] [fCT]. The probes have decent variety: indoors, outdoors, local lights, sky lights (cloudy day), simple motives, complex motives. In my radiance test set, the signals are typically less than 1.0 but some of them contain numerous bright spots caused by powerful light sources (maximum value is 116.5).

Figure 3.1: Examples of radiance probes. © by [Vog10].

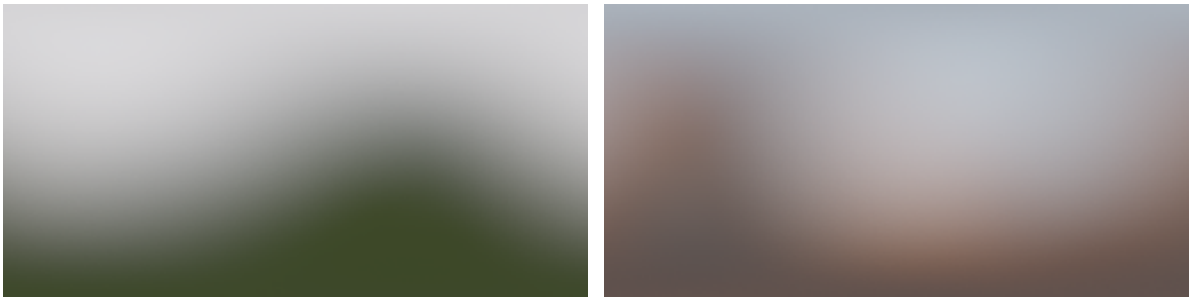


Figure 3.2: Examples of irradiance probes. © by [Vog10].

3.1.2 Irradiance

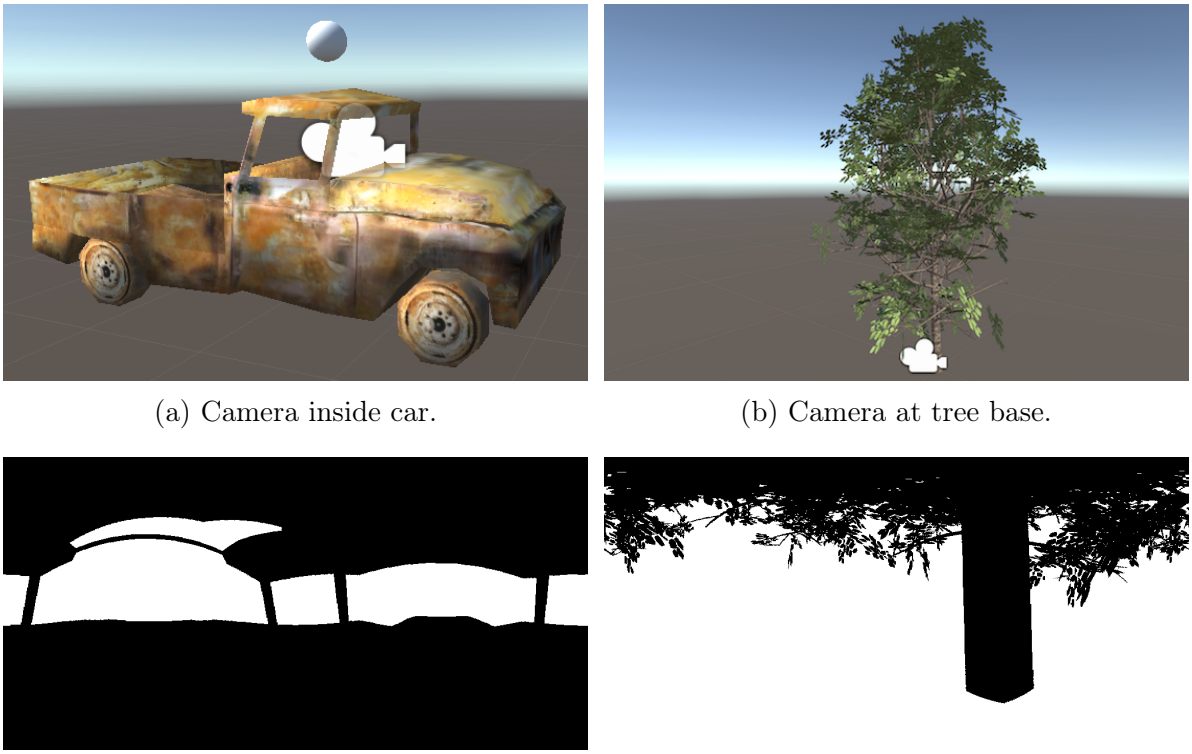
For each of the 17 radiance signals (described in section 3.1.1), we can derive a corresponding irradiance signal $E(\boldsymbol{n})$ using equation 1.1. To improve convergence time I used cosine-weighted importance sampling [PHW10, p. 668] in my implementation:

$$E(\boldsymbol{n}) = \int_{\Omega(\boldsymbol{n})} L(\boldsymbol{\omega}) \cos\theta \, \mathrm{d}\omega = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \frac{L(\omega_i)\cos\theta}{\frac{\cos\theta}{\pi}} = \lim_{N\to\infty} \frac{\pi}{N} \sum_{i=1}^{N} L(\omega_i),$$

where $\omega_i \in S^2$ is drawn from distribution $p(\theta\,\phi) = \cos(\theta)/\pi$. The irradiance examples shown in figure 3.2 are generated from the radiance signals exemplified in figure 3.1. It is important to note that these signals are clearly of low frequency and thus likely to be highly compressible. In my data set these signals have a maximum value of 4.72 but typically their values are below 1.0.

3.1.3 Self-occlusion

One of the disadvantages of probe lighting is that local geometry is ignored. For non-convex objects, in particular, this can cause the appearance of light where there really should not be any. As an example, consider the light probe shown at the top of figure 3.3a. If we were to draw a pixel inside the car, we would sample this light probe because it is the closest one, and use the sample to shade the pixel. This is clearly wrong since the light probe is not visible from inside of the car, and thus should not affect the shading. To solve this problem we need additional visibility information. One solution is to precompute objects' *self-occlusion* as a spherical signal [IS17b]. Self-occlusion refer to the degree to which an object provides shade onto itself from the surrounding environment.



(a) Camera inside car.

(b) Camera at tree base.

(c) Projection of car self-occlusion.

(d) Projection of tree self-occlusion.

Figure 3.3: Examples of equirectangular projections of self-occlusion signals.

A self-occlusion signal describes whether the surrounding environment is visible in any direction. Hence, it is a boolean signal as exemplified in figure 3.3. I generated these signals myself using the Unity game engine [Unib] and free 3D models from Unity Asset Store [Unia]. I did this by placing a camera in an partially occluded position as shown in figures 3.3a and 3.3b. Next, I rendered the surrounding object into a cubemap, i.e. each of the six axial directions is rendered into a separate square image. Finally, I transformed this cubemap into the equirectangular projection as shown in figures 3.3c and 3.3d. Due to time constraints and the manual work involved in this generation process, I included only 6 signals of this kind in my test.

3.1.4 Comparison

Since these signal types have different properties, I found it insightful to compare and contrast them. Radiance and self-occlusion may have discontinuities whereas irradiance is completely smooth. Theoretically, radiance and irradiance are unbounded, but in my data set the maximum value is 116.5. Self-occlusion only takes on the values $\{0, 1\}$. To get some insight into the values these signals take on, I gathered a few simple statistics which I have visualized in figure 3.4. Note that all signals have mean values below 1, but irradiance and radiance both go well above 1.

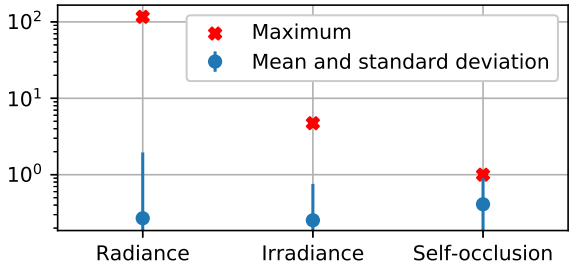


Figure 3.4: Signal statistics grouped by signal type.

3.2 Models

As described in chapter 2, a spherical basis is a set of functions that can be used to approximate an arbitrary spherical signal. If we deal with signals that represent light we have to consider that it, in addition to direction, also depends on wavelength. Ideally we would divide the human visible part of the electromagnetic spectrum into dozens or hundreds of buckets of equal size, and then treat each bucket as a separate signal. However, for video games this approach usually leads to more data than can be handled during the time budget of a single frame. A popular compromise is to use only three buckets, one for red, green, and blue respectively (RGB). I have adopted this convention in this thesis.

I will use the word *model* to refer to a basis onto which I can fit an RGB triplet of signals. In practice each colour channel is fitted onto the model separately. The result is a triplet of coefficient sets, one for each of the three colours. Occlusion signals (section 3.1.3) are exceptional in that they only have one channel.

Using the general bases discussed in chapter 2 I have constructed a total of 57 concrete models with various configurations. I show a condensed overview of all the models in table 3.1. Each model is a combination of:

- Some spherical basis functions, e.g. spherical harmonics.
- Number of bits used for floating point storage.
- Number of coefficients to store, i.e. the number of basis functions.

For all models it was natural to add variants with 16 bit and with 32 bit floating point encoding of coefficients. However, initial experimentation suggested that almost no loss occured when reducing precision from 32 bit to 16 bit. This led me to add an 8 bit precision variant as well (see section 3.5).

In the following sections I will explain the models in greater detail.

3.2.1 Ambient Cube Models

Of the bases presented in chapter 2, the ambient cube is the simplest one. The number of coefficients is fixed at 6, one for each axial direction. The only question is how to represent and store the coefficients. Since the coefficients are real numbers and we want to minimize storage costs, a natural choice is to use a floating point representation. But

Name	FP Precision (bits #)	Coefficients	Fitting Method
Ambient Cube	32, 16, 8	6	Least Squares
SH	32, 32/16, 16, 16/8, 8	4, 9, 16, 25, 36, 49	Projection
Linear SG	32, 16, 8	4, 9, 16, 25, 36, 49	Least Squares
General SG	32, 16, 8	4, 8	Gradient Descent

Table 3.1: Model overview. SH = Spherical Harmonics, SG = Spherical Gaussians, FP = floating point. Each model type has variants constructed from combinations of floating point precisions and number of coefficients.

these representations come in different precisions variants (e.g. `float` and `double` in C), so how many bits should we devote to each coefficient? Since there is no obvious answer to this question, I included models that use 8 bit, 16 bit, and 32 bit precision respectively.

The ambient cube is linear in its parameters (equation 2.4) and I decided to use linear least squares fitting. I present this fitting technique in section 3.3.1.

3.2.2 Spherical Harmonics Models

As explained in section 2.5 spherical harmonics provide an infinite series of basis functions. However, in practice we usually truncate this series to avoid having to store an infinite number of basis coefficients. The video game industry often uses 4 or 9 coefficients per channel for light probe signals [Gre03] [Slo08]. These relatively low numbers allow for fast runtime reconstruction and compact representation. I adopted this convention but also added models with 16, 25, 36, and 49 coefficients respectively. The primary motivation for adding models with this many coefficients was that they may be useful for self-occlusion signals. These signals are high frequency but require only a single channel and this may afford more coefficients.

As mentioned in section 2.5.4 the first spherical harmonics coefficient is proportional to the average of the approximated signal. The remaining coefficients encode how the signal is distributed relative to that average. For light signals the average term is usually larger than the rest of the terms, and thus it is likely to require higher encoding precision. For this reason I decided to add models where the first coefficient has high precision (32 or 16 bit) and the rest have lower precision (16 or 8 bit).

How do we fit a signal onto a model based on spherical harmonics? As with the ambient cube, the spherical harmonics representation is linear in its parameters so we could use linear least squares. However, since spherical harmonics are orthogonal I will instead do direct projection (equation 2.2) via numerical integration. Compared to least squares, this is overall simpler and we avoid computing a potentially expensive matrix inversion. I describe this fitting technique in section 3.3.2.

3.2.3 General Spherical Gaussian Models

Recall from section 2.6 that we can approximate arbitrary functions using a sum of n spherical Gaussians:

$$f(\boldsymbol{v}) \approx \sum_{i=1}^{n} G_i(\boldsymbol{v}) = \sum_{i=1}^{n} \mu_i e^{\lambda_i(\boldsymbol{v}\cdot\boldsymbol{p_i}-1)}. \tag{3.1}$$

From equation 3.1 it is clear that this model is not linear in the parameters λ, and $\boldsymbol{p}$. This means that I had to use non-linear fitting techniques. I chose to use gradient descent simply because I was familiar with this method. This fitting approach is further discussed in section 3.3.3.

Each spherical Gaussian requires four parameters: amplitude, sharpness, and 2 numbers to parametrize the direction $\boldsymbol{p_i} = \boldsymbol{p_i}(\theta_i, \phi_i)$. The total number of coefficients will thus be $4n$. To enable interesting comparisons it was important to choose n such that the final number of coefficients was close to the number of coefficients used by the other models. I wanted to add models with $n = 1, 2, 4, 8$ effectively yielding models with $4, 8, 16, 32$ coefficients respectively. However, despite application of various of the known gradient descent techniques, I did not manage to make my implemetation stable for models with $n = 4, 8$. I suspect that the gradient descent method is simply not well-suited for this model. It would be interesting to examine whether alternative non-linear optimization methods would perform better, but this is beyond the scope of this thesis. For this reason I settled on only 2 models with 4 and 8 coefficients respectively ($n = 1, 2$).

The reason that I used the word "general" in the title of this section, is that all parameters (i.e. μ, λ, and $\boldsymbol{p}$) are free in this model. This is opposed to "*Linear* Spherical Gaussians" which I will describe in section 3.2.4 where only the amplitude μ is free.

3.2.4 Linear Spherical Gaussian Models

In the general spherical Gaussian model (section 3.2.3) all parameters were free and part of the fitting problem. Inspired by [Pet16] I decided to add a simpler version of the spherical Gaussian model in which only the amplitude parameters μ_i are free, while the sharpnesses λ_i and directions $\boldsymbol{p_i}$ are fixed at some predetermined values (see equation 3.1 above). By doing this we lose some of the flexibility of spherical Gaussians, but there are benefits as well:

- Since we only need to store the amplitudes, i.e. μ_i, we can now store 4 times as many spherical Gaussians in the same amount of space (compared to the general spherical Gaussian model).
- The model becomes linear in its parameters and this means we can use the simpler and faster linear least squares fitting method (section 3.3.1).
- We avoid a bit of overhead during reconstructing. Specifically we can avoid the trigonometric function calls requried by $\boldsymbol{p_i}(\theta_i, \phi_i)$ because $\boldsymbol{p_i}$ is now predetermined.

This begs the question: If directions and sharpnesses are predetermined, how do we find appropriate values for these? Since we have no directional information a priori, it

makes sense to choose directions that are evenly spread over the sphere. To generate these directions I used the spherical version of Vogel's Method [Art15]. As illustrated in figure 3.5 this provides directions that are evenly spread across the sphere.

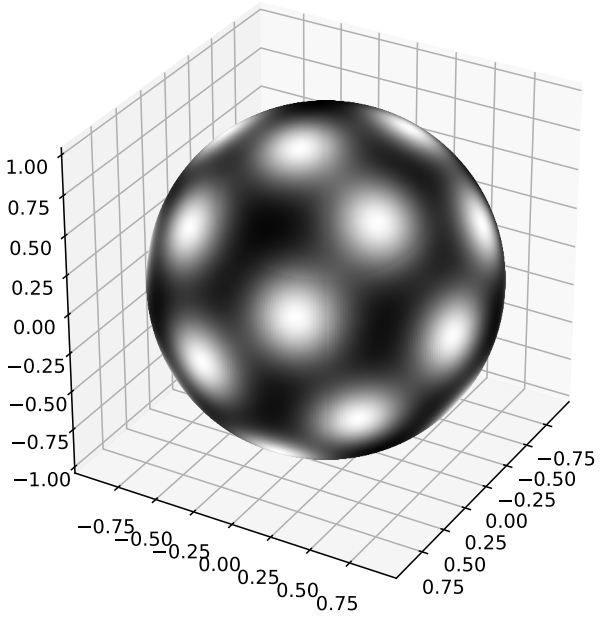


Figure 3.5: A sum of 36 Gaussians where directions are generated using Vogel's method. In this plot the sharpness parameters λ_i are exagarated for illustrative effect.

Likewise, we have no information a priori whether light from any direction is better approximated with a narrow or wide Gaussian, and therefore it makes sense to use the same sharpness parameter for all terms, i.e. $\lambda_i = \lambda$. I came up with the heuristic $\lambda = n/k$ where n is the number of Gaussians in the sum and k is a constant. The reasoning behind this choice is as follows: As the number of spherical Gaussians n increases, the distance between each Gaussian decreases, and this allows us to make each Gaussian sharper while still covering the whole sphere. I further hypothesized that the optimal k would be different for each type of signal due to their different frequency contents. Simple experimentation suggested that this was indeed correct. I found that the following values were suitable: $k = 1$ for radiance, $k = 10$ for irradiance, and $k = 4$ for occlusion signals.

I chose to use the same numbers of coefficients as in the spherical harmonics based models, i.e. 4, 9, 16, 25, 36, and 49. By using the same number of coeffients the comparison becomes more fair and thus the results become more insightful. As with the other models, I add variants of the linear spherical Gaussians with respectively 8, 16, and 32 bit floating point precision.

3.3 Function Fitting

Fitting is the process of finding a set of parameters $\boldsymbol{p}$ such that the difference between a given input function $f(x)$ and a model function $g(x; \boldsymbol{p})$ with parameters $\boldsymbol{p}$ is minimized. There are many known ways to address this problem. Which technique to use depends

on the characteristics of the particular model. In this section I will describe the fitting processes that I used for the various models described in section 3.2.

3.3.1 Linear Least Squares Fitting

Several of the spherical function models presented in this thesis are linear in their parameters, e.g. the ambient cube (section 2.4) and spherical Gaussians with fixed direction and sharpness values (section 3.2.4). We can use linear least squares fitting to find the optimal parameters for these models. In this section I will describe this technique in general so that it can be applied to any model that is linear in its parameters.

If a model is constructed as linear combinations and parametrized only by the coefficients of these combinations, i.e.

$$g(x;\boldsymbol{p}) = \sum_{j=1}^{n} p_j g_j(x),$$

then the model is said to be linear in its parameters. However, note that the $g_j(x)$'s are allowed to be non-linear. Using a given input function $f(x)$ we can generate a set of m sample points $\{(x_i, y_i)|y_i = f(x_i)\}$ simply by evaluating f at various points in its domain. The goal is to find a parameter set $\boldsymbol{p}$ such that $g(x_i;\boldsymbol{p}) = y_i$ for all i. We can formalize this as a system of equations, which can be written in matrix form:

$$\left.\begin{array}{l} p_1 g_1(x_1) + \ldots + p_n g_n(x_1) = y_1 \\ \vdots \\ p_1 g_1(x_m) + \ldots + p_n g_n(x_m) = y_m \end{array}\right\} \implies \begin{bmatrix} g_1(x_1) & \ldots & g_n(x_1) \\ & \vdots & \\ g_1(x_m) & \ldots & g_n(x_m) \end{bmatrix} \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

The matrix equation on the right can be written succinctly as $\boldsymbol{Ap} = \boldsymbol{y}$ where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. Most often we need more samples than there are basis functions in $g(x;\boldsymbol{p})$, i.e. $m > n$. This means that $\boldsymbol{Ap} = \boldsymbol{y}$ is an over-determined system with no solution. Instead we want to find "the best possible" parameter set $\boldsymbol{p}'$ which we define to be the solution that minimizes the squared distance between $\boldsymbol{Ap}$ and $\boldsymbol{y}$, i.e.

$$\boldsymbol{p}' = \underset{\boldsymbol{p}}{\operatorname{argmin}} \|\boldsymbol{Ap} - \boldsymbol{y}\|^2. \tag{3.2}$$

You could solve this minimization problem using calculus (derivative testing), but I choose to instead use the linear algebra approach. The distance $\|\boldsymbol{Ap} - \boldsymbol{y}\|$ is minimized precisely when $\boldsymbol{Ap} = \text{Proj}_{\mathcal{A}}(\boldsymbol{y})$ as shown in figure 3.6, so therefore $\boldsymbol{Ap}' = \text{Proj}_{\mathcal{A}}(\boldsymbol{y})$. This means that $\boldsymbol{Ap}' - \boldsymbol{y}$ is orthogonal to all columns of $\boldsymbol{A}$.

We can use this orthogonality property to derive an expression for $\boldsymbol{p}'$:

$$\begin{aligned} & \boldsymbol{A}^T(\boldsymbol{Ap}' - \boldsymbol{y}) = 0 \\ \implies & \boldsymbol{A}^T\boldsymbol{Ap}' = \boldsymbol{A}^T\boldsymbol{y} \\ \implies & \boldsymbol{p}' = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{y} \end{aligned}$$

With this expression we can directly calculate $\boldsymbol{p}'$, i.e. the parameter set that minimizes $\|\boldsymbol{Ap} - \boldsymbol{y}\|^2$. When working with the least squares method, we have to be careful about

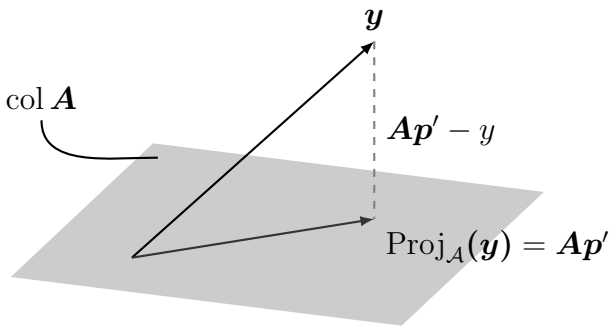


Figure 3.6: Projecting vector $\boldsymbol{y}$ onto col $\boldsymbol{A}$.

how we interpret this. Ideally we would like to minimize the absolute value of the difference between $f(x)$ and $g(x;\boldsymbol{p})$ over all x. However, this is not what the $\boldsymbol{p'}$ solution represents. Consider that

$$\|\boldsymbol{Ap} - \boldsymbol{y}\|^2 = \sum_{i=1}^{m} \left[(\boldsymbol{Ap})_i - y_i\right]^2 = \sum_{i=1}^{m} \left[g(x_i;\boldsymbol{p}) - y_i\right]^2.$$

From this we can see that $\boldsymbol{p'}$ instead minimizes the *squares* of the differences. Although this is not ideal, least squares fitting remains a valuable tool in practice due to its simplicity.

3.3.2 Orthogonal Projection

Linear function fitting is extraordinarily easy when dealing with orthogonal bases. If g_i and g_j are part of an orthonormal basis we know that

$$\langle g_i, g_j \rangle = \begin{cases} 1 \text{ if } i = j, \\ 0 \text{ otherwise.} \end{cases}$$

Suppose an arbitrary function $f(x)$ can be decomposed in terms of the basis functions $\{g_i\}$ where $i \in \{1, \ldots, n\}$, then

$$f(x) = \sum_{i=1}^{n} c_i g_i(x).$$

Now take the inner product with respect to some function in the basis g_j on both sides:

$$\langle f, g_j \rangle = \left\langle \sum_{i=1}^{n} c_i g_i(x), g_j(x) \right\rangle = \sum_{i=1}^{n} c_i \langle g_i, g_j \rangle = c_j. \tag{3.3}$$

We see that coefficient c_j is equal to the inner product of f and g_j (only if $\{g_i\}$ is orthogonal). As explained in section 2.5.3, one of the great properties of spherical harmonics is that they are in fact orthogonal. This means that for arbitrary function f we can calculate its coefficients simply by using equation 3.3 with the real spherical harmonics basis functions (equation 2.8), i.e.

$$c_l^m = \langle f, y_l^m \rangle = \int_{S^2} f(\omega) y_l^m(\omega)\, \mathrm{d}\omega.$$

An interesting fact is that the linear least squares technique described in section 3.3.1 turns out to reduce to integrating against basis functions (as in equation 3.3) if the basis is orthogonal. That is, orthogonal projection and linear least squares fitting are equivalent in this case. For a full argument why this is true, see [Slo08, Appendix A6].

3.3.3 Non-linear Fitting

As mentioned in section 3.2.3 I wanted to fit the general spherical Gaussians using gradient descent. Therefore I need to define a loss function and derive its gradient, and I will do this in this section. I attempted to perform the following derivation purely using matrix calculus. With help from my advisor I got half-way through but ultimately gave up because I kept running into tensors and other exotic mathematical objects, which I had no prior experience with. Below I will instead find the partial derivatives for each component separately.

Suppose we are given a data set $(\boldsymbol{x_i}, y_i)$ composed of n samples from a real-valued function defined on the sphere. Recall that $\boldsymbol{x_i}$ is a direction vector, i.e. $\boldsymbol{x_i} \in \mathbb{R}^3$ and $\|\boldsymbol{x_i}\| = 1$. In the context of this thesis, $(\boldsymbol{x_i}, y_i)$ are samples of light signals, e.g. radiance.

The model that we want to fit to, is a sum of k spherical Gaussians,

$$G(\boldsymbol{v}) = \sum_{j=1}^{k} \mu_j e^{\lambda_j(\boldsymbol{v}\cdot\boldsymbol{p_j}-1)},$$

where μ_j, λ_j, and $\boldsymbol{p_j}$ are the parameters for the jth Gaussian (section 2.6).

We can then define the ith error as the difference between the ith sample and the model:

$$\varepsilon_i(\boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{P}) = y_i - G(\boldsymbol{x_i}) = y_i - \sum_{j=1}^{k} \mu_j e^{\lambda_j(\boldsymbol{x_i}\cdot\boldsymbol{p_j}-1)}.$$

where the function arguments are spherical Gaussian parameters (see section 2.6):

- $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)$ is a vector of amplitude parameters.
- $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_k$ is a vector of sharpness parameters.
- $\boldsymbol{P} = (\boldsymbol{p_1}, \ldots, \boldsymbol{p_k})$ is a matrix where each column is a dominant direction parameter.

The full error function is then given by

$$f(\boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{P}) = \sum_{i=1}^{n} \left[\varepsilon_i(\boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{P})\right]^2$$

In order to perform gradient descent, I need to compute the gradient of f. To simplify this calculation I choose to think of f's arguments as scalars rather than vectors and matrices. Also I will use the fact that any direction vector $\boldsymbol{v_i}$ can be calculated from its polar and azimuthal angles, i.e. $\boldsymbol{v_i} = \boldsymbol{v_i}(\theta_{v,i}, \phi_{v,i})$ (see section 2.1 about spherical coordinates). Using this I can reparametrize f:

$$f(\boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{p}) = f(\mu_1, \ldots, \mu_k, \lambda_1, \ldots, \lambda_k, \theta_{p,1} \ \ldots, \theta_{p,k}, \phi_{p,1}, \ldots, \phi_{p,k})$$

The elements of the gradient ∇f is given by a vector of the function's partial derivatives. To save computation I first find a general expression for $\frac{\partial f}{\partial z}$ where z is any argument of f. For brevity I will omit arguments of the error term ε_i.

$$\begin{aligned}\frac{\partial f}{\partial z} &= \frac{\partial}{\partial z}\sum_{i=1}^{n}\varepsilon_i^2 = 2\sum_{i=1}^{n}\varepsilon_i\frac{\partial \varepsilon_i}{\partial z} = 2\sum_{i=1}^{n}\varepsilon_i\frac{\partial}{\partial z}\left(y_i - G(\boldsymbol{x_i})\right)\\ &= -2\sum_{i=1}^{n}\varepsilon_i\sum_{j=1}^{k}\frac{\partial}{\partial z}\mu_j e^{\lambda_j(\boldsymbol{x_i}\cdot\boldsymbol{p_j}-1)}\end{aligned} \tag{3.4}$$

I can now use the general expression in equation 3.4 to find the specific partials for each "category" of parameters. For arbitrary but fixed m such that $1 \geq m \geq k$:

$$\begin{aligned}\frac{\partial f}{\partial \mu_m} &= -2\sum_{i=1}^{n}\varepsilon_i\sum_{j=1}^{k}\frac{\partial}{\partial \mu_m}\mu_j e^{\lambda_j(\boldsymbol{x_i}\cdot\boldsymbol{p_j}-1)} = -2\sum_{i=1}^{n}\varepsilon_i e^{\lambda_m(\boldsymbol{x_i}\cdot\boldsymbol{p_m}-1)}\\ \frac{\partial f}{\partial \lambda_m} &= -2\sum_{i=1}^{n}\varepsilon_i\sum_{j=1}^{k}\frac{\partial}{\partial \lambda_m}\mu_j e^{\lambda_j(\boldsymbol{x_i}\cdot\boldsymbol{p_j}-1)} = -2\mu_m\sum_{i=1}^{n}\varepsilon_i e^{\lambda_m(\boldsymbol{x_i}\cdot\boldsymbol{p_m}-1)}(\boldsymbol{x_i}\cdot\boldsymbol{p_m}-1)\end{aligned}$$

The angular parts are slightly more complicated. I will here use the fact that $\boldsymbol{x_i}$ and $\boldsymbol{p_j}$ can be rewritten in terms of their angular components (as discussed in section 2.1):

$$\begin{aligned}\frac{\partial f}{\partial \theta_{p,m}} &= -2\sum_{i=1}^{n}\varepsilon_i\sum_{j=1}^{k}\frac{\partial}{\partial \theta_{p,m}}\mu_j e^{\lambda_j(\boldsymbol{x_i}\cdot\boldsymbol{p_j}-1)} = -2\sum_{i=1}^{n}\varepsilon_i\sum_{j=1}^{k}\mu_j e^{\lambda_j(\boldsymbol{x_i}\cdot\boldsymbol{p_j}-1)}\lambda_j\frac{\partial}{\partial \theta_{p,m}}\left(\boldsymbol{x_i}\cdot\boldsymbol{p_j}\right)\\ &= -2\mu_m\lambda_m\sum_{i=1}^{n}\varepsilon_i e^{\lambda_m(\boldsymbol{x_i}\cdot\boldsymbol{p_m}-1)}\left(\sin\theta_{x,i}\cos\theta_{p,m}\cos\left(\phi_{x,i}+\phi_{p,m}\right)-\cos\theta_{x,i}\sin\theta_{p,m}\right).\end{aligned}$$

To reach this expression I used the chain rule and the trigonemetric identity $\cos(a+b) = \cos a\cos b - \sin a\sin b$.

Similarly, I derive the partial with respect to $\phi_{p,m}$:

$$\begin{aligned}\frac{\partial f}{\partial \phi_{p,m}} &= -2\sum_{i=1}^{n}\varepsilon_i\sum_{j=1}^{k}\frac{\partial}{\partial \phi_{p,m}}\mu_j e^{\lambda_j(\boldsymbol{x_i}\cdot\boldsymbol{p_j}-1)} = -2\sum_{i=1}^{n}\varepsilon_i\sum_{j=1}^{k}\mu_j e^{\lambda_j(\boldsymbol{x_i}\cdot\boldsymbol{p_j}-1)}\lambda_j\frac{\partial}{\partial \phi_{p,m}}\left(\boldsymbol{x_i}\cdot\boldsymbol{p_j}\right)\\ &= -2\mu_m\lambda_m\sum_{i=1}^{n}\varepsilon_i e^{\lambda_m(\boldsymbol{x_i}\cdot\boldsymbol{p_m}-1)}\sin\theta_{x,i}\sin\theta_{p,m}\sin\left(\phi_{x,i}-\phi_{p,m}\right)\end{aligned}$$

Here I used the identity $\sin(a-b) = \sin a\cos b - \cos a\sin b$.

Now that I have identified all partials of f I am able to calculate the full gradient vector for any point:

$$\nabla f = \left[\frac{\partial f}{\partial \mu_1}\ \cdots\ \frac{\partial f}{\partial \mu_k}\ \ \frac{\partial f}{\partial \lambda_1}\ \cdots\ \frac{\partial f}{\partial \lambda_k}\ \ \frac{\partial f}{\partial \theta_{p,1}}\ \cdots\ \frac{\partial f}{\partial \theta_{p,k}}\ \ \frac{\partial f}{\partial \phi_{p,1}}\ \cdots\ \frac{\partial f}{\partial \phi_{p,k}}\right]^T$$

This allows me to perform gradient descent by iteratively stepping in the opposite direction of this gradient vector to find local minima.

3.4 Metrics

Given an approximation of a signal, how do you determine how good it is? For that you need an error metric. An error metric is a function that maps an approximation to a numerical value. This value usually represents error, so smaller is better. There are many popular error metrics but you have to be careful which ones you choose. The well-known root mean square error (RMSE) is often used for its simplicity, but since it squares each error term, it causes larger error terms to have disproportionately large effect on the total error. I also considered Mean absolute percentage error (MAPE) and Symmetric mean absolute percentage error (SMAPE) but I did not find them suitable due to various shortcomings.

To avoid having only a single perspective on my data, I chose two fundamentally different error metrics. I will now describe these.

3.4.1 Relative Mean Absolute Error (RMAE)

In statistics the mean absolute error (MAE) is the sum of the absolute differences divided by the number of samples:

$$\text{MAE} = \frac{1}{n}\sum_{i}^{n}|y_i - x_i|$$

where y_i is the original and x_i is the reconstruction. This is fine for some use cases. However, suppose we have a reconstruction with MAE = 1. Is this a *good* error? That most likely depends on the mean value $\mu = \sum_{i=1}^{n} y_i/n$ of the original signal. For example, if $\mu = 1$ then an error of 1 sounds bad, whereas if $\mu = 1000$ then an error of 1 sounds decent. To address this issue, I chose instead to use a relative variant of MAE:

$$\text{RMAE} = \frac{\text{MAE}}{\frac{1}{n}\sum_{i}^{n} y_i} = \frac{1}{\mu n}\sum_{i}^{n}|y_i - x_i|.$$

Here MAE is divided by μ to get the error relative to the average of the original signal. This adjustment allows me to meaningfully take the mean of each test signal's RMAE. The mean of the RMAEs can then be considered a net error that describes how well a model performs across the entire set of test signals.

3.4.2 Structural Similarity Index

Error metrics used in the field of statistics are objective and based on numerical quantities. But for some data sets one could argue that what really matters is the subjective human perception of the error. The structural similarity (SSIM) index is a method introduced in 2004 [WBSS04] that aims to predict the *perceived* quality of digital images. SSIM strikes a balance between computational simplicity, accuracy of prediction, and intuitiveness of design [Bru12].

My signals are spherical but SSIM works only on flat 2D images. I solved this by projecting each signal as shown in figure 3.7. Then I apply SSIM on these new images. I

(a) Original probe data. © by [Vog10].

(b) 6-sided sphere projection.

Figure 3.7: SSIM metric is applied to projections like shown in (b).

assessed that this projection was suitable, since it resembles how light probes are used in video games in practice.

A full explanation of the SSIM formula is beyond the scope of this thesis. For a thorough introduction to SSIM, see [Bru12].

3.5 Implementation

I implemented my evaluation framework in the programming language Rust [Rus]. I chose Rust because I find it expressive and safe, and because it compiles to fast machine code. Besides implementing all models mentioned in chapter 2, I wrote procedures for various spherical projections (section 3.4.2), gradient descent, image format transformations, coordinate conversions, simple tonemapping, and radiance to irradiance convolution (section 3.1.2). I used open source libraries for matrix inversions [Cro], 16 bit floating point encoding [Lon], structural similarity error calculation [Les], and general image file encoding/decoding [N+].

Since I could not find any 8 bit floating point encoding library, I wrote a simple encoder myself inspired by the source code of a library called "half" [Lon]. In this implementation I chose to have 3 exponent bits and 4 mantissa bits (leaving a single bit for sign). This partitioning yields a dynamic range of -15.5 to 15.5 while still preserving some precision near zero.

The application was designed to be extensible. It defines simple interfaces (Rust traits)

```
fn fit(
  models: &Vec<Model>, input_type: InputType,
  test_signals: &Vec<HDRImage>, metric: &Metric
) -> Vec<Dist1D>
```

Listing 1: Signature of fitting function (slightly simplified). The function accepts a metric, an array of models, and an array of test signals. It returns an array of error distributions, one for each model.

for models and metrics with the intention to make it easy to add more. Likewise, it should be trivial to add new signal types. As an example of this, consider the function signature shown in listing 1. This function accepts an array of models, an array of input images, and a metric to use for evaluation. The motivation for this design was to encourage experimentation.

See appendix A for information on how to compile and use the application.

Chapter 4

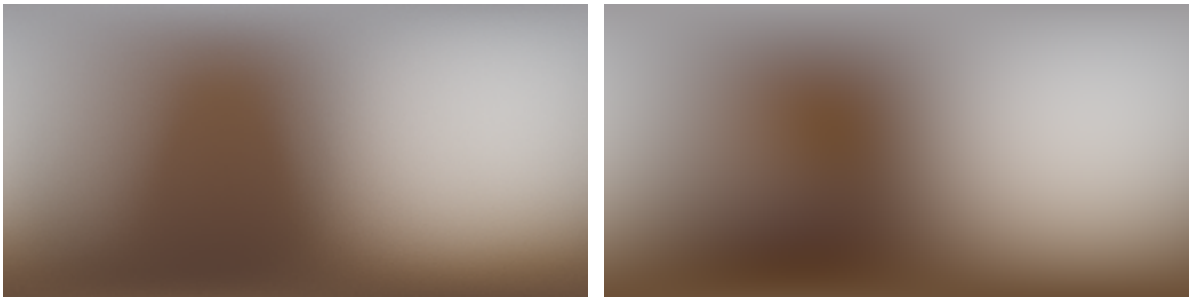
Analysis

Using the models and implementation discussed in chapter 3, I produced a set of error measurements based on the metrics presented in section 3.4. In this chapter I aim to make sense of this data. Since each signal type has different characteristics (as discussed in section 3.1), I will analyze each of them separately.

Recall that the models I presented in section 3.2 use a varying amounts of space. It is expected that the models using large numbers of coefficients and the models using high floating point precision will perform better than the more austere models. But which models provide the best reconstruction quality *relative* to their space requirements? I will attempt to answer this question in the following sections.

4.1 Irradiance

As discussed in section 3.1.2, irradiance is usually very low frequency. Therefore I expected it to be a good candidate for compression. I show an irradiance reconstruction example in figure 4.1 and as expected the reconstruction looks reasonable, at least at a first glance. The reconstructed signal in figure 4.1b has low error and it is created from only 54 bytes of data in total (9 coefficients, 3 channels, 16 bits each). This is a great example of why spherical harmonics (and similar bases) are a great tool for representing irradiance signals.



(a) Original, 256 × 128 HDR file, 99 KiB. (b) Spherical Harmonics, degree 2 (54 bytes).

Figure 4.1: Irradiance reconstruction example.

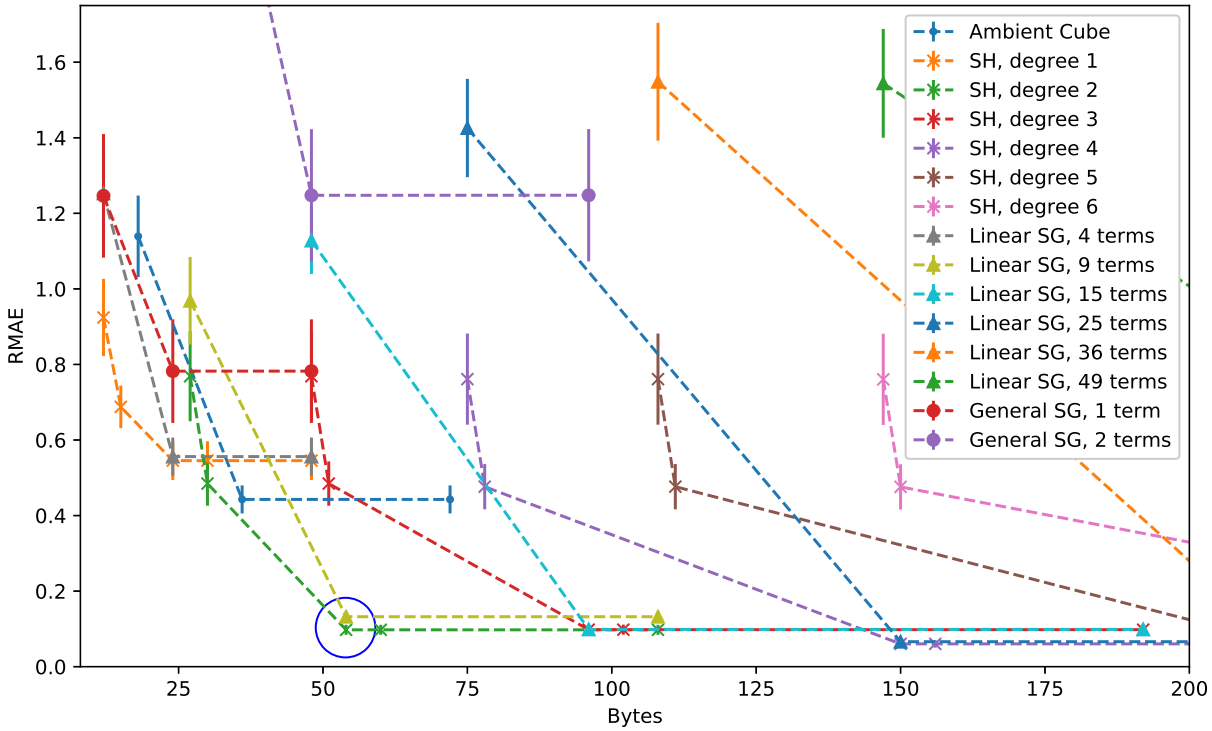


Figure 4.2: Relative mean absolute error versus space requirements for irradiance signals. Each model has been given a distinct colour. Each model family has been assigned a distinct symbol (cross, triangle, disc). Vertical line segments indicate relative standard deviations. SH = Spherical Harmonics, SG = Spherical Gaussians.

Figure 4.2 shows how well each model performs on average as measured by the RMAE metric (section 3.4.1). I have zoomed in on the lower left of the plot since this region contains models that have low error *and* low space requirements. The dashed lines connect models that have the same underlying mathematical representation but differ in floating point precision. For example, the ambient cube family is comprised of models with 8, 16, and 32 bit precision respectively, and this is represented as three dots connected by dashed lines (blue).

From the plot it is clear that error generally decreases as space usage rises (as expected). The horizontal lines represent increases in storage usage that does not yield any meaningful error reduction. Upon further examination, I learnt that these horizontal lines represent the jump from 32 bit precision to 16 bit precision. This tells us that there is virtually no reason to store coefficients at 32 bit because 16 bit is enough. This deserves to be boxed:

> 16 bit of precision is often enough to encode irradiance coefficients.

The lower left of figure 4.2 represents low error, low space usage. It shows that the spherical harmonics degree 2 model does particularly well by striking a good trade-off between error and space requirements (marked with blue circle). The 9 term spherical Gaussians is a close runner-up. Note that both models even have low standard deviation (indicated by the vertical line segment in the figure). Models with larger space requirements than these two yield almost the same error. Thus, these models are presumably

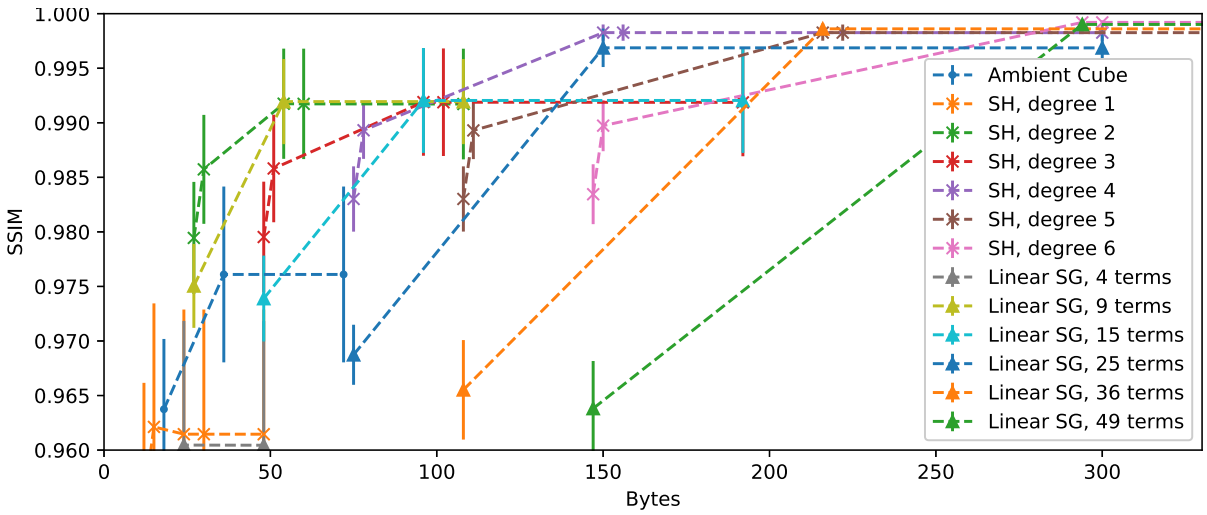


Figure 4.3: Structural similarity index versus space requirements for irradiance signals.

not worthwhile. Therefore:

> For irradiance signals, Spherical Harmonics Degree 2 and 9-term Linear Spherical Gaussians often represent a suitable trade-off in quality versus space requirements.

Additional terms/coefficients lead to rapidly diminishing returns.

Figure 4.3 shows space requirement versus the measured structural similarity errors for irradiance. This data support what I already concluded from figure 4.2.

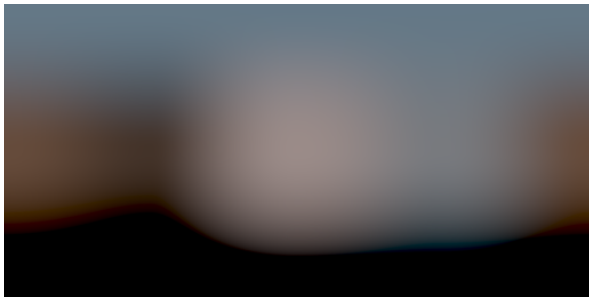
4.2 Radiance

Radiance is perhaps the most challenging of the three signal types. It has a wider range than self-occlusion, and it is of higher frequency than irradiance. Therefore I expected my models to provide only very crude approximations of this type of signal. I show several reconstruction examples in figure 4.4. The approximation is very rough indeed. Note however, that the space requirements decrease dramatically. The size of the original probe is 1,6 MiB (1024 × 512 HDR file) and spherical harmonics-based approximation (figure 4.4d) is only 294 bytes, a 5707x improvement.

In figure 4.5 I show a plot relating space usage to the RMAE metric (section 3.4.1). Note that errors are generally large compared to the corresponding irradiance plot (figure 4.2). This is presumably because radiance usually contains high frequency contents, which are lost during the fitting process. This plot surprised me: It seems there is a tendency for errors to *increase* as space usage increases. I speculated that this is because most models optimize with respect to *squared difference* between original and model, whereas this plot shows the linear difference. I verified this hypothesis by instead plotting according to squared errors, and with this error metric the curve indeed decreased as space usage increased, as expected (plot omitted for brevity).

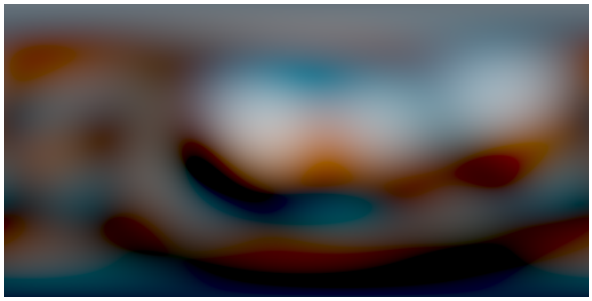
As was the case with irradiance, we see that 16 bit is more than enough to represent the coefficients. However, note that I have excluded the 8 bit variants of linear spherical

(a) Original, 1024 × 512 HDR file (1,7 MiB).



(b) Ambient Cube (12 bytes).

(c) Linear Gaussians, 49 terms (98 bytes).



(d) Spherical Harmonics, degree 6 (98 bytes).

Figure 4.4: Comparison of reconstructions of a radiance signal.
Original probe © by [Vog10].

Gaussians models. This is simply because their coefficients were out of the 8 bit floating point range, and thus not meaningful to include here. This shows a clear limitation of 8 bit floating point with regards to radiance.

> 16 bit of precision is enough to encode radiance coefficients. 8 bit precision often fails completely for linear Spherical Gaussians.

We see that error is decreasing significantly as space usage increases, even as space usage gets large. Note how this is different from irradiance that had severe diminishing returns as storage increased. Interestingly, linear spherical Gaussians consistently perform better than spherical harmonics. For example, the linear spherical Gaussians with 15 terms performs better than spherical harmonics degree 6, even though the latter uses more than three times as much storage (90 vs 294 bytes, assuming 16 bit coefficients).

> Linear spherical Gaussians are consistently better than spherical harmonics at representing radiance.

Intuitively, I suspect this is because the spherical Gaussians basis functions are more pointy than spherical harmonics, and hence they are simply a better fit for radiance which tends to have spikes (relatively small bright light sources).

Another interesting observation is that the non-linear spherical Gaussian performs the best among models that use less than 50 bytes. This makes me wonder how well this model would have faired if it was not capped at 2 terms (see section 3.2.3).

For the sake of brevity, I did not include the plot for the SSIM radiance errors. However, it

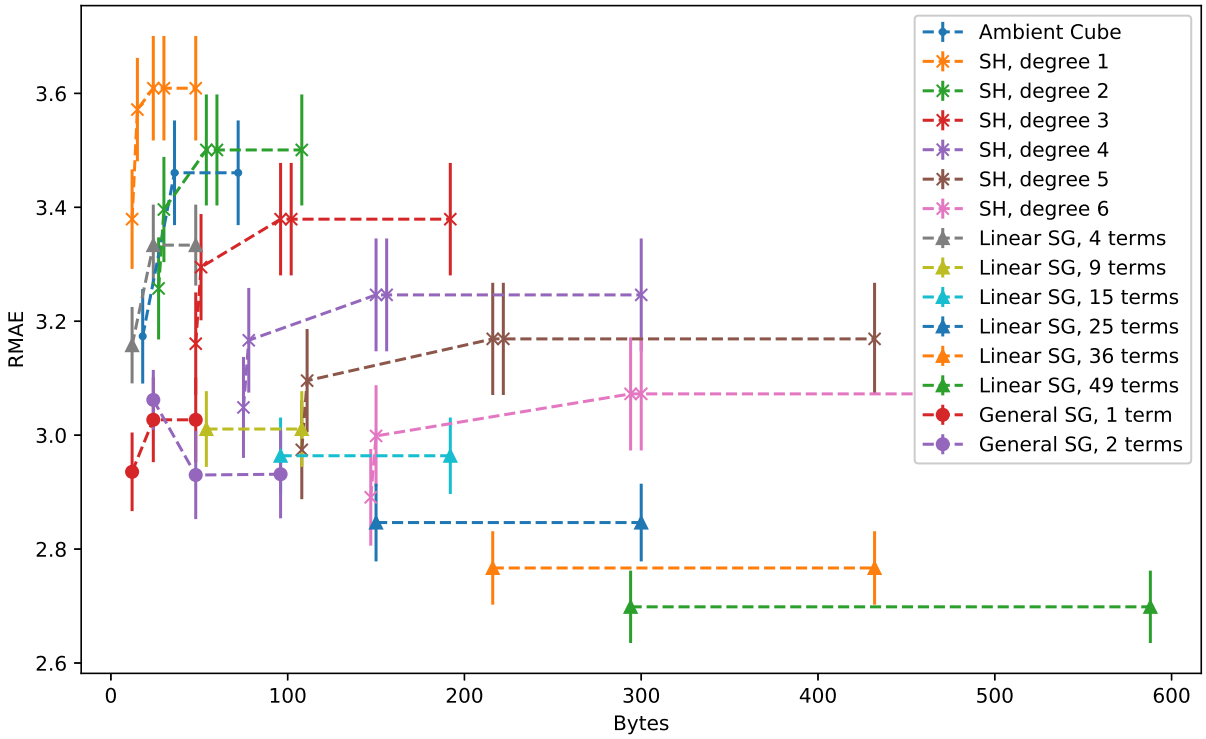


Figure 4.5: Relative mean absolute error versus space requirements for radiance signals. Each model has been given a distinct colour. Each model family has been assigned a distinct symbol (cross, triangle, disc). Vertical line segments indicate relative standard deviations. SH = Spherical Harmonics, SG = Spherical Gaussians.

surprisingly showed that the linear Gaussian model with 9 terms outperformed the linear Gaussian model with 49 terms. I investigated this and found that it is presumably because of the imperfect sharpness parameter heuristic ($\lambda = n/k$) discussed in section 3.2.4. The problem is illustrated in figure 4.6. This suggests that a better heuristic or alternative method would improve the performance of spherical Gaussians even more. Solving this issue is beyond the scope of this thesis and left for future research (section 5.3).

4.3 Self-occlusion

Self-occlusion signals can change abruptly from 0 to 1, but their ranges only contain two elements, 0 and 1. Beforehand, I would expect these signals to be easier to fit than radiance due to the reduced range, but harder to fit than irradiance due to their discontinuities. I show reconstruction examples in figure 4.7. As expected, the reconstructions are only rough approximations but decent ones nonetheless. The results are quite impressive when you consider the reduction in storage requirements. In the case shown in figure 4.7c, storage is reduced from 23 KiB (1024 × 512 PNG file) to only 98 bytes, a 240x improvement.

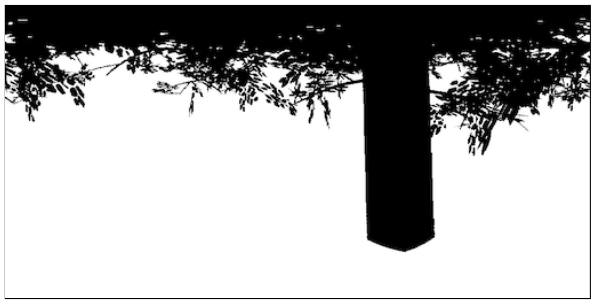
As with the other signal types, in figure 4.8 I show a plot relating space requirement to RMAE (section 3.4.1). As expected, errors are generally higher than irradiance but lower than radiance. From the figure it is clear that linear spherical Gaussians once again outperform spherical harmonics (as was the case for radiance).

(a) Original. © by [Vog10].

(b) Linear Spherical Gaussian, 9 terms.

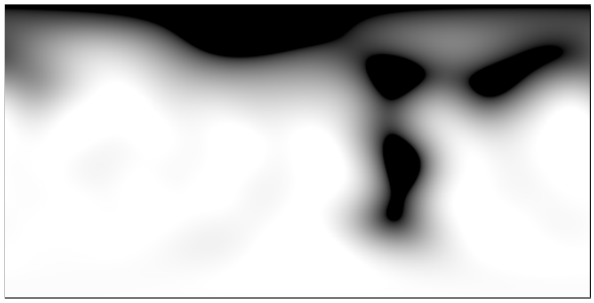
(c) Linear Spherical Gaussian, 49 terms.

Figure 4.6: These images show projections performed as part of the SSIM measurement (section 3.4.2). My sharpness heuristic (section 3.2.4) makes the Gaussian bump too sharp when there are many terms. This results in bright spots between dark bands, as shown in (c) above.

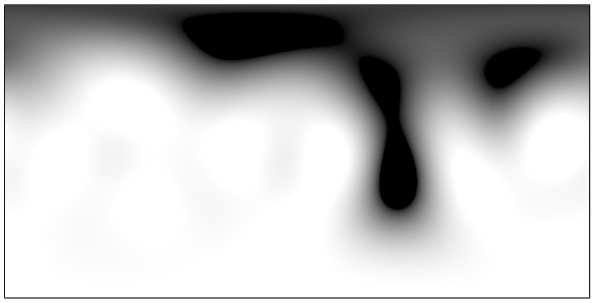


(a) Original.

(b) Ambient Cube (12 bytes).



(c) Linear Gaussians, 49 terms (98 bytes).



(d) Spherical Gaussians, degree 6 (98 bytes).

Figure 4.7: Comparison of reconstructions of a self-occlusion signal.

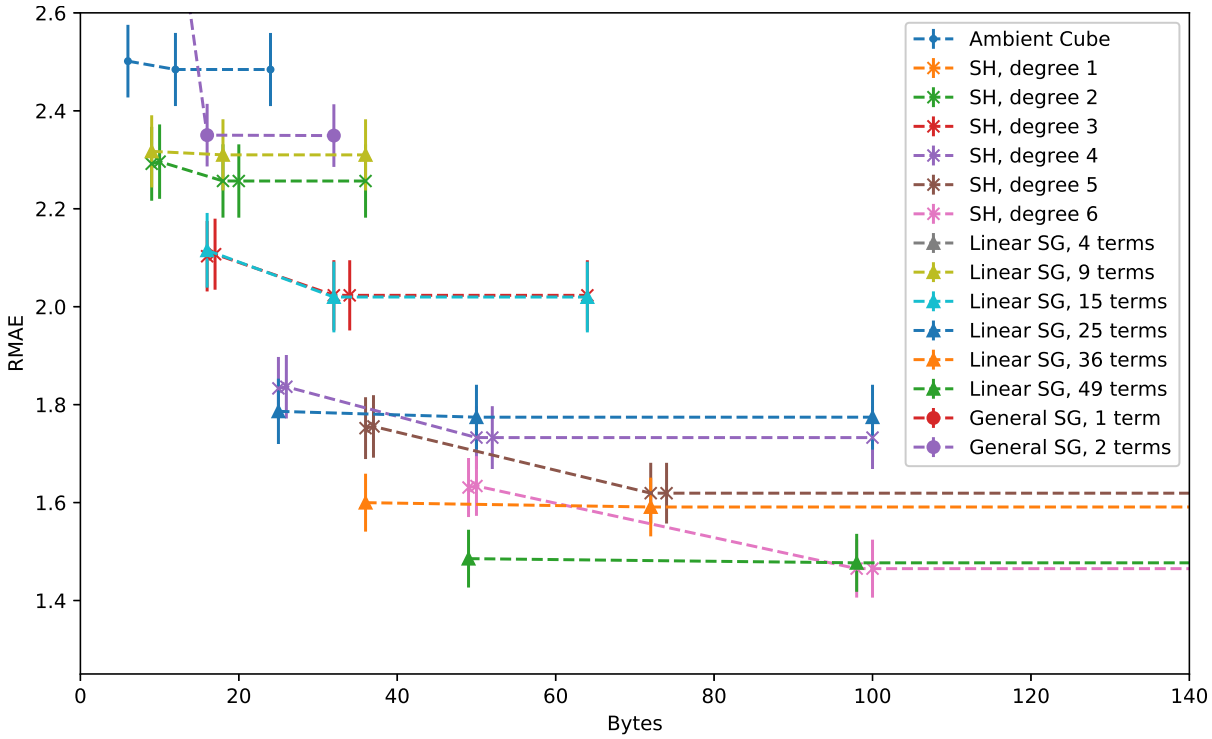


Figure 4.8: Relative mean absolute error versus space requirements for self-occlusion signals. Each model has been given a distinct colour. Each model family has been assigned a distinct symbol (cross, triangle, disc). Vertical line segments indicate relative standard deviations. SH = Spherical Harmonics, SG = Spherical Gaussians.

For self-occlusion signals linear spherical Gaussians consistently outperform spherical harmonics.

From the plot I furthermore deduce that 16 bit is enough to represent self-occlusion coefficients.

For the sake of brevity, I did not include the plot for the SSIM self-occlusion errors. In general the SSIM measurement data supported the conclusions derived from the RMAE metric (figure 4.8).

Chapter 5

Discussion

In this chapter, I will take a step back and consider the implications of my findings. First I will make a few remarks about the validity of the results from chapter 4, i.e. what is their domain applicability (section 5.1). Next, I will compare my findings for spherical harmonics and spherical Gaussians, and relate this to some of the current trends in video games (section 5.2). Finally, I will suggest three interesting research projects for the future (sections 5.3, 5.4, and 5.5).

5.1 Validity of Results

My measurements are ultimately derived only from my particular set of test signals (section 3.1). For this reason, my results (chapter 4) are not guaranteed to apply in general and therefore they should not be treated as such. To produce equivalent results that do apply in general, it would have been necessary to redo the evaluation using a much larger set of test signals.

This being said, the irradiance and radiance test signals that I used did have some variety to them (as detailed in section 3.1). Therefore I find it likely that my results concerning these signal types do apply in some general cases. However, my set of self-occlusion test signals was particularly small. This means that the credence we assign to these results should be correspondingly low.

At the very least, my results apply for all input signals that are sufficiently similar to the test signals that I used for my measurements. Generally, my results can be considered a decent starting point when facing the choice of which spherical basis to use to encode a given set of signals.

5.2 Current Trends

While spherical harmonics are widely adopted within the video game industry, spherical Gaussians are used less frequently. It is well-known that spherical harmonics of 2nd degree is a suitable basis to encode irradiance signals [RH01]. Therefore, it was not surprising that my measurements showed exactly this. Interestingly though, the linear spherical

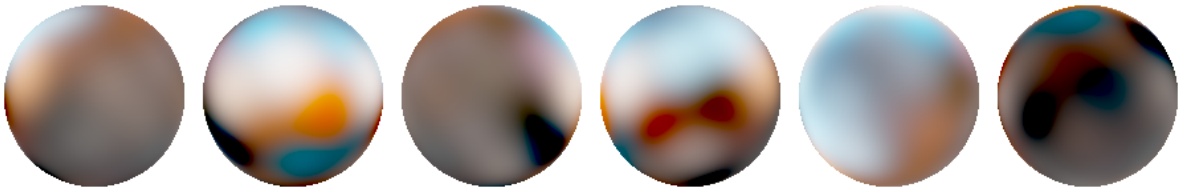
Gaussian model also did well for irradiance, almost as good as spherical harmonics. For both radiance and self-occlusion however, linear spherical Gaussians turned out to consistently outperform spherical harmonics. Consider that the spherical Gaussian models used in this thesis achieved these results even with a less-than-optimal heuristic for its sharpness parameter, λ (section 3.2.4). This suggests that they may be able to perform even better with minor changes (see section 5.3). Therefore it seems likely that video games may benefit from considering spherical Gaussians instead of spherical harmonics in some cases.

Irradiance has been the most common quantity to store in light probes. However, in recent years we have seen new types of data being stored in probes [IS17b] [Ste16], e.g. self-occlusion and precomputed radiance transfer [SKS02]. As new ways of using light probes emerge it is important to continuously consider which bases to use. My results suggest that spherical Gaussians ought to be included in such considerations.

5.3 Gaussian Sharpness Parameter

An interesting aspect of spherical Gaussians compared to spherical harmonics is that they are inherently more flexible, i.e. they have more parameters to tweak. In this thesis, I used a simple heuristic for the sharpness parameter λ (section 3.2.4). For future research, it would be interesting to find better ways to treat this parameter. Figure 5.1 gives an idea of the potential improvements. By manually tweaking the sharpness parameter λ for 49-term spherical Gaussian, I was able to reduce the RMAE (section 3.4.1) of a radiance probe reconstruction by 13%.

(a) λ set via heuristic (section 3.2.4). RMAE = 0.696.



(b) λ manually set. RMAE = 0.603.

Figure 5.1: Spherical Gaussian reconstructions with differing sharpness parameter λ.

5.4 Block/Cube Compression

In this thesis, I have focused on how light changes as a function of direction. But the light environment at a particular point in space also changes as a function of position. For

some signal types this rate of change is typically relatively small. It would be interesting to research whether we can exploit this fact by encoding several nearby light probes together as a single entity. This is analogous to how JPG and similar image formats work on blocks of pixels rather individual pixels. Instead of 2D blocks of pixels, we could imagine 3D cubes containing uniformly distributed light probes.

5.5 More Bases

There are many known ways to represent light probes (and spherical functions in general) and I have not been able to include them all. For future research, it would be interesting to compare my results with similar evaluations of bases/models such as:

- Spherical Gaussians with fixed directions, but varying sharpnesses and amplitudes.
- Ambient Dice [IS17a].
- Spherical Wavelets [SS95].
- Low-resolution cubemaps with texture compression.

Chapter 6

Conclusion

In this thesis, I presented the theory of spherical function bases (chapter 2) and used this to construct an evaluation method that quantifies reconstruction quality (chapter 3). I focused particularly on signals related to lighting in 3D graphics: irradiance, radiance, and self-occlusion (section 3.1). I wrote an implementation of this evaluation method and used it to test the properties of a variety of spherical models (section 3.2). These evaluation tests generated a set of measurement data from which I deduced a handful of key findings (chapter 4). Finally, I put these findings into perspective and discussed how we may improve the models further in the future (chapter 5).

The key findings were:

- Linear spherical Gaussian models consistently outperform spherical harmonics models in representing radiance and self-occlusion signals.
- 16 bit of floating point precision is often enough to encode coefficients for irradiance, radiance, and self-occlusion.
- You can use 8 bit floating point precision to encode coefficients, but it reduces quality significantly. 8 bit precision tend to overflow when it is used to encode radiance with linear spherical Gaussian models.
- For irradiance signals, spherical harmonics degree 2 and 9-term linear spherical Gaussians often represent a suitable trade-off in quality versus space requirements. Additional terms/coefficients lead to rapidly diminishing returns.

As detailed in section 5.1 these results are not guaranteed to apply generally but can be considered a decent starting point when considering a choice of basis.

In addition to the above list of findings, this thesis can serve as an approachable introduction to spherical function bases for the uninitiated reader with interest in this subject.

Bibliography

[AMHH18] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-time rendering*. AK Peters/CRC Press, 2018.

[Art15] Mary K Arthur. Point picking and distributing on the disc and sphere. Technical report, ARMY RESEARCH LAB ABERDEEN PROVING GROUND MD WEAPONS AND MATERIALS RESEARCH, 2015.

[Bru12] Dominique Brunet. A study of the structural similarity image quality measure with applications to image processing. Technical report, University of Waterloo, 2012.

[Cro] Sébastien Crozet. Nalgebra - linear algebra library. https://www.nalgebra.org/. Online; accessed 8-Jan-2019.

[Cup12] Robert Cupisz. Light probe interpolation using tetrahedral tessellations. In *Game Developers Conference (GDC)*, 2012.

[fCT] USC Institute for Creative Technologies. High-resolution light probe image gallery. http://gl.ict.usc.edu/Data/HighResProbes/.

[Gre03] Robin Green. Spherical harmonic lighting: The gritty details. In *Archives of the Game Developers Conference*, volume 56, page 4, 2003.

[IS17a] Michał Iwanicki and Peter-Pike Sloan. Ambient dice. In *Eurographics Symposium on Rendering-Experimental Ideas & Implementations*, 2017.

[IS17b] Michał Iwanicki and Peter-Pike Sloan. Precomputed lighting in call of duty: Infinite warfare. *SIGGRAPH 2017 Course: Advances in Real-Time Rendering in Games*, 2017.

[Lar] Greg Ward Larson. Radiance file formats. http://radsite.lbl.gov/radiance/refer/filefmts.pdf.

[Les] Kornel Lesiński. Dssim. https://github.com/kornelski/dssim. Online; accessed 8-Jan-2019.

[Lon] Kathryn Long. half. https://github.com/starkat99/half-rs. Online; accessed 8-Jan-2019.

[MMG06] Jason Mitchell, Gary McTaggart, and Chris Green. Shading in valve's source engine. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, pages 129–142, New York, NY, USA, 2006. ACM.

[N+] Sven Nilsen et al. image. https://github.com/PistonDevelopers/image. Online; accessed 8-Jan-2019.

[ON] Yuriy O'Donnell and David Neubelt. Probulator. https://github.com/kayru/Probulator. Online; accessed 2-Jan-2018.

[Pet16] Matt Pettineo. Lightmap baking and spherical gaussians. mynameismjp.wordpress.com, 2016.

[PHW10] Matt Pharr, Greg Humphreys, and Jakob Wenzel. *Physically Based Rendering: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., 3rd edition, 2010.

[RH01] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 497–500. ACM, 2001.

[Rus] Rust Team. Rust progamming language. https://www.rust-lang.org/. Online; accessed 11-Jan-2019.

[Sil] Silicon Studio. Enlighten - real time global illumination solution. https://www.siliconstudio.co.jp/middleware/enlighten/en/. Online; accessed 2-Jan-2019.

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 527–536. ACM, 2002.

[Slo08] Peter-Pike Sloan. Stupid spherical harmonics (sh) tricks. In *Game developers conference*, volume 9. Citeseer, 2008.

[SS95] Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 161–172. ACM, 1995.

[Ste16] N Stefanov. Global illumination in tom clancy's the division. In *Game Developers Conference*, 2016.

[TS06] Yu-Ting Tsai and Zen-Chung Shih. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Trans. Graph.*, 25(3):967–976, July 2006.

[Unia] Unity Technologies. Unity asset store - the best assets for game making. https://assetstore.unity.com/. Online; accessed 10-Jan-2019.

[Unib] Unity Technologies. Unity (game engine). https://unity3d.com/. Online; accessed 10-Jan-2019.

[Vog10] Bernhard Vogl. Lightprobes. http://dativ.at/lightprobes/, 2010.

[WBSS04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[WGS+07] Jiaping Wang, Minmin Gong, John Snyder, Baining Guo, and Peiran Ren. All-frequency rendering of dynamic, spatially-varying reflectance. *ACM Transactions on Graphics*, January 2007.

[Wik18] Wikipedia contributors. Spherical harmonics — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Spherical_harmonics, 2018. Online; accessed 1-Oct-2018.

[Wik19] Wikipedia contributors. Signal — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Signal, 2019. Online; accessed 10-Jan-2019.

Appendix A

Framework Compilation and Usage

This thesis was submitted with a companion evaluation framework called `sigeva`, introduced in section 3.5. It is implemented using version 1.29.1 of the Rust [Rus] programming language. The application depends on the following Rust crates:

- `nalgebra` [Cro]: A linear algebra library.
- `image` [N+]: An image encoding and decoding library.
- `half` [Lon]: A library for encoding and decoding 16 bit floating point numbers.
- `dssim` [Les]: A library for calculating structural similarity index from two images.

The framework can be compiled using Rust's package manager, `cargo`, using the command `cargo build`. This will download all dependencies and produce an executable in `target/debug/sigeva`. For an optimized build you can use `cargo build --release`. The framework includes light probes, which can be found in the `assets` folder.

The application accepts several commands:

- `> sigeva fit=[signal type],[metric]`
  This will fit signals of type [signal type] to all models and evaluate their error with respect to [metric]. The output includes error mean and standard deviation for each model. Possible values for [signal type]: irradiance, radiance, occlusion. Possible values for [metric]: rmae, ssim, rmse.
- `sigeva convert=[cubemap dir],[output file]`
  Convert a cubemap located in [cubemap dir] into equirectangular format written to [output file]. The cubemap dir must hold files with names: PositiveX.png, NegativeX.png, PositiveY.png, NegativeY.png, PositiveZ.png, NegativeZ.png. These files correspond to each face of the cubemap.
- `> sigeva convolve=[in file],[out file]`
  Based on a equirectangular radiance signal stored in [in file], this command calculates a irradiance signal via diffuse convolution, and stores the result in [out file]. Technically, it is irradiance divided by pi that is stored in the output, so you must multiply by pi to get real irradiance values.
- `> sigeva project=[in file],[out file]`
  Performs a sphere projection of [in file] for each of the six axial directions, and write

the results side by side into [out file]. [in file] must be in equirectangular format.

- `> sigeva help`
  Prints out usage instructions.

For example, `sigeva fit=irradiance,rmae` will fit the built-in irradiance probes to all models, and calculate their RMAE (section 3.4.1).

The framework has a dozen unit tests which can be executed using `cargo test`.