FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Realistic Rendering in Mobile Augmented Reality

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media Informatics and Visual Computing

by

Johannes Unterguggenberger, BSc
Registration Number 0721639

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Assoc. Prof. Dr. Hannes Kaufmann
Assistance: Mag. Dr. Peter Kán

Vienna, 5th October, 2016

Johannes Unterguggenberger   Hannes Kaufmann

Acknowledgements

First and foremost, I have to thank Peter Kán for his extraordinary support, motivation, interesting discussions, and feedback. It was a pleasure working with him on the project and on the thesis. Many thanks go to Hannes Kaufmann for being supervisor of this thesis.

I am very grateful to my parents for enabling me my course of education which finally led me to this point.

I would like to thank my beloved girlfriend Theresa Wallner, my sister and her family: Katja, Markus, and my enchanting nieces Carla and Romana, and the good friends in my life. Furthermore, I would like to thank so many amazing people who share their knowledge, experience, and findings on the internet.

A big thank you goes to Ravi Ramamoorthi for answering a bunch of questions about spherical harmonics. I am thankful to Manuel Thomas Schmidt and Gabriel Größbacher for providing thesis feedback. Special thanks go out to fellow student and main companion during my time on Vienna University of Technology Michael Probst, who supported me during this thesis by creating several 3D models and light maps, and by profound discussions.

Kurzfassung

Augmented Reality (AR)-Applikationen kombinieren eine Sicht auf eine Echtweltumgebung mit computergenerierten Objekten in Echtzeit. Je nach AR-Applikation ist es wünschenswert, die visuelle Kohärenz der computergenerierten Bilder zu den Bildern aus der echten Welt zu maximieren. Um dies zu erreichen, müssen die virtuellen Objekte so realistisch wie möglich gerendert werden. In dieser Diplomarbeit wird eine bildbasierte Beleuchtungsmethode (image-based lighting (IBL)) präsentiert, um virtuelle Objekte auf mobilen Geräten mit Beleuchtungsinformationen aus der echten Welt realistisch zu rendern.

Die präsentierte Technik nutzt im ersten Schritt die Kamera und die Bewegungssensoren eines mobilen Gerätes, um ein omnidirektionales Bild der Umgebung mit hohem Dynamikumfang (high dynamic range (HDR)) aufzunehmen und in einer *Environment-Map* zu speichern. Im zweiten Schritt wird diese aufgenommene Environment-Map für das Rendering mit unterschiedlichen Materialien vorbereitet, indem mehrere Varianten der ursprünglichen Environment-Map berechnet werden, die je nach Material zum Rendern ausgewählt werden. Die Map, welche diffuse Beleuchtungsinformationen beinhaltet, heißt *Irradiance-Map* und die Maps, die spiegelnde oder glänzende Beleuchtungsinformationen beinhalten, werden *Reflection-Maps* genannt. Die Berechnung dieser Maps entspricht einer gewichteten Faltung nach einem bestimmten Beleuchtungsmodell, wobei die korrekte Menge an einfallendem Licht aus allen Richtungen in die Berechnungen miteinbezogen wird. Wie diese Berechnungen effizient auf mobilen Geräten durchgeführt werden können ist der Hauptbeitrag dieser Diplomarbeit. Es werden mehrere Herangehensweisen zur Durchführung dieser Berechnungen präsentiert, deren Eigenschaften, Resultate, Stärken und Schwächen werden analysiert sowie Optimierungen beschrieben.

Wir beschreiben drei verschiedene Methoden zur Berechnung der Irradiance- und Reflection-Maps und beschreiben diese detailiert: Die akkurate Berechnung, eine MIP-Mapping basierte Approximation, sowie eine Berechnung im Frequenzbereich basierend auf Spherical Harmonics (SH). Wir beschreiben detailiert die Implementierungen, präsentieren Analysen und diskutieren jede dieser drei Methoden hinsichtlich ihrer Eigenschaften und Beschränkungen im Hinblick auf mobile Geräte. Des Weiteren wird beschrieben, wie die berechneten Maps in IBL-Rendering genutzt und mit gängigen Echtzeit-Renderingeffekten kombiniert werden können, um eine hohe visuelle Kohärenz von virtuellen Objekten in AR-Szenen zu erreichen.

Das wichtigste Novum dieser Diplomarbeit ist der Fokus auf die Möglichkeiten von mobilen Geräten und die Eigenschaft, dass alle Schritte auf einem einzigen handelsüblichen mobilen Gerät durchgeführt werden können: vom Aufnehmen der Environment-Map an einem bestimmten Punkt im Raum, über die Berechnung der Irradiance- und Reflection-Maps, bis hin zum Rendern der virtuellen Objekte mit den berechneten Maps in einer AR-Szene.

Abstract

Augmented Reality (AR) applications combine a view of a physical, real-world environment with computer-generated objects and effects in real-time. Depending on the application, it is desirable to maximize the visual coherence of the virtual objects compared to the real-world image. To achieve this goal, virtual objects have to be rendered as realistically as possible. This thesis presents an image-based lighting (IBL) technique for realistic rendering of virtual objects on mobile devices which uses lighting information from the real-world environment.

In the first step, the presented technique uses a mobile device's camera and motion sensors to capture an omni-directional image of the surrounding in high dynamic range (HDR) and stores it in an *environment map*. In the second step, the captured environment map is prepared for rendering with different materials by calculating a set of maps. During rendering, the most suitable of these maps are selected for each material and used for shading a virtual object with the specific material. The map which contains diffuse illumination information is called *irradiance map*, and the maps which contain glossy or specular illumination information are called *reflection maps*. The calculation of the maps corresponds to a weighted convolution. The weighting is determined by a reflection model which takes the correct amount of incident lighting from all directions into account. How these calculations can be performed efficiently on mobile devices is the main focus of this thesis. Multiple approaches to perform the calculations are described. Their properties, results, strengths and weaknesses are analyzed and optimizations are proposed.

We describe three different approaches for the calculation of irradiance and reflection maps in this thesis: the accurate calculation, a MIP-mapping based approximation method, and calculation via spherical harmonics (SH) frequency space. We provide detailed implementation instructions, analyses, and discussions for each of these approaches with regard to the properties and limitations of mobile devices. Furthermore, we describe how the calculated maps can be used with IBL rendering and be combined with established rendering techniques to achieve a high degree of visual coherence of virtual objects in AR scenes.

The main novelty of this thesis is its focus on the capabilities of mobile devices. We describe how to do all steps on a single commodity mobile device: From capturing the environment at a certain point in space, to calculating the irradiance and reflection maps, and finally rendering virtual objects using the calculated maps in an AR scene.

Contents

CHAPTER 1
Introduction

Augmented Reality is a technology which allows to combine virtual objects with views of the real world. Some types of applications work either better or only as mobile applications and some require a strong visual coherence between virtual and real objects. In order to render virtual objects in a visual coherent way, the real world's illumination information has to be captured and used for shading them.

1.1 Motivation

Mobile devices enable new kinds of Augmented Reality (AR) applications since the user is free to move around and is not bound to a stationary computer. There are various different areas of application for mobile AR, some of which benefit from or require that the virtual objects are rendered as realistically as possible.

Rendering virtual objects realistically in an AR scene means making it hard for an observer to distinguish between virtual and real objects in the rendered image. The **visual coherence** of virtual and real objects is to be maximized to achieve this goal. A high visual coherence can be essential for some applications and it can increase the user's immersion.

While performance of mobile devices is increasing by a huge amount every year, they are still limited in terms of computation power compared with stationary computers. While the latter are capable of running physically based rendering techniques [PH10] in real-time, those techniques are still too demanding for mobile hardware. The challenge is to develop a technique which is able to capture real world illumination and use it to render virtual objects in a visually coherent way while maintaining real-time frame rates during rendering on a mobile device.

In many AR rendering techniques, special cameras with fish-eye lenses are used to capture the full sphere of light from a specific position in the real world. On a mobile device, it is desirable to not have any dependencies on additional hardware. While there are no commodity mobile devices with integrated fish-eye cameras, cameras and motion sensors have become standard in consumer hardware. Most current mobile phones and tablets feature accelerometers, which measure the acceleration of the device, and gyroscopes, which measure device orientation. Using these sensors, camera images from different directions can be accumulated to capture the full sphere of light.

The image of the full sphere of light can be stored as an environment map and can be further processed into irradiance maps and reflection maps which contain the illumination information for specific types of materials with varying degrees of glossiness. Depending on the method for calculating these irradiance and reflection maps, an intermitting compute step can be required. The general rule of thumb is that more elaborate methods potentially lead to an increased quality and thus help to increase the visual coherence.

Our technique is targeted to AR applications on mobile devices which require or benefit from realistic rendering. This can apply to applications in the areas of entertainment, education, gaming, architecture, commerce, tourism, sightseeing, industrial design, and others.

1.2 Approach

In this thesis, we present a technique to capture real world illumination and use it for rendering virtual objects. We developed a mobile application that consists of two distinct parts: The first part performs environment map capturing using only a single mobile device as described by Kán in [Kán15]. The second part is the rendering which uses the environment map from the first part to generate irradiance and reflection maps and uses those for rendering virtual objects in an AR scene in a visually coherent way.

Calculating the irradiance and reflection maps can be an expensive and time-consuming task and various methods to calculate them exist. In order to calculate the reflected outgoing radiance, an environment map can be sampled by rays, originating from the surface point. Depending on the surface material, many incident radiance directions contribute to the reflected outgoing radiance. In the case of a perfectly diffuse material, all directions in the hemisphere around the surface normal have to be taken into account. In order to achieve real-time rendering performance on mobile devices, the environment map has to be pre-processed by bidirectional reflectance distribution function (BRDF) convolution and its result stored in another map. We call the maps which store diffuse illumination information **irradiance maps**, and those which store specular and glossy illumination information **reflection maps**.

The different methods, how to calculate irradiance and reflection maps, and their results are the main part of this thesis. It can be approached in different ways, ranging from creating an accurate solution which requires a huge amount of computation and does not run in real-time, to approximating techniques which operate in real-time. Regardless of the particular approach, the aim is to do all steps on the same mobile device: from capturing the envrionment map to calculating the maps and rendering of the virtual objects. There shall be no dependency on any other devices or systems.

In order to render virtual objects realistically, they have to be illuminated in a way that matches the lighting of the real objects as closely as possible. In this thesis, we propose to use irradiance and reflection maps, generated from captured high dynamic range (HDR) environment maps to render virtual objects in an AR application which runs in real-time on current mobile consumer hardware. We analyze different methods for creating the irradiance and reflection maps and describe algorithms for their calculation.

In the envrionment map capturing step, the users can capture an HDR environment map of the real world from their current position. The environment is captured using the mobile device's integrated camera and inertial measurement unit (IMU). Multiple images of the surrounding environment are captured at different device orientations and are then stitched together to produce a HDR sphere map in the angle map projection format like described by Debevec [Deb98]. This environment map serves as input to the second step - the rendering - which processes it, calculates irradiance and reflection maps and renders the virtual objects in an AR scene. For AR tracking, a state of the art image-based mobile AR tracking framework is used. An image target has to be placed in the real world which serves as the reference for rendered objects, defining the coordinate origin and the scale of the scene.

The application provides options to calculate the irradiance and reflection maps with all our three methods. Depending on the method, the calculation step can take a long time, but rendering the AR scene using the calculated irradiance and reflection maps works in real-time. We compare the different methods in terms of computation time, quality, and implementation complexity.

To increase realism further, some additional well known graphics techniques are joined with the proposed IBL scheme, namely normal mapping, and light mapping. These techniques improve the realism of the rendered results and thus increase the visual coherence. Furthermore, we use tone mapping to transform a high dynamic range image into a low dynamic range image suitable for being displayed on a mobile device's display.

For the accurate calculation of irradiance or reflection maps, King points out the high performance demands [Kin05]. He states that it is not possible at all to compute them in real-time. This thesis elaborates on the possibilities of current mobile hardware in that regard and proposes a specific workflow for creating accurate maps. Furthermore, an alternative approach for calculating high-quality maps in some cases is examined: Using a mathematical tool called spherical harmonics [Mac28], an environment map can be transformed into frequency space. Frequency space has some properties that can help greatly to speed up the computations for generating the irradiance and reflection maps, but it comes with a quality trade-off. The third method we are analyzing is based on MIP-mapping. It has an even bigger quality trade-off while performance is dramatically improved.

The main contributions of this thesis are:

- A technique which requires only a single commodity mobile device for capturing real-world Illumination, processing it, and using it in realistic IBL within AR;
- Detailed descriptions and analyses of three different methods for calculating irradiance and reflection maps, and comparison between them regarding quality and performance;
- An algorithm for accurate irradiance and reflection maps calculation, targeted towards highly parallel execution and thus well suited for GPU implementation;
- Adaptions of algorithms for irradiance and reflection maps calculation in spherical harmonics frequency space, targeted towards highly parallel execution and thus well suited for GPU implementation.

CHAPTER 2

Background and Related Work

Irradiance and reflection maps are named after radiometric quantities which describe what is stored in those maps. In this chapter, we describe the most relevant radiometric quantities in the context of our technique, where they appear in the rendering equation, and which reflection model our technique uses for calculating irradiance and reflection maps. Illumination information can be stored in textures or, as an alternative, its frequency space representation can be stored. The latter can be achieved with spherical harmonics (SH).

2.1 Theory of Light Transport

The main part of this thesis is about computing irradiance and reflection maps which store irradiance per normal direction or radiance per reflected view direction, respectively. In our implementation, the incoming radiance is sampled from captured environment maps. This section gives an overview of the underlying theory of light transport by describing the radiometric definitions, the rendering equation, and the bidirectional reflectance distribution function (BRDF).

2.1.1 Radiometry and Photometry

There are two fundamental terms we need to distinguish: **radiometry** and **photometry**. Radiometry is a set of techniques for the measurement of electromagnetic radiation such as visible light, while photometry is the measurement of perceived brightness of light by the human eye. When we are describing our techniques for capturing the environment and for preprocessing the captured high dynamic range (HDR) environment maps, those are radiometric operations. During rendering, there has to be a photometric step as well: In section 5.3.4 we describe how to transform a HDR image into low dynamic range (LDR) with respect to the perceived brightness of the human visual system.

The following radiometric definitions help to accurately describe the physical background of creating irradiance and reflection maps. The definitions are in accordance with [PH10, Han05].

Light is electromagnetic radiation. Only a small range of that electromagnetic radiation, namely a wave length between 400 nanometers and 700 nanometers, lies in the spectrum of visible light.

Radiant energy is the term used for the energy of electromagnetic radiation and thus also the energy of light. It is measured in joule.

Radiant flux (or *radiant power*) is the radiant energy emitted, transmitted, or perceived per unit of time. It is measured in watt.

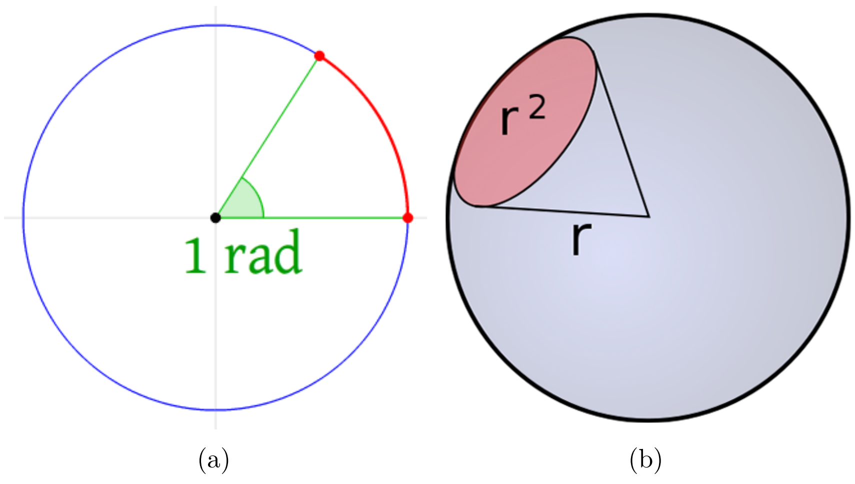


Figure 2.1: Figure (a), adapted from[1], shows how one radian is defined: The arc length is equal to the radius. Figure (b), reprinted from[2], shows that the steradian is defined analogously in 3D space as the ratio between the surface area subtended on a sphere and the square of its distance from the center of the sphere. *One steradian* means that the surface area and the square of the distance are equal.

Radiant intensity is the radiant flux emitted, reflected, transmitted or received per *unit solid angle*. A unit solid angle in 3D space is measured in *steradian*. The steradian is a dimensionless unit because a solid angle is the ratio between the area subtended on a sphere and the square of its distance from the center of the sphere. The analogue to the steradian in 3D space is the radian in 2D space, which is the ratio between the arc length and the radius. Figure 2.1 shows explanatory illustrations for radian and steradian. The radiant intensity is measured in watt per steradian.

[1]Wikimedia Commons, Lucas V. Barbosa, File:Circle radians.gif
`https://commons.wikimedia.org/wiki/File%3ACircle_radians.gif`
[2]Wikimedia Commons, Marcelo Reis, File:Steradian.svg
`https://commons.wikimedia.org/w/index.php?curid=356598`

Irradiance is the radiant flux incident on a surface per unit area. It is measured in watt per square meter. In this thesis, we describe several techniques for generating so-called **irradiance maps** which are named after this radiometric quantity. More specifically, only the maps capturing the diffuse illumination are called irradiance maps, which means that light incident from all directions affects the appearance of every unit area.

Radiosity is the radiant flux leaving a surface per unit area, i.e. the radiant flux emitted, reflected, and transmitted by the surface per unit area. The measurement unit for radiosity is the same as for irradiance: watt per square meter. The radiant flux emitted by a surface is called **radiant exitance** and is consequently also measured in watt per square meter.

Radiance is a radiometric quantity which is especially important for describing the rendering equation. It is the radiant flux per unit solid angle per unit projected area. It is measured in watt per steradian per square meter, because in contrast to the previous two radiometric quantities, irradiance and radiosity, radiance describes radiant flux not only per unit area but additionally per unit solid angle. Radiance can be used to describe radiant flux in various ways like emitted, reflected, transmitted, or received. We use it to describe the properties of different kinds of maps: **environment maps**, which store incident radiance values, and **reflection maps**, which store reflected radiance values (see sections 2.2, and 2.3, respectively). The mathematical symbol used for Radiance is L. Table 2.1 gives an overview of the radiometric quantities, their symbols and units.

Radiometric quantity	**Symbol**	**Unit**	
Radiant energy	Q	$[J]$	joule
Radiant flux	θ	$[W]$	watt
Radiant intensity	I	$\left[\frac{W}{sr}\right]$	watt per steradian
Irradiance	E	$\left[\frac{W}{m^2}\right]$	watt per square meter
Radiosity	J	$\left[\frac{W}{m^2}\right]$	watt per square meter
Radiant exitance	M	$\left[\frac{W}{m^2}\right]$	watt per square meter
Radiance	L	$\left[\frac{W}{sr \cdot m^2}\right]$	watt per steradian per square meter

Table 2.1: Radiometric quantities with their symbols and units

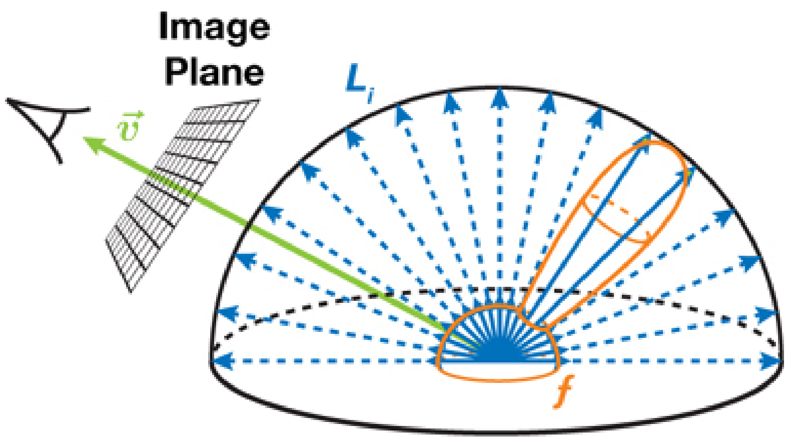


Figure 2.2: This illumination integral illustration, adapted from [CK07], depicts a typical rendering application which calculates the amount of light reflected from the surrounding environment towards a virtual camera at a given surface point for each pixel. In this case, the BRDF f is constructed by combining a diffuse part (the hemisphere in orange) with a glossy part (the reflection lobe in orange) which yields the resulting color for the point where the viewing ray $\vec{v}$ hits the surface.

2.1.2 Rendering Equation

Equation 2.1 shows the rendering equation. It is an integral equation which describes light transport as illustrated in figure 2.2 and has been introduced simultaneously by Kajiya and Immel et al. [Kaj86, ICG86]. It is the mathematical basis of realistic rendering techniques in computer graphics which are attempting to solve or approximate this equation. Depending on the specific technique or the requirements of an application, this equation is often approximated to such a degree that real-time frame rates are feasible. Our application has the requirements to achieve interactive frame rates during irradiance and reflection map calculation, and to achieve real-time frame rates during rendering the AR scene. Furthermore, our technique shall be executed on a mobile device. These requirements mean that we have to apply major approximations to the rendering equation.

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega L_i(x,\vec{\omega}')f_r(x,\vec{\omega},\vec{\omega}')\ \cos\theta\ d\vec{\omega}' \tag{2.1}$$

The rendering equation is composed of several radiance values: outgoing radiance, emitted radiance, and incoming radiance. It describes that the exiting radiance L_o in a certain point x in space towards a certain direction $\vec{\omega}$ is equal to the emitted radiance L_e at that point in the same direction plus the sum of incoming radiance L_i from all directions multiplied by the BRDF of the surface material. The different parts of the rendering equation have the following meanings:

$L_o(x, \vec{\omega})$ is the outgoing radiance from point x towards direction $\vec{\omega}$.
$L_e(x, \vec{\omega})$ is the radiance emitted from point x towards direction $\vec{\omega}$.
$\int_\Omega L_i(x, \vec{\omega}')f_r(x, \vec{\omega}, \vec{\omega}') \cos\theta d\vec{\omega}'$ is the sum of radiance received at point x from all $\vec{\omega}'$ directions.
$L_i(x, \vec{\omega}')$ is the radiance received at point x from direction $\vec{\omega}'$.
$f_r(x, \vec{\omega}, \vec{\omega}')$ is the bidirectional reflectance distribution function which describes the behavior of light reflection on a particular material.
$\cos\theta$ is the light attenuation. It depends on the surface normal at point x.

Our techniques for calculating irradiance and reflection maps simplify the parts of the rendering equation in order to achieve interactive or real-time frame rates, respectively. In chapters 3 and 4 we describe in detail the steps of our techniques and how they relate to the rendering equation.

2.1.3 Bidirectional Reflectance Distribution Function

The bidirectional reflectance distribution function (BRDF) is an essential part of the rendering equation. It describes how light is reflected by a particular material. A BRDF can either be explicitly stored by measuring a real-world materials or it can be approximated by analytic functions, which can be very complex. Depending on the accuracy chosen, they can even include the wave length of light as parameters, but we choose a simpler form that is sufficient for describing our techniques. It is shown in equation 2.2.

$$f_r(x, \vec{\omega}, \vec{\omega}') = \frac{dL_r(x, \vec{\omega})}{L_i(x, \vec{\omega}')\, \cos\theta\, d\omega'} \tag{2.2}$$

$f_r(x, \vec{\omega}, \vec{\omega}')$ is the BRDF defined at a point x for the outgoing drection $\vec{\omega}$ and the incoming direction $-\vec{\omega}'$.
$dL_r(x, \vec{\omega})$ is the differential reflected radiance at point x outgoing to direction $\vec{\omega}$.
$L_i(x, \vec{\omega}')$ is the incoming radiance at point x from direction $-\vec{\omega}'$.
$\cos\theta$ is the cosine of the incident angle between the surface normal and the incoming radiance direction $\vec{\omega}'$.

Some examples for different materials can be observed in figure 2.3 A perfectly diffuse surface reflects the light equally into all directions on the hemisphere (figure 2.3a), perfectly specular surfaces reflect exactly one direction (figure 2.3c), and a glossy surface reflects light around the specular reflection direction. The reflection intensity for a given light direction depends on the angle to the specular reflection direction (figure 2.3b).

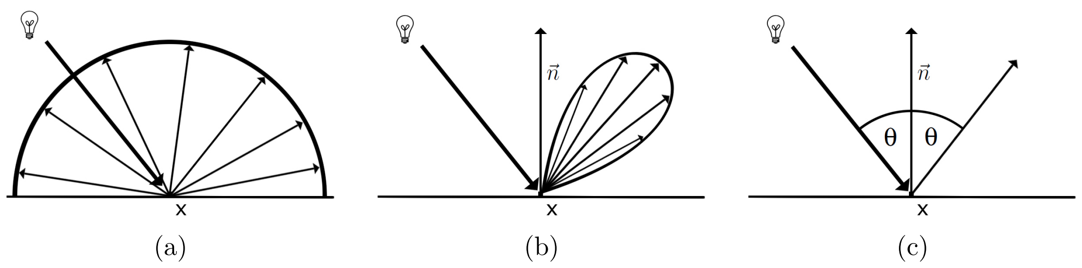


Figure 2.3: These images, adapted from [KK12a], show how light is reflected on a lambertian diffuse surface (a), on a glossy specular surface (b), and on a perfect specular surface (c).

BRDF models can be divided into different groups [AMHH08, Mat03]:

- Physically based BRDFs
- Measured BRDFs
- Empirical BRDFs

Physically based BRDFs are trying to simulate the behavior of a material in a physically correct way. They have to conform to the following laws [Mat03, Kán14]:

- Positivity

$$f_r(x, \vec{\omega}, \vec{\omega}') \geq 0 \tag{2.3}$$

  All values of f_r must be non-negative.

- Helmholtz reciprocity

$$f_r(x, \vec{\omega}, \vec{\omega}') = f_r(x, \vec{\omega}', \vec{\omega}) \tag{2.4}$$

  If incoming and outgoing light directions are interchanged, the BRDF must yield the same result.

- Energy conservation

$$\int_\Omega f_r(x, \vec{\omega}, \vec{\omega}') \cos\theta \, d\vec{\omega} \leq 1, \ \forall x, \forall \vec{\omega}' \tag{2.5}$$

  Equation 2.5 expresses the total hemispherical reflectivity. It means that the energy of light, reflected from the material at position x cannot be higher as the energy of incoming light at position x.

Measured BRDF models are sampled from real-world materials. Matusik et al. [Mat03, MPBM03] measured approximately 100 different materials. Their results can be used in physically based rendering.

Empirical BRDFs are derived from empirical observation of light reflection on materials but do not have a physical basis. While empirical BRDFs are often trying match a physical model, their formulae are not based on actual physical events. Ease of use and intuitive parameters are more important than physical correctness [AMHH08, Kán14]. Examples include the Ward BRDF [War92] and the very well-known and widely used Phong BRDF model [Pho75]. The Phong BRDF model is very intuitive, computationally inexpensive, is able to model a wide variety of materials and has thus become one of the most used BRDF models for real-time applications. There are various versions of the Phong BRDF model some of which are targeted towards being physically plausible [Pho75, LW94, Lew94, AMHH08]. We use the version of the Phong BRDF model shown in equation 2.6 for calculating the irradiance and reflection maps in our technique,

$$f_r(x, \vec{\omega}, \vec{\omega}') = k_d + k_s \cdot \cos^n \alpha \tag{2.6}$$

where k_d is the diffuse reflectivity coefficient representing the fraction of light which is reflected diffusely, k_s is the specular reflectivity coefficient representing the fraction of light which is reflected specularly, n is the specular exponent representing the shininess of the surface, α is the angle between the perfect specular reflection direction $\vec{\omega}_r$ and the outgoing direction $\vec{\omega}$ clamped to a maximum value of $\pi/2$ in order to prevent negative cosine values. The perfect specular reflection direction $\vec{\omega}_r$ is described by equation 2.7

$$\vec{\omega}_r = 2(\vec{\omega}' \cdot \vec{n})\vec{n} - \vec{\omega}' \tag{2.7}$$

where $\vec{\omega}$ is the incoming direction and n is the surface normal at point x. The condition given by equation 2.8 has to be met to satisfy the conservation of energy.

$$k_d + k_s \leq 1 \tag{2.8}$$

2.2 Environment Mapping

Environment mapping [BN76] is an image-based lighting (IBL) technique which approximates the appearance of reflective surfaces. Depending on the setup, results produced with environment mapping can be similar to those generated with physically based rendering techniques like ray tracing, but it depends on the scene and the viewer position. In contrast to physically based methods, environment mapping is very cheap in terms of computational costs. The technique involves capturing an omni-directional representation of real-world light information as an image. Depending on the respective environment mapping technique, the light information is stored in one or multiple textures which are used to retrieve lighting information for shading 3D objects.

Debevec [Deb05] describes the basic IBL steps like follows:

1. capturing real-world illumination as an omni-directional, high dynamic range image;
2. mapping the illumination onto a representation of the environment;
3. placing the 3D object inside the environment; and
4. simulating the light from the environment illuminating the computer graphics object. [Deb05]

An environment map stores an omni-directional image of the surrounding from one point in space in one or more textures. During rasterization, this texture or the set of these textures is sampled to get the reflected color for the current texel.

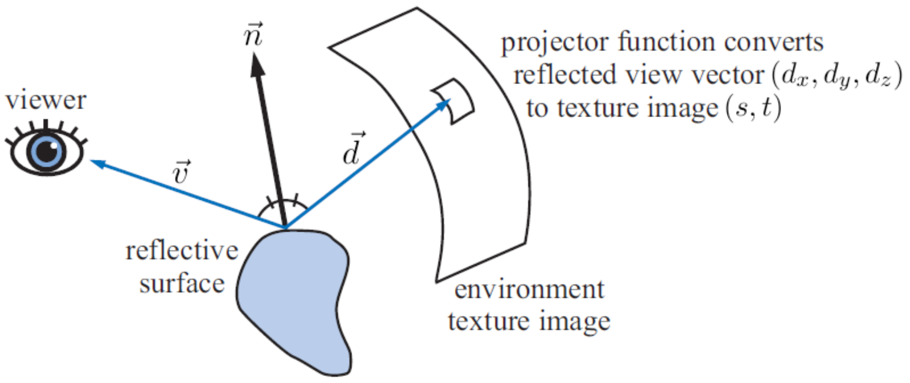

$\vec{n}$
projector function converts reflected view vector (d_x, d_y, d_z) to texture image (s, t)
viewer
$\vec{v}$
$\vec{d}$
reflective surface
environment texture image

Figure 2.4: This illustration, adapted from [AMHH08], depicts the general idea of environment mapping: For a given point at the surface, the viewing ray $\vec{v}$ is reflected around this surface point's normal, yielding the reflected view vector $\vec{d} = (d_x, d_y, d_z)$ which we call the reflected direction. Environment mapping applies a projector function to the reflected direction, converting it to texture coordinates (s, t) which are used to sample the color for shading the surface point from the environment map texture.

Environment mapping has been used extensively in real-time computer graphics applications. It is obvious that the vast majority of 3D games released in the past few years use some kind of environment mapping technique. It is computationally very cheap because environment mapping basically involves only the calculation of the angles of incidence and reflection and a texture lookup to the appropriate texture containing the relevant part of the omni-directional representation of illumination information. Figure 2.4 depicts the general idea of environment mapping. Compared to ray tracing techniques which have to trace a ray against the scene geometry and compute the radiance of the ray, environment mapping is faster by magnitudes and simplifies the workload for the graphics processing unit (GPU). The trade-off with environment mapping is that the reflections are not accurate. While ray tracing produces correct reflections, environment mapping relies on two assumptions which are actually never satisfied: Firstly, the radiance incident upon the illuminated computer graphics object comes from an infinite distance. Secondly, the illuminated computer graphics object is convex, meaning that the object can not reflect itsef. It only reflects the environment [FK03, AMHH08].

To compensate for the first assumption - the infinite distance of the environment - several researchers have developed techniques to make this property of environment mapping less obvious and to achieve better rendering results. Bjorke [Bjo04] points out the limitations of environment mapping especially for small, enclosed environments and presents a technique to approximate local reflections which requires only a small amount of shader math. He proposes to approximate the illuminated object's geometry with a sphere of a certain radius and modifying the texture lookup vector by this approximating geometry. Since a sphere can easily be described algorithmically, the technique of [Bjo04] adds only little computational costs when integrated into a vertex shader or fragment shader.

Another technique by Szirmay-Kalos et al. [SKAL08] which is addressing the first assumption is following a different approach. They point out the fundamental problem of environment mapping: The environment map is the correct representation of an illuminated computer graphics object only at a single point, the reference point of the object - i.e. the origin of the environment map. In their technique, they extend an environment map with distance information of the environment which they suggest to store in the alpha channel of the texture. They call these extended environment maps as *distance impostors*. During rendering, the distance to the environment is used to alter the lookup direction into the environment map.

Popescu et al. [PMDS06] are describing a more advanced environment mapping technique which is based on approximating the geometry of the reflected scene with impostors. Regarding the impostors, they describe two approaches: one is based on billboards, the other on depth maps. They achieve very high-quality results and describe their technique as a middle ground between classical environment mapping and ray tracing.

After environment mapping has been developed by Blinn and Newell in 1976 [BN76], different ways of storing and accessing the omni-directional representation of the environment in one or more textures have been developed. Sphere mapping uses one texture and has been first mentioned by Williams in 1983 [Wil83], and independently developed by

Miller and Hoffman in 1984 [MH84]. Greene introduced cube mapping in 1986 [Gre86]. Heidrich and Seidel introduced paraboloid mapping in 1998 [HS98]. In more recent work further mapping techniques have been introduced: octahedron mapping by Praun and Hoppe in 2003 [PH03], pyramid mapping by Steigleder in 2005 [Ste05], and HEALPix mapping for rendering by Wong et al. in 2006 [WWLL06]. We describe the details of the most common mapping techniques sphere mapping, cube mapping, and dual paraboloid mapping in the next sections.

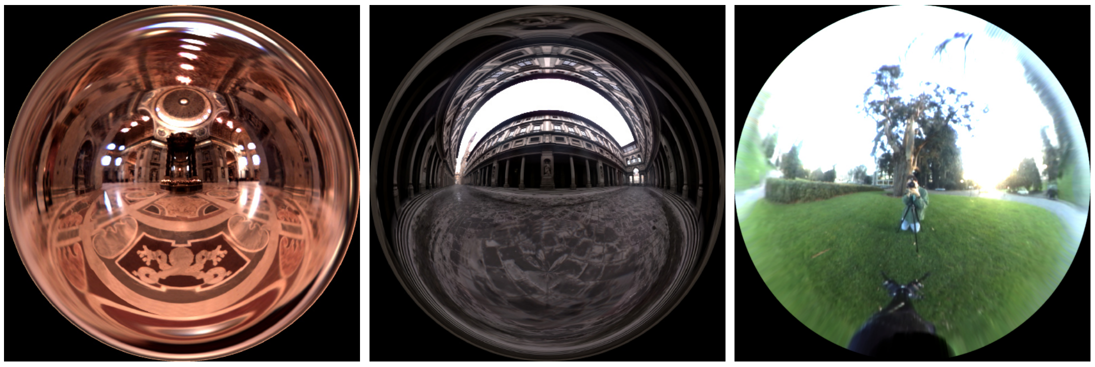

(a) (b) (c)

Figure 2.5: Light probes captured by Paul Debevec, labeled like follows: *St. Peter's Basilica, Rome* (a); *The Uffizi Gallery, Florence* (b); *Campus at Sunset* (c). All images adapted from [Deb01]

2.2.1 Sphere Mapping

Sphere mapping [Wil83, MH84] was the first environment mapping technique supported in general commercial graphics hardware [AMHH08]. It uses one texture to store an omni-directional image of the environment as viewed orthographically (or, in other words, from infinite distance) in a perfectly reflective sphere. See figure 2.6 for an illustration of an orthographic sphere map projection. Real environments can be captured by taking a photograph of a shiny sphere. Debevec [Deb01] has captured several HDR sphere maps in this way. Some of them are shown in figure 2.5.

The orthographic sphere map projection for mapping a direction vector to the sphere map texture is arguably the most common projection type. It is the projection type which is implemented also in OpenGL [SG$^+$09]. It can be activated for OpenGL's automatically generated texture coordinates with the following commands for s and t texture coordinates:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
```

These OpenGL-commands have been more relevant for OpenGL's fixed function pipeline. When using the programmable pipeline, the mapping can be implemented in shaders by using equation 2.9,

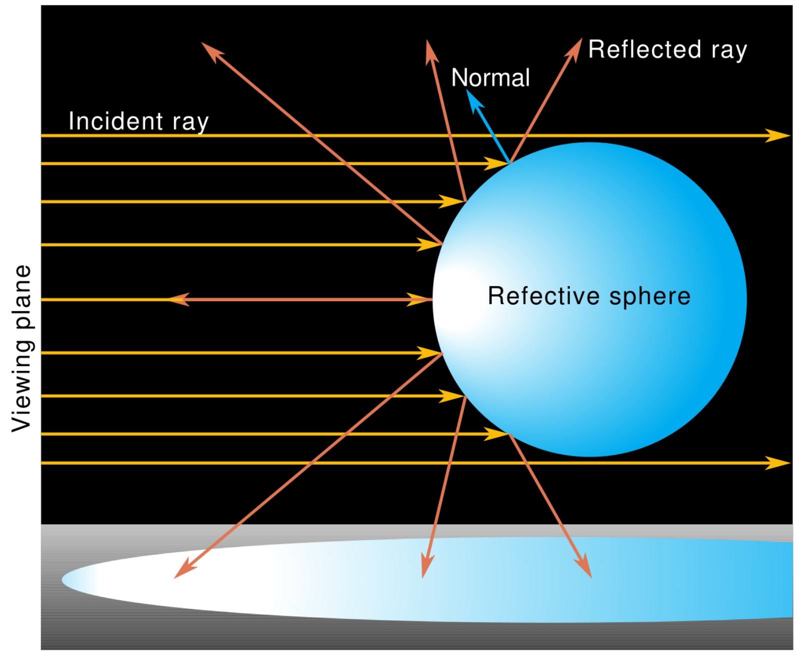


Figure 2.6: Illustration of an orthographic sphere mapping projection; reprinted from [MBGN98]. This can be labeled as the standard way of sphere map projection. It is also implemented in OpenGL (see [SG+09]) and has nice mathematical properties. Orthographic incident viewing rays are reflected on a perfectly reflective sphere and thereby mapped to texture coordinates.

$$(s,t) = \left(\frac{d_x}{2\sqrt{d_x^2 + d_y^2 + (d_z+1)^2}} + 0.5\,,\ \frac{d_y}{2\sqrt{d_x^2 + d_y^2 + (d_z+1)^2}} + 0.5 \right) \tag{2.9}$$

where d is the reflected direction vector, and (s,t) are the resulting texture coordinates in the range $[(0,0),(1,1)]$. How equation 2.9 is derived is described in [AMHH08] and can be explained with figure 2.7. The original view vector $\vec{v} = (0,0,1)$ and the reflected view direction $\vec{d}$ are added together, yielding vector $\vec{l}$, shown in equation 2.10. Normalizing $\vec{l}$ by dividing it through its magnitude yields the unit normal $\vec{n}$, shown in equation 2.11. Since we only need h_x and h_y for describing a point on the image of the sphere, we can ignore the z-coordinate of $\vec{n}$, which finally leads to equation 2.9 for orthographic sphere map projection.

$$\vec{l} = (d_x\,,\ d_y\,,\ d_z + 1) \tag{2.10}$$

$$\vec{n} = \left(\frac{d_x}{|\vec{l}|}\,, \frac{d_y}{|\vec{l}|}\,, \frac{d_z+1}{|\vec{l}|} \right) \tag{2.11}$$

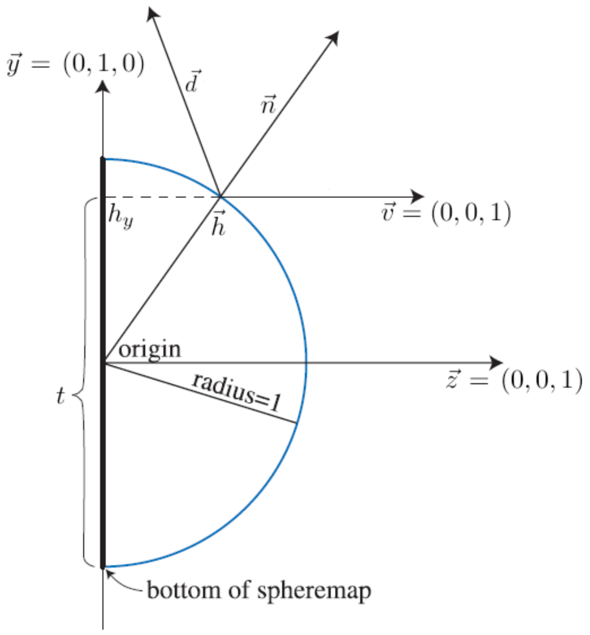


Figure 2.7: This illustration, adapted from [AMHH08], shows that the sphere map's normal $\vec{n}$ is halfway between the constant view direction $\vec{v}$ and the reflected view vector $\vec{d}$ which we call reflected direction. The intersection point $\vec{h}$ has the same coordinates as the unit normal $\vec{n}$. The sphere map's t coordinate can be calculated by finding h_y.

Algorithm 2.1: Calculation of sphere map coordinates of a given direction vector using angle map projection. [AMHH08]

```
1 function sphereMappingOrtho( $\vec{d}$ )
      input  : Normalized direction vector $d = (d_x, d_y, d_z)$
      output : Sphere map texture coordinates $(s, t)$ in range $[0,0]..[1,1]$
2     $m = \sqrt{d_x^2 + d_y^2 + (d_z+1)^2}$;
3     return $(\frac{d_x}{2m} + 0.5, \frac{d_y}{2m} + 0.5)$;
```

Algorithm 2.1 describes how to implement sphere mapping with orthographic projection. This algorithm can be used for a shader to calculate sphere map texture coordinates.

In some cases, other sphere map projections are more suitable than the orthographic projection. Debevec uses a different type of sphere map projection with the sphere maps from his light probe image gallery [Deb01] which is called *angle map projection*. The angle map projection has a different distribution information density like can be observed in figure 2.8. Generally it is desirable to have a uniform distribution of information density across an environment map. However, a sphere map has a singularity located around the edge of the sphere map. Taking texel size into account, a non-uniform distribution which maps fewer directions to the outer regions of a sphere map can have advantages because sampling a specific texel from the outer regions preserves more information and shows less distortion. The formula for angle map projection has been described by Debevec in [Deb01]. Equation 2.12 shows a slightly adapted version of it which maps to texture coordinates in the range $[(0,0),(1,1)]$.

$$(s,t) = \left(d_x \cdot \frac{acos(d_z)}{2\pi\sqrt{d_x^2 + d_y^2}} + 0.5\,,\; d_y \cdot \frac{acos(d_z)}{2\pi\sqrt{d_x^2 + d_y^2}} + 0.5 \right) \tag{2.12}$$

Algorithm 2.2 shows our implementation of sphere mapping using angle map projection. We have included a check in line 4 for preventing *not a number* (`NaN`)-values in the subsequent calculations. This check evaluates to false for directions which would get mapped to the sphere map's singularity. In those situations, a value is returned which lies exactly at the border of the sphere map: the texture coordinates $(0.5, 1.0)$. We have observed that `NaN`-values can lead to unexpected behavior if they emerge in shader calculations. Therefore, it is important to perform this check in GPU shader programs.

Algorithm 2.2: Calculation of sphere map coordinates of a given direction vector using angle map projection. Adapted from [Deb01]

```
1  function sphereMappingAngleMap( $\vec{d}$ )
       input  : Normalized direction vector $\vec{d} = (d_x, d_y, d_z)$
       output : Sphere map texture coordinates $(s,t)$ in range $[(0,0),(1,1)]$
2      $a = \arccos(d_z)$;
3      $b = d_x^2 + d_y^2$;
4      if $a > 0 \land b > 0$ then
5          $r = \frac{1}{\pi} \cdot \frac{a}{\sqrt{b}}$;
6          $s = d_x \cdot r$;
7          $t = d_y \cdot r$;
8          return $(\frac{s}{2} + 0.5, \frac{t}{2} + 0.5)$;
9      else
10         return $(0.5, 1.0)$;
11     end
```

Depending on the use case, in some cases also the reverse mapping is required which calculates the direction vector $\vec{d}$ from given sphere map coordinates (s, t). Our technique requires such reverse mapping for environment maps convolution. Chapter 4 describes its utilization for calculating irradiance and reflection maps. The reverse orthographic mapping can be implemented via algorithm 2.3.

Algorithm 2.3: Calculation of the corresponding direction vector to given sphere map texture coordinates for orthographic projection. [MBGN99]

```
1 function reverseSphereMappingOrtho( $s, t$ )
      input  : Sphere map texture coordinates $(s, t)$ in range $[(0, 0), (1, 1)]$
      output : Direction vector corresponding the input texture coordinates
2     $m = 2\sqrt{-4s^2 + 4s - 1 - 4t^2 + 4t}$;
3     $x = m \cdot (2t - 1)$;
4     $y = m \cdot (2s - 1)$;
5     $z = -8s^2 + 8s - 8t^2 + 8t - 3$;
6     return $(x, y, z)$;
```

Algorithm 2.4: Calculation of the corresponding direction vector to given sphere map texture coordinates for angle map projection. Adapted from [Deb01]

```
1 function reverseSphereMappingAngleMap( $s, t$ )
      input  : Sphere map texture coordinates $(s, t)$ in range $[(0, 0), (1, 1)]$
      output : Direction vector corresponding the input texture coordinates
2     $s' = 2s - 1$;
3     $t' = 2s - 1$;
4     $\phi = \pi\sqrt{s'^2 + t'^2}$;
5     $\theta = \arctan(s', t')$;
6     $\vec{d} = (0, 0, -1)$;
      /* Rotate $\vec{d}$ around the $x$-axis:                    */
7     $\text{rotx}_y = d_y \cdot \cos(\phi) - d_z \cdot \sin(\phi)$;
8     $\text{rotx}_z = d_y \cdot \sin(\phi) + d_z \cdot \cos(\phi)$;
9     $\vec{d} = (d_x, \text{rotx}_y, \text{rotx}_z)$;
      /* Rotate $\vec{d}$ around the $z$-axis:                    */
10    $\text{rotz}_x = d_x \cdot \cos(\theta) - d_y \cdot \sin(\theta)$;
11    $\text{rotz}_y = d_x \cdot \sin(\theta) + d_y \cdot \cos(\theta)$;
12    $\vec{d} = (\text{rotz}_x, \text{rotz}_y, d_z)$;
13    return $\vec{d}$;
```

The algorithm for reverse sphere mapping with the angle map projection is described by Debevec in [Deb01]. Our adaption of it is listed in algorithm 2.4. We adapted it to the coordinate system of our rendering pipeline which uses texture coordinates in the range $[(0, 0), (1, 1)]$ and requires to rotate vector $\vec{d}$ around the x and z-axis.

A major disadvantage of a sphere map is that it is not view-independent. Quality is better around the direction from where the sphere map has been captured. Depending on the view, the singularity around the edge and artifacts at the outer regions can become noticeable.

Sphere maps are often called **light probes**, as they capture the lighting situation at the sphere's location [Deb98]. Especially for storing illumination information in the form of irradiance and reflection maps, sphere maps are well-suited. We are using this format for environment maps in our implementation.

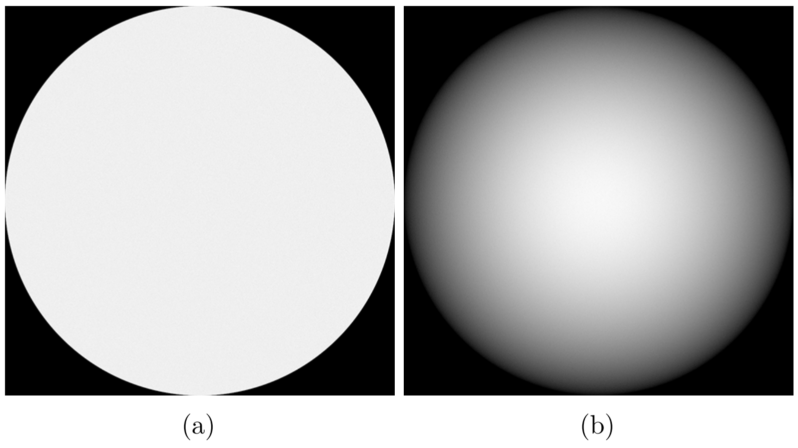


Figure 2.8: These images have been generated by creating 1 billion (10^9) uniformly distributed direction vectors and applying a sphere map projection technique, adding 1 to the corresponding sphere map texture coordinate. Image (a) was created using orthographic projection, while image (b) was created using angle map projection. While for orthographic projection the density of information is uniform, angle map projection maps more directions towards the center of the sphere map texture. The analytic formula to calculate the angle map's information density is $sinc(r \cdot \pi)$ like it can be seen in the implementation of [RH01a] available at[3].

[3]Ramamoorthi and Hanrahan, An Efficient Representation for Irradiance Environment Maps, Source Code, `http://graphics.stanford.edu/papers/envmap`

2.2.2 Cube Mapping

Cube mapping is a different format for capturing and storing environment maps. It has been introduced by Greene in 1986 [Gre86]. Due to its speed and flexibility it is the most popular environment mapping method implemented in modern graphics hardware [AMHH08]. A cube map is created by positioning a camera at the center of the envrionment map and capturing the environment in six different directions (left, right, front, back, up, and down; or in vectors $(1, 0, 0)$, $(-1, 0, 0)$, $(0, 0, 1)$, $(0, 0, -1)$, $(0, 1, 0)$, and $(0, -1, 0)$) with a field of view of 90 degrees, storing the result of each direction in a texture. A cube map can consist of six different textures in total like shown in figure 2.9. OpenGL provides the texture type `GL_TEXTURE_CUBE_MAP` to store a cube map efficiently [SG+09].

A cube map is accessed by the direction vector $\vec{d}$, which is used directly as three-component texture coordinate [AMHH08]. A cube map can be problematic since MIP-mapping is usually applied to each face separately – possibly leading to noticeable artifacts near the borders.

Compared to sphere maps, cube maps are view independent. They have good, although not uniform, sampling characteristics. The sampling characteristics can be improved by modifications to sphere mapping. An example is *isocube* mapping by Wan et al. [WWL07]. Lagarde and Zanuttini presented a technique for parallax-corrected cube maps especially well-suited for image-based lighting [LZ12].

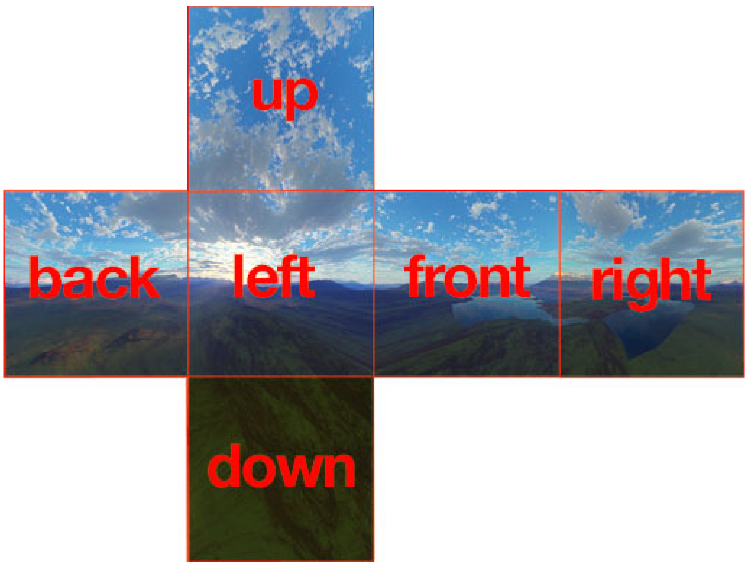


Figure 2.9: An unfolded cube map showing all its six sides facing front, back, left, right, up, and down. Reprinted from[4].

[4]Wikimedia Commons, Arieee, File:Skybox_example.png
`https://commons.wikimedia.org/wiki/File:Skybox_example.png`

2.2.3 Dual Paraboloid Mapping

The third environment mapping technique among the most popular is called *dual paraboloid mapping*. It has some similarities to sphere mapping but instead of using a perfectly reflective sphere, two perfectly reflective paraboloids are used. Each paraboloid creates a circular texture similar to a sphere map. Both of those textures cover an environment hemisphere. The concept of dual paraboloid mapping is shown in figure 2.10. [AMHH08]

Dual paraboloid maps have several good properties. Contrary to sphere maps, there is no singularity, so interpolation can be done between the two reflected view directions. Dual paraboloid maps have more uniform texel sampling than both, cube maps and sphere maps. They are view-independent, like cube maps. [AMHH08]

The main drawback of dual paraboloid maps is their creation. Sphere maps are straightforward to create from real environments. Cube maps are straightforward to create from synthetic scenes. Dual paraboloid maps are neither straightforward to create from real environments nor from synthetic scenes. Parabolid maps have to be created by warping images, by transforming objects in vertex or tessellation shaders accordingly, or by using ray tracing. [AMHH08]

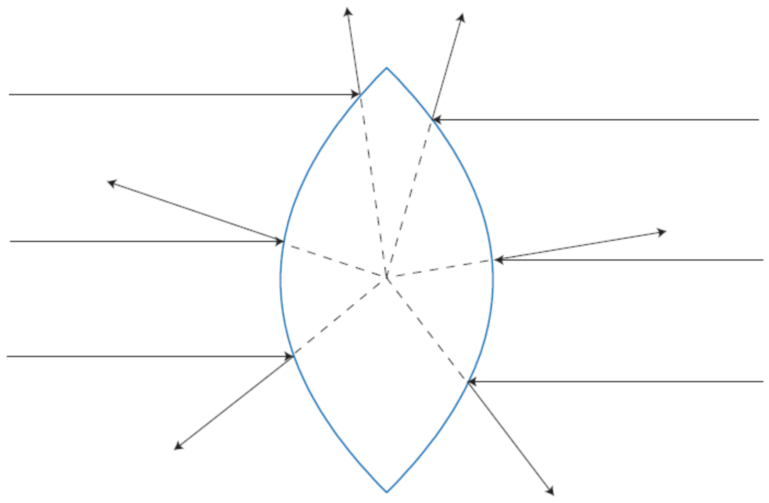


Figure 2.10: Two paraboloids with the properties that each reflected view vector extends to meet at the common focus point of both paraboloids. The paraboloids are perfectly reflective and capture an environment using diametrically opposite views. Reprinted from [AMHH08].

2.3 Irradiance and Reflection Mapping

Environment mapping was intended as a technique for rendering mirror-like surfaces. However, it can be extended to glossy reflections and even to diffuse reflections. As figure 2.11 shows, multiple reflection directions have to be taken into account for glossy surfaces for a specific reflection direction.

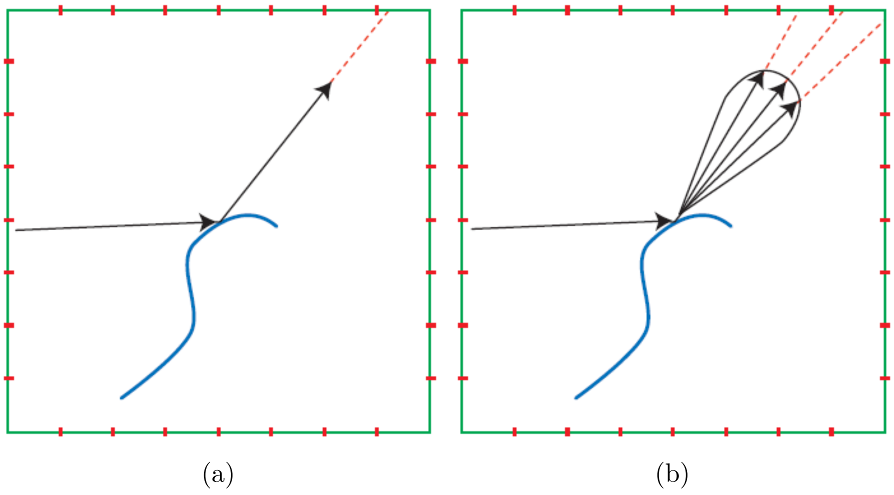


Figure 2.11: Both images show a view direction which is reflected off an object. Image (a) shows a single reflection direction which means that the object will look like a perfect mirror. Image (b) shows a specular lobe which means that multiple directions are sampled from the environment map, weighted by the BRDF, and added together for the final pixel color. The object would appear glossy, but not like a mirror as the object in (a) does. Adapted from [AMHH08].

A so-called **reflection map** is an environment map which stores the pre-calculated glossy reflection for each possible reflection direction. That means, in order to create a reflection map, the specular lobe for each possible reflection direction has to be evaluated and the resulting value is stored in the reflection map texture. The same idea can be extended to diffuse surfaces. In figure 2.12 it is illustrated that for a diffuse surface all directions over the hemisphere around a direction vector contribute to the final color, weighted by their cosine to the direction vector. A map storing the diffuse lighting for each possible direction like shown in figure 2.12 is called **irradiance map** [RH01a].

Recalling the radiometric definitions from section 2.1.1, we can easily establish the obvious connections to the radiometric quantities *radiance* and *irradiance*. Environment maps store incoming radiance values, reflection maps store outgoing radiance values, and irradiance maps store irradiance values. Reflection maps are accessed with the reflected view direction while irradiance maps are accessed with the surface normal direction.

Different approaches can be taken to create irradiance and reflection maps. Since the human eye is quite forgiving, the environment map texture can simply be blurred to

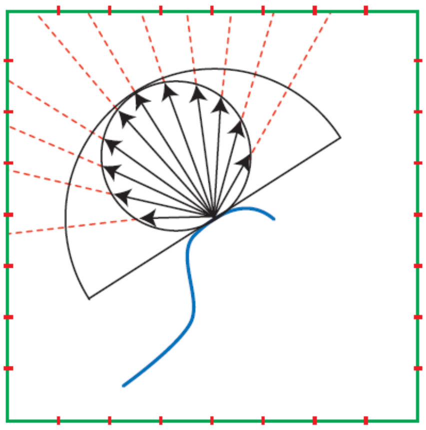


Figure 2.12: In order to calculate an irradiance map, the entire hemisphere around a normal direction has to be sampled. The sampling directions are weighted by their cosine to the normal's direction. Reprinted from [AMHH08].

achieve an approximation of the desired effect [AMHH08]. Using a BRDF lobe to calculate the convolution, such as a cosine power or a Gaussian lobe, leads to more realistic results [MH84]. Heidrich and Seidel discuss methods for calculating physically based reflection maps [HS99]. Several techniques [GKD07, MLH02, McA04] propose storing environment maps filtered with cosine or Gaussian lobes of different widths in the MIP-map levels of a single texture where blurrier maps are stored in lower-resolution MIP-map levels.

As Akenine-Möller et al. [AMHH08] point out, reflection maps are only valid if all viewing and surface directions have the same shape of specular lobes. In the case depicted in figure 2.13, a part of the lobe would have to be chopped off in order to yield perfectly realistic results.

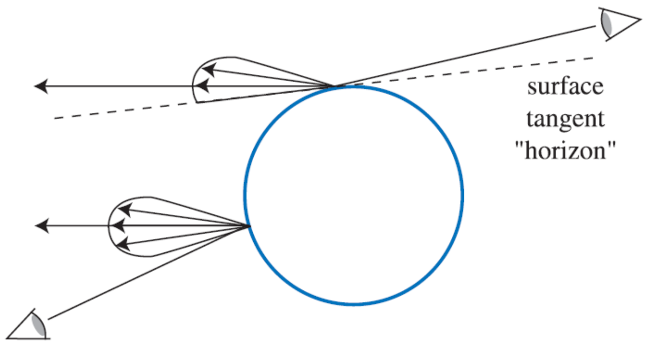


Figure 2.13: The reflection direction can be the same for different viewing directions. Both reflected view directions, the lower left viewer's and the upper right viewer's, result in the same reflection direction. While the proper lobe for the lower left viewer is a symmetric lobe, the upper right viewer's reflection lobe would have to be chopped off since light cannot reflect off the surface below its horizon. Reprinted from [AMHH08].

The techniques of McAllister et al. [MLH02, McA04] can handle spatially varying BRDFs. The technique of Green et al. [GKD07] takes a similar approach but uses Gaussian lobes instead of cosine lobes. Colbert and Křivánek [CK07] describe a technique based on Monte Carlo quadrature and importance sampling which can handle arbitrary BRDFs and chopped off lobe shapes.

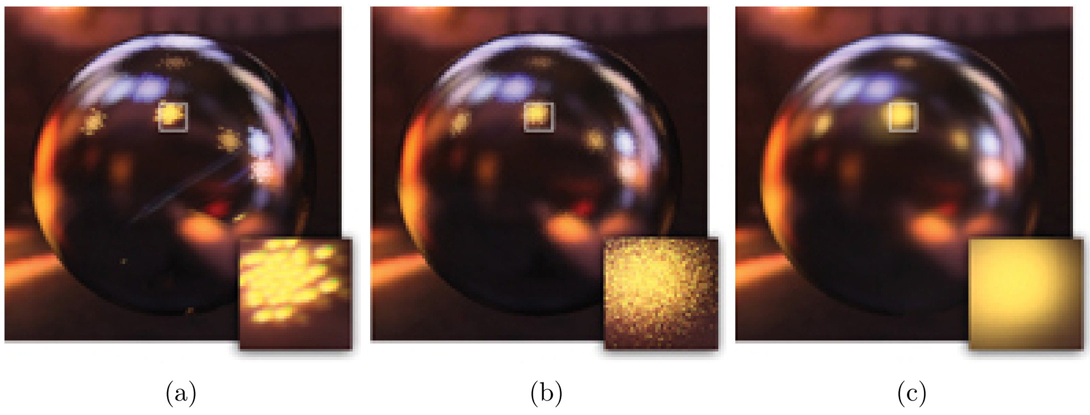


Figure 2.14: The results of various configurations of the technique by Colbert and Křivánek [CK07] are shown in images (a), (b), and (c), all adapted from [CK07]. Image (a) shows the result of using Monte Carlo quadrature only, image (b) shows the effect of importance sampling combined with Monte Carlo quadrature, and image (c) shows how an additional MIP-mapping based filtering helps to eliminate aliasing artifacts and noise when combined with Monte Carlo quadrature and importance sampling.

There are three important parts in the technique [CK07] by Colbert and Křivánek which can be combined to improve the quality: Monte Carlo quadrature, importance sampling, and MIP-mapping based filtering. Monte Carlo quadrature relies on a number of random samples which are averaged to yield an approximation to the accurate results instead of integrating over the entire hemisphere. Colbert and Křivánek are using 24 to 40 random direction samples. Importance sampling is an improvement, which does not use uniform random directions for approximating the integral over the hemisphere but use more random samples in the direction around the specular direction based on a probability density function (PDF). This PDF is used to calculate a weighted average of the samples. While using Monte Carlo quadrature alone, the results show a significant amount of aliasing, importance sampling changes aliasing into more visually acceptable noise which is an improvement that can be observed in figure 2.14. The figure also shows a further improvement which has been achieved by using MIP-mapping based filtering that is based on the PDF. A small value of a random sample direction's PDF means that it is unlikely that similar sample directions will be generated. Therefore, the illumination information for this sample is taken from a higher MIP-map level of the light probe, in which each texel contains the MIP-mapping-approximated averaged lighting information from a larger area. Vice versa, directions with a high PDF are taken from lower MIP-map levels of the light probe where each texel contains the MIP-mapping-approximated lighting information of a smaller area from the original light probe [CK07]. As a side note, the samples from the lower MIP-map levels are more accurate since the error introduced by simple averaging texel values gets bigger on higher MIP-map levels because every MIP-map level introduces additional error which is propagated up to all further MIP-map levels. While Colbert and Křivánek's method is able to handle arbitrary lobe shapes, they have introduced the described inaccuracies as a trade off for speed. This enables their technique to run at real-time frame rates. With a varying number of direction samples, their technique can scale between real-time frame rates and a result tending to the accurate solution.

Irradiance and reflection maps can be used in advanced rendering techniques like radiance caching [KGPB05] which uses the idea of irradiance caching [WRC88] and extends it to directional light information, where *radiance* is cached. The basic idea behind irradiance caching and radiance caching algorithms is to compute lighting information at certain locations across a scene and store them in a cache. Final shading values are computed by interpolating the previously computed lighting information from the adjacent cache items. Figure 2.15 depicts this approach. The colored areas represent cache locations which are calculated accurately. The shading of all other locations is interpolated from the caches.

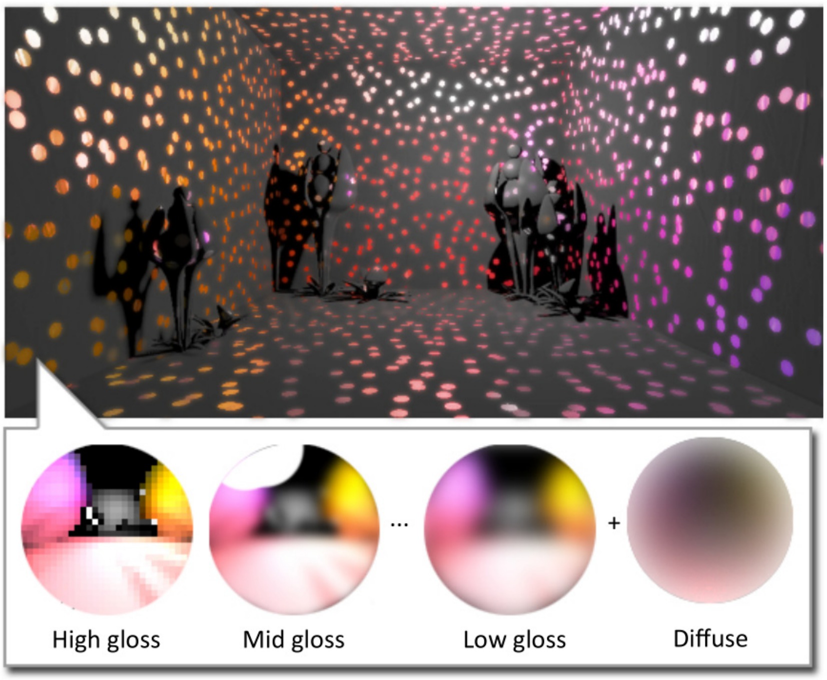


Figure 2.15: The colored regions represent cache locations. For these cache locations, irradiance and radiance values are computed accurately and stored. Each cache stores one irradiance map and multiple reflection maps which can be used to render a variety of materials: from diffuse to glossy and specular types. Adapted from [SNRS12].

Figure 2.15 shows radiance caching which stores directional light information. More specifically, the figure depicts the technique of Scherzer et al. [SNRS12]. While spherical harmonics (which we describe in the next section, 2.4) can be used to store lighting information, Scherzer et al. propose a MIP-mapping based solution. Radiance values are pre-computed for various types of materials: from diffuse and low glossy results to high glossy and specular results. The lighting information with low frequency changes are stored in the high MIP-map levels, which are smaller in size. The lighting information with high frequency changes, such as the high glossy and specular ones, are stored in the lower MIP-map levels which have a higher resolution. Scherzer et al. point out that their technique achieves higher performance with only a single constant-time lookup per pixel compared to computationally more demanding spherical harmonics based solutions while achieving almost the same quality. They decided to use a cache item texture size of 32×32 for the specular radiance values, hence the high glossy values are stored in a 16×16 region, low glossy in 8×8 and the diffuse lighting information in a 4×4 region which means the 3rd MIP-map level [SNRS12].

2.4 Spherical Harmonics

Storing irradiance and reflection maps as images is one possibility, but other representations also exist. Spherical harmonics [Mac28] (SH) store a frequency domain representation of the input signal and have become very popular recently – especially for representing irradiance maps [RH01a]. Spherical harmonics (SH) are basis functions which means a set of functions that can be weighted and summed to approximate some general space of functions [AMHH08]. More precisely, the functions which we describe here are called *the real spherical harmonics*. Also complex-valued SH exist, but we focus solely on the real spherical harmonics in this thesis.

Spherical harmonics are functions defined on the surface of a sphere. They are arranged in frequency bands. The first basis function on level zero is constant. The basis functions on level one are linear, which means they change very slowly over the surface of the sphere. The basis functions on level two are quadratic. They change faster than the basis functions on lower levels, and so on. The increasing orders of the basis functions depend on the SH-order, which can be observed in figure 2.16. Green areas on the sphere surfaces or green lobes mean that the SH basis functions have positive values. Red areas or red lobes mean that the SH basis functions have negative values.

While many sets of functions can form a basis (see figure 2.17 for an example), choosing an *orthogonal* set of basis functions has some important advantages. An orthogonal set of basis functions means that the inner product $\langle f_i(), f_j() \rangle$ of any two different functions f_i and f_j from the set is zero. For spherical functions, the inner product is defined like follows: [AMHH08]

$$\left\langle f_i(\vec{d}), f_j(\vec{d}) \right\rangle \equiv \int_{\Theta} f_i(\vec{d}) f_j(\vec{d}) d\omega \tag{2.13}$$

where Θ indicates that the integral is performed over the unit sphere. Parameter $\vec{d}$ is a direction vector. If additionally a set of orthogonal basis functions satisfies equation 2.14 for any combination of f_i and f_j, an orthogonal set is called *orthonormal.* [AMHH08]

$$\langle f_i(), f_j() \rangle = \begin{cases} 0, & \text{where } i \neq j. \\ 1, & \text{where } i = j. \end{cases} \tag{2.14}$$

The big advantage of a set of orthonormal basis functions is the process to approximate a target function, which is called *basis projection* [AMHH08]. Figure 2.18 depicts that orthonormal basis functions have superior properties compared to arbitrary function (as those in figure 2.17) by providing more control of the approximation. They are easier to calculate since they do not overlap.

For a given target function $f_{\text{target}}()$ it can be calculated how similar it is to a given basis function f_j by their inner product, yielding a coefficient k_j like shown in equation 2.15.

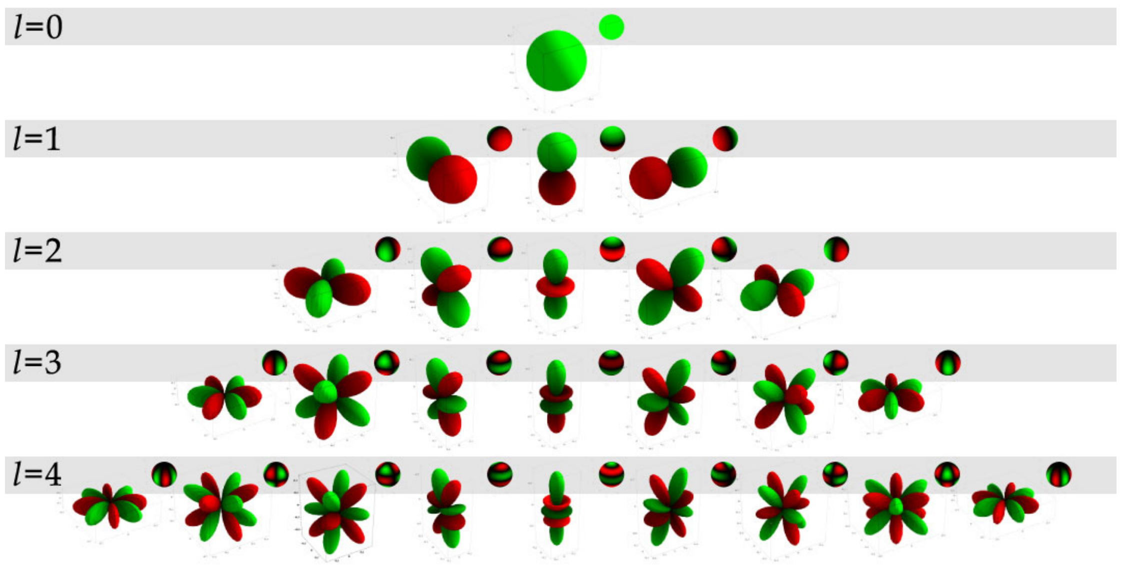


Figure 2.16: This is an illustration of the first five frequency bands of spherical harmonics. Each frequency band adds the same number of basis functions than the previous band plus two in addition. The SH functions' order increases with increasing frequency band which means that their value changes more often across the surface of the sphere than SH functions on lower frequency bands. This can be observed in either of the two different illustration types: The small spheres show green values on the surface of the spheres where the SH function is positive and turns red where the SH function has areas of negative values. The second illustration type shows the same information using lobes. The green lobes represent positive SH function values while the red lobes represent negative SH function values. Reprinted from [Gre03].

$$k_j = \langle f_{\text{target}}(), f_j() \rangle \tag{2.15}$$

The coefficients k_j can be used to approximate the original function by multiplying it with their corresponding basis functions and summing them up like shown in equation 2.16,

$$f_{\text{target}}() \approx \sum_{j=1}^{n} k_j f_j() \tag{2.16}$$

where the resulting approximation to $f_{\text{target}}()$ will get more accurate with an increasing number of basis functions in the set. In the context of spherical harmonics, this means: The more frequency bands are taken into account, the more functions will be in our orthonormal set of basis functions. Consequently, the higher the number of orthonormal basis functions, the more accurately a target function can be approximated.

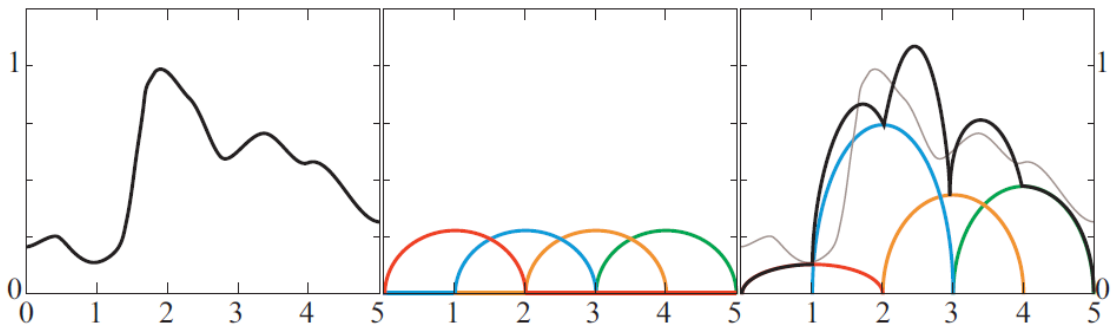


Figure 2.17: An explanatory example, reprinted from [AMHH08], shows how the target function in the left image is approximated by the set of basis functions in the center. Each basis function is shown in a different color. In the right image, the target function is approximated by the sum of the weighted basis functions. The black line shows the resulting reconstructed function while the gray line shows the target function. Since the basis functions are not orthonormal, it is hard to efficiently find their weights or coefficients.

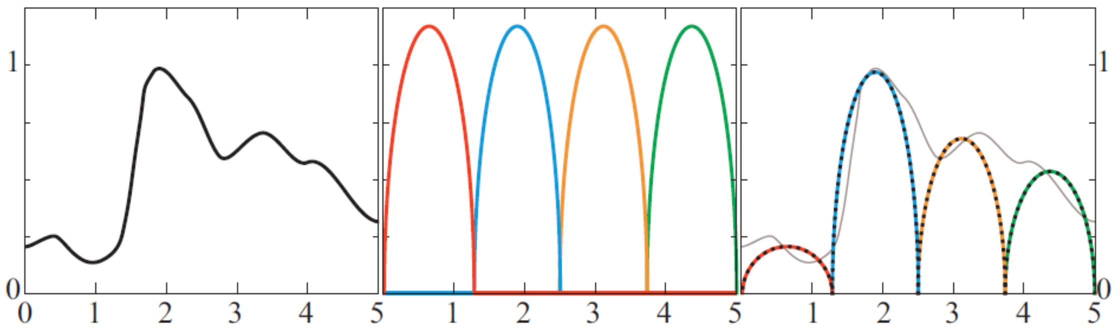


Figure 2.18: Another explanatory example, reprinted from [AMHH08], shows the same target function as in figure 2.17 in the left image. A different set of basis functions is used to approximate it, which is shown in the center image. These basis functions are orthonormal and the resulting approximation of the target function is shown as a dotted black line in the right image. Calculating the coefficients for these basis functions is straight-forward since they do not overlap.

Green [Gre03] and Schönefeld [Sch05] explain the background of spherical harmonic lighting comprehensively. The basis functions of real spherical harmonics are called *associated Legendre polynomials*, represented by the symbol P. They have two arguments l and m and are defined over the range $[-1, 1]$. The argument l is the *band index* and can be set to positive integer values starting from 0. The argument m can be set to integer values in the range $[0, l]$. Equations 2.17, 2.18, 2.19, and 2.20 describe the creation of the polynomials [Gre03].

$$P_0^0 = 1 \tag{2.17}$$

$$P_l^m = \frac{x(2l-1)P_{l-1}^m - (l+m-1)P_{l-2}^m}{(l-m)} \tag{2.18}$$

$$P_m^m = (-1)^m(2m-1)!!(1-x^2)^{m/2} \tag{2.19}$$

$$P_{m+1}^m = x(2m+1)P_m^m \tag{2.20}$$

These formulae are applied to the surface of a sphere by the standard conversions between cartesian coordinates (x, y, z) and spherical coordinates (θ, ϕ). Equations 2.21, 2.22, and 2.23 describe the conversion from cartesian coordinates to spherical coordinates,

$$r = \sqrt{x^2 + y^2 + z^2} \tag{2.21}$$

$$\phi = \tan^{-1}\left(\frac{y}{x}\right) \tag{2.22}$$

$$\theta = \cos^{-1}\left(\frac{z}{r}\right) \tag{2.23}$$

where r is the radius which is naturally 1 for points on the surface of the unit sphere. Equations 2.24, 2.25, and 2.26 describe the conversion from spherical coordinates to cartesian coordinates [Gre03, RH01a].

$$x = r\sin\theta\cos\phi \tag{2.24}$$

$$y = r\sin\theta\sin\phi \tag{2.25}$$

$$z = r\cos\theta \tag{2.26}$$

These conversions finally lead to the SH function y which is given by equation 2.27,

$$y_l^m(\theta,\phi) = \begin{cases} \sqrt{2}K_l^m \cos(m\phi)P_l^m(\cos\theta), & m > 0 \\ \sqrt{2}K_l^m \sin(-m\phi)P_l^{-m}(\cos\theta), & m < 0 \\ K_l^0 P_l^0(\cos\theta), & m = 0 \end{cases} \tag{2.27}$$

where P is the associated Legendre polynomial, K is a scaling factor defined like given by equation 2.28,

$$K_l^m = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}} \tag{2.28}$$

the parameter l is defined as a positive integer starting from 0, m is defined as a signed integer with valid values in the range $[-l, l]$ [Gre03]. Algorithms 2.5 and 2.6 show implementations of these equations.

Algorithm 2.5: Implementation of equations 2.17, 2.18, 2.19, and 2.20, following closely the code samples of [Gre03].

```
function P( l, m, x )
    input  : Parameters $l \in \mathbb{N}_0$, $m \in [0, l]$, and $x \in \mathbb{R}$
    output : Value of the associated Legendre polynomial $P_l^m$ at $x$
    pmm = 1;
    if $m > 0$ then
        somx2 = $\sqrt{(1-x)^2}$;
        fact = 1;
        for $i = 1..m$ do
            pmm = pmm $\cdot$ (−fact) $\cdot$ somx2;
            fact = fact + 2;
        end
    end
    if $l = m$ then
        return pmm;
    end
    pmmp1 = $x \cdot (2m+1) \cdot$ pmm;
    if $l = m + 1$ then
        return pmmp1;
    end
    pll = 0;
    for $i = m+2, i = i+1, \ldots, l$ do
        pll = $((2i-1) \cdot x \cdot$ pmmp1 $- (i+m-1) \cdot$ pmm$)/(i-m)$;
        pmm = pmmp1;
        pmmp1 = pll;
    end
    return pll;
```

Now we have everything needed to transform an environment map into its SH representation. A coefficient c_l^m is calculated by integrating the product of function f (i.e. the values sampled from our environment map) and the SH function y_l^m (equation 2.27) over the entire sphere Θ (i.e. all possible directions in our environment map) like shown in equation 2.29:

$$c_l^m = \int_\Theta f(s) y_l^m(s) ds \tag{2.29}$$

Algorithm 2.6: Implementation of equations 2.27 and 2.28, following closely the code samples of [Gre03].

```
1  function SH( l, m, θ, φ )
       input  : SH band $l \in \mathbb{N}_0$, band-index $m \in [-l, l]$, spherical coordinates
                $\theta \in [0, \pi]$, and $\phi \in [0, 2\pi]$
       output : Point sample of the SH basis function at band $l$ and band-index $m$
                with the spherical coordinates $\theta$ and $\phi$.
2      if $m = 0$ then
3          return $K(l, 0) \cdot P(l, 0, \cos(\theta))$;
4      else
5          if $m > 0$ then
6              return $\sqrt{2} \cdot K(l, m) \cdot \cos(m\phi) \cdot P(l, m, \cos(\theta))$;
7          else
8              return $\sqrt{2} \cdot K(l, -m) \cdot \sin(-m\phi) \cdot P(l, -m, \cos(\theta))$;
9          end
10     end
11 function K( l, m )
       input  : Parameters $l \in \mathbb{N}_0$, $m \in [-l, l]$
       output : Value of weighting factor $K$
12     return $\sqrt{((2l + 1) \cdot (l - m)!)/(4\pi \cdot (l + m)!)}$;
```

SH-order	SH bands l	total number of basis functions
1	0	1
2	0–1	4
3	0–2	9
5	0–4	25
10	0–9	100
35	0–34	1225
86	0–85	7396
100	0–99	10000
320	0–319	102400
1280	0–1279	1638400

Table 2.2: The number of SH basis functions has a quadratic relation to the SH-order.

Depending on the precision required, we calculate a certain number of coefficients and use them later to reconstruct the approximated function $\widetilde{f}$ like shown in equation 2.30.

$$\widetilde{f}(s) = \sum_{l=0}^{n-1} \sum_{m=-l}^{l} c_l^m y_l^m(s) \tag{2.30}$$

It is obvious that the number of SH basis functions (or SH coefficients) is in quadratic relation to the order. Table 2.2 gives an overview of some SH-orders, their corresponding frequency bands, and the numbers of total coefficients or basis functions to reconstruct a target function. Figure 2.19 shows the reconstruction of an image with a small number of SH basis functions using 4 SH bands (2.19b), 9 SH bands (2.19c), or 20 SH bands (2.19d), respectively. Since the original image contains high frequency image details, small numbers of SH bands are unable to reconstruct them and ringing artifacts occur.

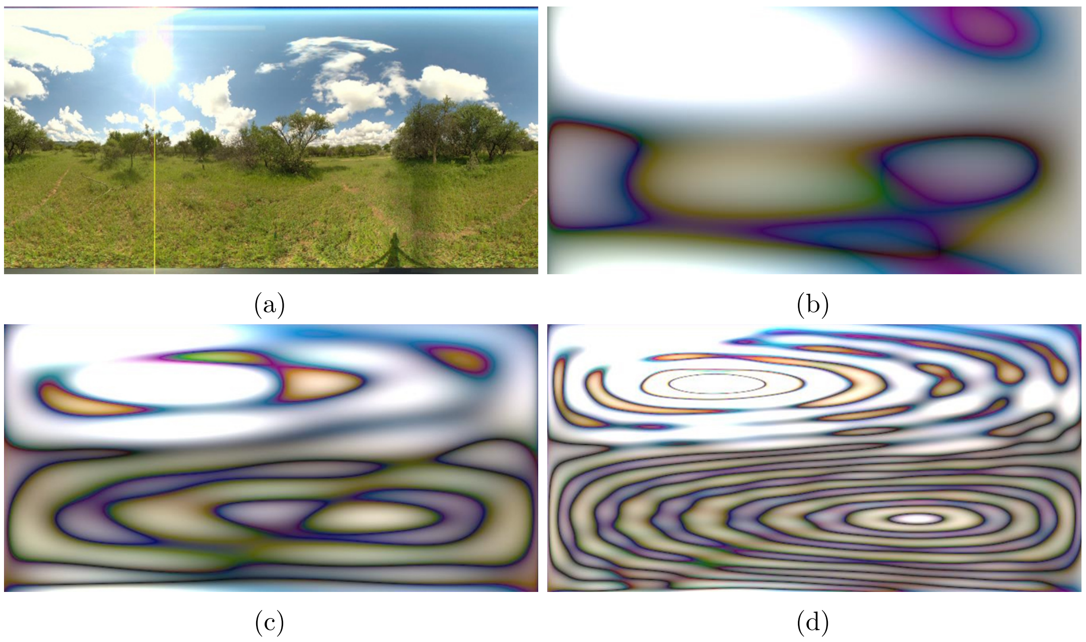


Figure 2.19: The original image (a) reconstructed with 4 SH bands (b), with 9 SH bands (c), and with 20 SH bands (d). The more SH bands are used for reconstruction, the more details can be reconstructed from the original image. Ringing artifacts are visible because the used numbers of SH bands are unable to capture all details of the original image sufficiently. All images adapted from [Add13].

If, however, an image contains only low frequency changes, even low-order SH are able to reconstruct the image very accurately. Irradiance maps change very slowly and therefore, low-order SH are able to represent them very accurately. Figure 2.20 shows the *St. Peters* light probe by Debevec [Deb01] and two irradiance maps, one of which was calculated accurately from the base image, while the second one was calculated using 3 SH bands, i.e. 9 SH basis functions. Ramamoorthi and Hanrahan [RH01a] point out that 9 SH coefficients are sufficient to reconstruct an irradiance map with an average error of less than 3% which is usually not noticeable.

In the context of irradiance and reflection mapping, the most important property of SH is an operation called *spherical convolution* [AMHH08]. It enables us to calculate the maps very efficiently in frequency space. A convolution with the Phong BRDF in the

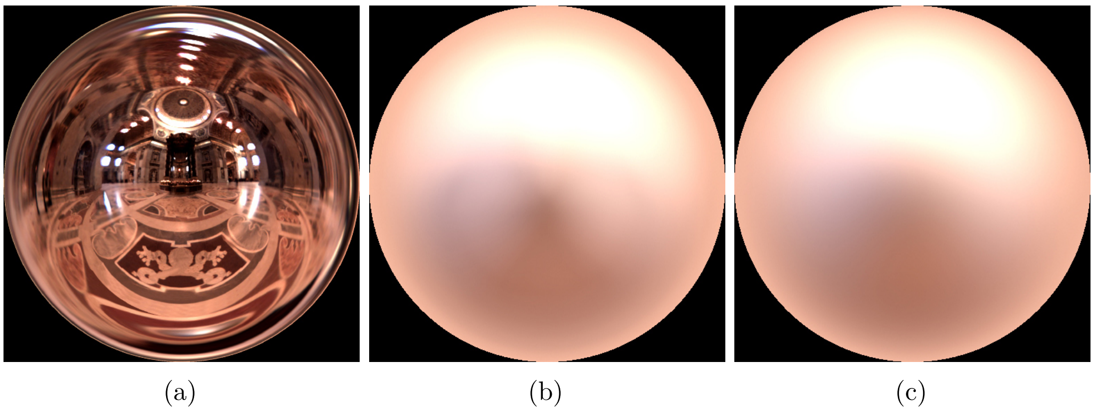


Figure 2.20: Image (a) shows the *St. Peters* light probe, adapted from [Deb01]. Image (b) shows an accurate calculation of the corresponding irradiance map, and image (c) shows a irradiance map calculated with spherical harmonics of order 3, which has 9 SH coefficients in total. Since irradiance maps do not have any medium- or high-frequency changes, but change slowly over the sphere surface, low-order SH are able to reconstruct the accurate irradiance maps very closely with small errors.

spatial domain can be expressed by a multiplication in the frequency domain.

Given a function f in spatial domain (our light probe, for instance) with its corresponding SH representation y_l^m, and a kernel function $h(z)$ with its corresponding SH representation $\hat{\rho}_l$, the convolution of h and f in the spatial domain can be expressed in frequency domain by equation 2.31,

$$(h * f)_l^m = \Lambda_l \ \hat{\rho}_l \ y_l^m \tag{2.31}$$

where Λ_l is defined in equation 2.32, and $h(z)$ has to satisfy the requirement to be rotationally symmetric around the z-axis [Slo08]. The Phong BRDF is rotationally symmetric, so the result of the convolution can also be represented in the SH domain [Slo08]. The SH coefficients $\hat{\rho}_l$ of rotationally symmetric functions $h(z)$ are indexed solely by l because they have only one non-zero SH coefficient per SH band l which corresponds to the basis functions in the center column of figure 2.16. They are also known as the *zonal harmonics* [AMHH08, Slo08].

$$\Lambda_l = \sqrt{\frac{4\pi}{2l+1}} \tag{2.32}$$

While Akenine-Möller et al. [AMHH08] give an overview of SH properties, Sloan [Slo08] describes them in detail, including SH properties like their *rotational invariance*. An in-depth description of the theory of light reflection as convolution in terms of SH is given by Ramamoorthi and Hanrahan in [RH01b]. They are describing how to deduce the SH coefficients for several suitable BRDFs, including the Phong BRDF, which was first described by MacRobert [Mac48]. The Phong BRDF's SH coefficients are given by equation 2.33,

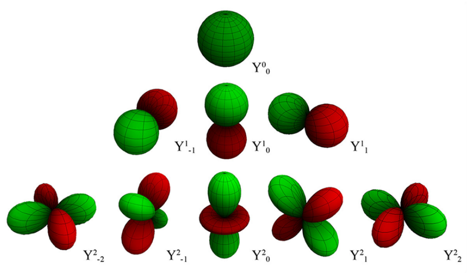
$$\Lambda_l \, \hat{\rho}_l = \begin{cases} \dfrac{(s+1)(s-1)(s-3)\dots(s-l+2)}{(s+l+1)(s+l-1)\dots(s+2)} & \text{ODD } l \\ \dfrac{s(s-2)\dots(s-l+2)}{(s+l+1)(s+l-1)\dots(s+3)} & \text{EVEN } l \end{cases} \tag{2.33}$$

where s is the specular shininess coefficient, and l represents the SH band.

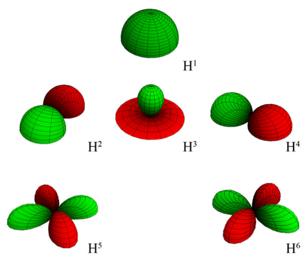
Due to their properties, being able to represent functions changing with a low frequency accurately, inexpensive to evaluate, and some beneficial mathematical properties like the spherical convolution in frequency domain, SH have become popular in the past years in computer graphics applications. Fundamental work in the area of spherical harmonics for real-time rendering has been created by Ramamoorthi and Hanrahan in [RH01a], [RH01b], and [RH02]. Their techniques have been used by many others. King describes a GPU-accelerated implementation in [Kin05]. Also Colbert and Křivánek use spherical harmonics for the diffuse illumination in [CK07]. For the specular part, however, they use importance sampling within Monte Carlo rendering. More of the mathematical background and further information about SH can be found in [Gre03, Sch05, Slo08].

A technique similar to ours described in this thesis has been created by Pessoa et al. In [PML$^+$10], they use an irradiance and multiple reflection maps to store illumination information for rendering virtual objects. They use SH for the irradiance and low-glossy reflection maps, but limit their their technique to SH of order 15 while we propose a more general solution. In contrast to our work, they do not focus on mobile devices.

SH are not the only basis functions suitable for representing illumination information. Habel and Wimmer created the $\mathcal{H}$-basis [HW10] which is a hemispherical basis. Its first few basis functions are shown in figure b.



(a)



(b)

Figure 2.21: Comparison of the first few SH basis functions in image (a) and the first few basis functions of the $\mathcal{H}$-basis in image (b). Adapted from [HW10].

2.5 Augmented Reality

Augmented Reality (AR) is a technology which combines a real-time view of a physical, real-world environment with virtual, computer-generated data. Virtual data is superimposed upon or composited with the real world, i.e. it is augmenting the real world [Azu97]. While computer-generated data is a general term which fundamentally can encompass data for all human senses like graphics, sound, haptics, taste, and smell. This thesis focuses solely on graphics. Graphics and sound are arguably the two fields where AR technology has advanced the most so far. Augmenting the other senses will be a fascinating field for future research.

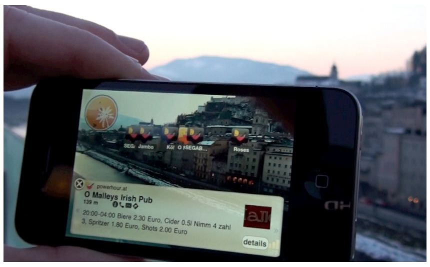
AR can be seen as a variation of Virtual Reality [Azu97]. The difference between these two technologies is that Virtual Reality completely immerses a user in a synthetic environment, while an AR environment will always include the real world. Azuma [Azu97] defines AR as systems that have the following three characteristics:

1. Combine real and virtual
2. Are interactive in real-time
3. Are registered in 3-D

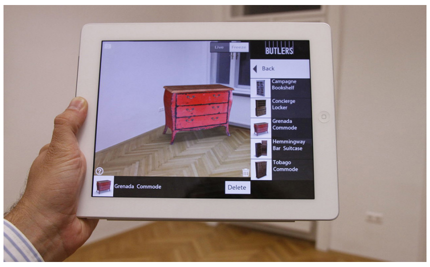
In order to augment reality, a capturing device is required to acquire real world data in real-time. After processing and augmenting the data, it is passed on to an output device which finally provides a combination of reality and virtual data to one or multiple users. Examples for a capturing device and an output device could be a camera or a display, respectively. Figure 2.22 shows different generations of AR devices. Head-up displays like shown in figure 2.22a have a long history in aircraft. They display information about the aircraft, like the angle of inclination, but also information about other objects in the real world, like the position of other aircraft tracked by radar like shown in figure 2.22b. Figure 2.22c shows an AR app on a smartphone. Since the iPhone was introduced in 2007, it has become standard that smartphones are equipped with a camera, IMUs, and relatively powerful processors which made them well-suited for AR applications. This led to rapid developments of powerful AR applications for smartphones. Two of them are shown in figures 2.22c and 2.22d - a global positioning system (GPS)-based application and an image-recognition-based application. Figure 2.22e shows an advanced AR device developed by Microsoft: the HoloLens. It is a head-mounted see-through device developed especially for AR applications and comprises a variety of sensors, multiple cameras and a special so-called "holographic processing unit" which should enable a higher degree of visual and spatial coherence between real and virtual objects than is possible with smarthphones today. Image 2.22f shows an illustration of a possible usage scenario of the HoloLens with multiple AR apps running at the same time, each having a strong spatial coherence with the real world.

(a)

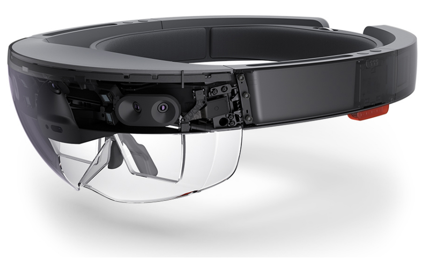
(b)



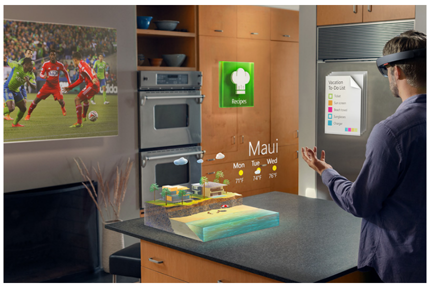
(c)



(d)



(e)



(f)

Figure 2.22: Head-up displays have a long history in aircraft like the synthetic vision system in a NASA's Gulfstream GV in image (a), adapted from[5]. Image (b), reprinted from[6], shows the view through a head-up display of a F-18 jet fighter during a mock dogfight where the location of another real aircraft is highlighted. Images (c), adapted from[7], and (d), adapted from[8], show AR applications on mobile devices: A GPS-based application in (c) which uses geo location data and a mobile device's IMU to annotate real-world buildings with their names and other information; and an application which uses image-recognition to put virtual furniture into real rooms (d). Image (e), adapted from[9], is an illustration of Microsoft's advanced AR device HoloLens showing its sensors, cameras, and see-through displays. Image (f), adapted from[10], shows how Microsoft intends the device to be used: Multiple AR apps are pinned to real walls and tables.

[5]Wikimedia Commons, Jeff Caplan for NASA
`https://commons.wikimedia.org/wiki/File:SVS_Cockpit_Gulfstream_GV.jpg`

[6]Wikimedia Commons, US Navy
`https://commons.wikimedia.org/wiki/File:F-18_HUD_gun_symbology.jpeg`

[7]Wikimedia Commons, Pucky2012 via Wikitude World Browser app
`https://commons.wikimedia.org/wiki/File:Wikitude_World_Browser_@Salzburg_Old_Town_3.jpg`

[8]Wikimedia Commons, Meximex via BUTLERS app
`https://commons.wikimedia.org/wiki/File:ViewAR_BUTLERS_Screenshot.jpg`

[9]Microsoft, Microsoft HoloLens
`https://compass-ssl.xboxlive.com/assets/34/9d/349d01da-3c16-4cd0-8094-89b0ce13d1b6.png`

[10]Microsoft, Microsoft Developer resources
`https://az835927.vo.msecnd.net/sites/holographic/resources/images/Image10.png`

Azuma states in [Azu97] that there are at least six fields for potential AR applications:

- Medical visualization
- Maintenance and repair
- Annotation
- Robot path planning
- Entertainment
- Military aircraft navigation and targeting

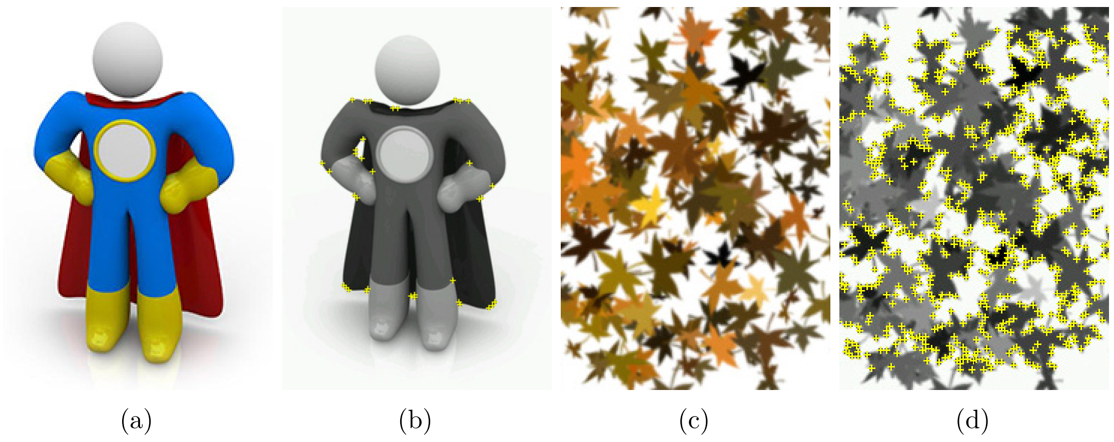
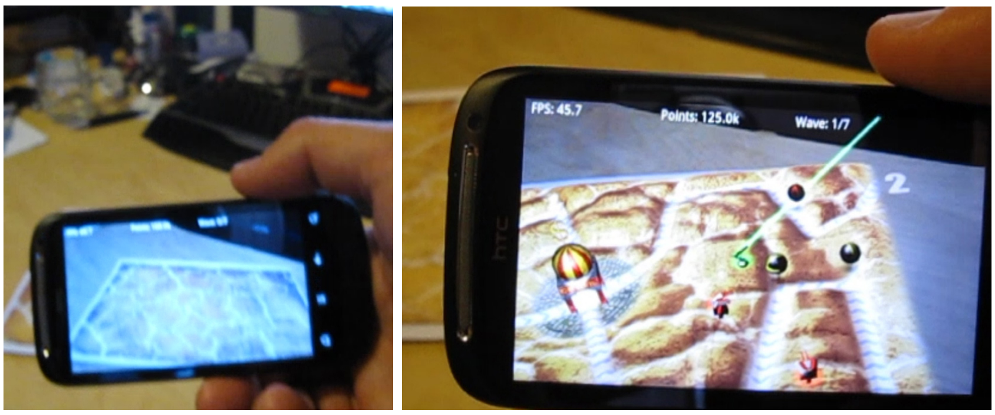


Figure 2.23: These images from the Vuforia developer portal[11] show which kinds of images the Vuforia SDK is able to track well and which images it has problems with in that regard. First of all, it does not consider colors which is why images (b) and (d) are shown in grayscale. It only considers local contrast changes and strives to detect mainly corners which it considers to be image features. From these image features, it calculates the pose matrix in each frame and provides it to the SDK user. Image (a) has almost no sharp edges and a lot of round elements which results in a very small number of image features as can be seen in (b). The locations of the recognized image features are represented by small yellow crosses. Image (c), on the other hand, has high local contrast and a lot of sharp edges which results in a lot of image features as can be seen in (d). Image (c) should track a lot better than image (a) with the Vuforia tracking SDK, showing a much higher spatial and temporal coherence. All images adapted from[11].

[11]Vuforia Developer Portal, PTC Inc., `https://developer.vuforia.com`

2.5.1 Augmented Reality on Mobile Devices

Mobile devices have undergone rapid development in the past years. Many smartphones feature high resolution cameras and powerful multi-core processors. Virtually every smartphone which is released to the market at the time of writing feature a camera, at least a dual-core central processing unit (CPU) combined with a GPU in a System on a Chip (SoC), and an inertial measurement unit (IMU) which include a gyroscope, an accelerometer and a magnetometer. These devices are well-suited for AR applications, thus several high-quality libraries have been ported to mobile platforms, like the powerful computer vision library *OpenCV*[12], and software development kits (SDKs) especially targeted to mobile devices have been created like the AR tracking SDK *Vuforia*[13]. A tracking SDK is analyzing the mobile device's camera images for target markers or target images. If it successfully recognizes one, it calculates the mobile device's position relative to the target marker or image and usually calculates a so-called *pose matrix* which represents the transformation matrix from the mobile device to the target. This matrix can be used in a 3D rendering application to position objects in 3D space in relation to the tracking target.



(a) (b)

Figure 2.24: An AR tower defense game created by Michael Probst and Johannes Unterguggenberger for the course *Augmented Reality on Mobile Devices* on Vienna University of Technology in 2011. It uses Vuforia tracking SDK to find the image target, shown in (a). The image target looks like a dry cracked desert ground and has many image features like sharp corners and high local contrast which Vuforia is able to track well. Once Vuforia has recognized the target, it provides a pose matrix. The game uses it to render the battlefield atop the image target like shown in (b).

The Vuforia tracking SDK can recognize image features like corners. Based on the recognized features, it computes the pose matrix. The more appropriate image features a target image has, the better the tracking will be in terms of precision and stability. Figure

[12]OpenCV by Itseez, Inc., `http://opencv.org`
[13]Vuforia by PTC Inc., `http://www.vuforia.com`

2.23 illustrates features the Vuforia tracking SDK takes into account when searching for image targets and tracking them. Image targets without sharp edges and corners are not well-suited as tracking targets, while images with a lot of sharp edges and corners, and a high local contrast should provide the Vuforia SDK with plenty of features which it can use to track and calculate the pose matrix accurately and in a temporally coherent way. An example of an AR application that uses Vuforia and a well-suited tracking target is shown in figure 2.24.

Zhou et al. [ZDB08] describe trends in AR tracking, interaction and display research including an overview of AR tracking techniques. They point out that vision-based tracking was the most active area of tracking research.

2.5.2 Realistic Rendering in Augmented Reality

Depending on the AR application, it can be desirable to render the virtual objects realistically or in a non-realistic style. Haller [Hal04] investigated use cases and properties of photo-realistic and non-photo-realistic rendering in AR. For some visualization tasks or productivity tools, non-photo-realistic rendering might be more suitable. For example, it can be observed that the HoloLens apps in figure 2.22f are deliberately using non-photo-realistic rendering for the virtual elements. On the other hand, there are many applications that benefit from rendering virtual objects as realistically as possible like the app shown in figure 2.22d which has the purpose to show the user how it would look like if there was an additional piece of furniture placed inside a room. That piece of furniture should be rendered as realistically as possible with the exact lighting conditions of the real room in which the user is using the app. There are many more types of AR applications which can benefit from realistic rendering in various areas including archeology, architecture, commerce, construction, education, entertainment, gaming, industrial design, television, tourism and sightseeing.

Our technique contributes to the realistic rendering category. Therefore, we are mentioning some previous research work from this field. Many of these techniques have not been targeted towards being used on a mobile device in contrast to our technique.

Pioneering work in light transport for AR has been created by Fournier [FGR93] and later by Debevec [Deb98], who provides an extensive light probe gallery under [Deb01]. Techniques which specifically focus on rendering shadows in AR have been created by Gibson et al. [GM00] and by Haller et al. [HDH03]. Besides these two techniques, the more advanced rendering techniques like [KTM$^+$10] also render shadows in both directions: real objects cast shadows on virtual objects, and virtual objects cast shadows on real objects. The ray-tracing-based methods like [KK12b, KK13a, KK13b, Kán14] naturally produce shadows inherently. Grosch was one of the first to present a technique to render accurate refractions and caustics in AR using differential photon mapping [Gro05]. Karsch created a technique to realistically insert synthetic objects into existing photographs without requiring access to the scene or any additional scene measurements [KHFH11].

Knecht [KTM$^+$10] created a plausible realistic rendering method for mixed reality systems based on Instant Radiosity and Differential Rendering exploiting temporal coherence to improve quality and to reduce flickering artifacts. It is a rasterization solution and achieves real-time performance with the cost of approximation error compared to ray-tracing solutions.

Kán and Kaufmann describe a technique called *differential progressive path tracing* in which they are using path tracing to create a framework capable of simulating complex global lighting between real and virtual scenes including automatic light source estimation [KK13b].

Another technique by Kán and Kaufmann based on photon mapping is presented in [Kán14]. It is able to achieve high-quality AR rendering at interactive frame rates. Their primary goal was to create a ray-tracing based technique for high-quality AR and to improve visual coherence between real and virtual objects. Their technique is able to create high-quality effects such as caustics, reflection, refraction, and depth of field [Kán14].

Kán and Kaufmann created a study about the effects of direct and global illumination on presence in AR [KDB$^+$14]. Participants were asked to judge which of the shown objects were virtual and which ones were real. One of their conclusions was that shadows are very important. Especially when real shadows are present in a scene, it is important that virtual objects also have them. Inconsistencies in shadows can be very obvious to the viewer. Precise camera pose tracking improves spatial coherence while realistic rendering of virtual objects improves visual coherence. They state that global illumination leads to a higher sense of presence for certain scenes. According to their study, there is a significant correlation between the perception of realism and presence. Another study has been created by Hattenberger et al. In [HFJS09] they compare different rendering solutions and try to find out which of them yield better results than others. Kán and Kaufmann conclude based on their user study that each developed rendering effect in their application had a positive impact to visual coherence in AR [Kán14].

CHAPTER 3

Environment Map Capturing

There are various approaches to capture environment maps. Debevec [Deb01, Deb98] made photographs of reflective spheres to create static light probes which can be used directly in a rendering application as sphere maps. This technique was extended by Waese and Debevec in [WD02] to capture high dynamic range light probes in real-time by attaching several filters to a video camera which records a mirrored ball.

Many recent techniques are using cameras with a fish-eye lenses to capture the real-world light information in real-time. This approach is followed in [Fra13, Fra14, GEM07, KK13a, KK13b, KTM^{+}10]. These techniques are implemented as desktop applications and require a separate camera to capture the real-time light probe. Techniques which are better suited for mobile devices have been developed in the field of panorama capturing [SS97, DWWW08, WMLS10]. Many techniques, however, do not cover the full sphere of light.

Our application is targeted to run on a mobile device without the need for additional hardware like external cameras and it should be able to capture the full sphere of light. Therefore, we are using a novel technique by Kán [Kán15]. The user can create an omni-directional image of the surrounding by using the mobile device's camera to capture multiple images which are stitched together until the environment map is complete. This process is illustrated in figure 3.1. Of course, this technique can not capture environment maps in real-time, but to our knowledge, there is currently no technique which is capable of doing that with a commodity mobile device without additional hardware.

While the user turns the device around, the mobile device's IMU, which includes gyroscope, accelerometer and magnetometer, can be used to determine the orientation of the device [Kán15]. The algorithm assumes purely rotational motion and does not regard translational motion of the mobile device. The orientation gets mapped to a certain region in the sphere map like follows: For each pixel of the resulting sphere map with texture coordinates (u, v), the corresponding direction vector $\vec{d}$ is calculated via an

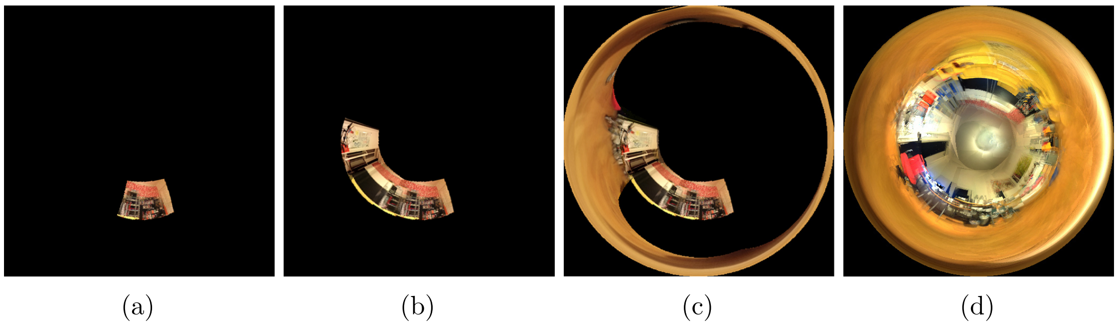


Figure 3.1: The process of capturing an environment map with the technique of Kán [Kán15] accumulates multiple camera images to produce the final HDR environment map, which captures the full sphere of light. Image (a) shows the very beginning of the environment capturing process where one single camera image has been projected to a certain region of the sphere map. Image (b) is composed of already four camera images, while even more have been added to the sphere map in image (c), one of which got mapped to the sphere map's singularity. Image (d) shows a completely captured environment map.

implementation like shown in algorithm 2.4. The angle map projection format is used as described in section 2.2.1. In contrast to the angle map projection as proposed by Debevec in [Deb01], these maps are oriented to have the positive z axis projected to the image center. For each new camera image, ray-plane intersections between all direction vectors $\vec{d}$ and the camera image plane are calculated. The image plane depends on the device orientation. For all intersection points that lie within the camera image frame, camera image data is stored at the corresponding (u, v) coordinates as high dynamic range (HDR) values [Kán15].

Camera image data which is low dynamic range (LDR) data is projected to HDR radiance values by using inverse camera response functions and the camera's exposure time. Kán used an approach presented by Robertson et al. [RBS99]. The mobile device's camera is set to auto-exposure mode which has the effect that the exposure times constantly change depending on where the camera is pointing to. If projected camera image data overlaps already existing data in the sphere map, HDR radiance values and weights are accumulated and, finally, the accumulated data is divided by the accumulated weights to get the resulting radiance values [Kán15].

In order to improve quality of the generated environment map, feature matching is applied to a projected captured camera image when it is merged with the already captured data. It can compensate small translational motion of the mobile device and drift from the device's IMU sensors. A new camera image is aligned with captured environment map by projecting the environment map to the camera frame. There, feature matching is applied and the matched points are used to calculate a rotational correction. Kán indicates that the quality of the image alignment depends on the number of features recognized by the algorithm but also states that imperfect alignment is barely noticeable after light reflection on a non-specular surface in terms of IBL [Kán15].

Kán points out that besides he created a novel method for HDR environment map capturing on a mobile device, the main contributions of his work include an accurate radiance estimation and accumulation from moving camera with automatic exposure, and a method for interactive alignment of new camera data with the previously captured data [Kán15]. Critical parts of the implementation are accelerated by efficient GPU implementations like HDR reconstruction and environment map accumulation into a floating point framebuffer [Kán15]. An example of an environment map captured with this technique is shown in image 3.2.

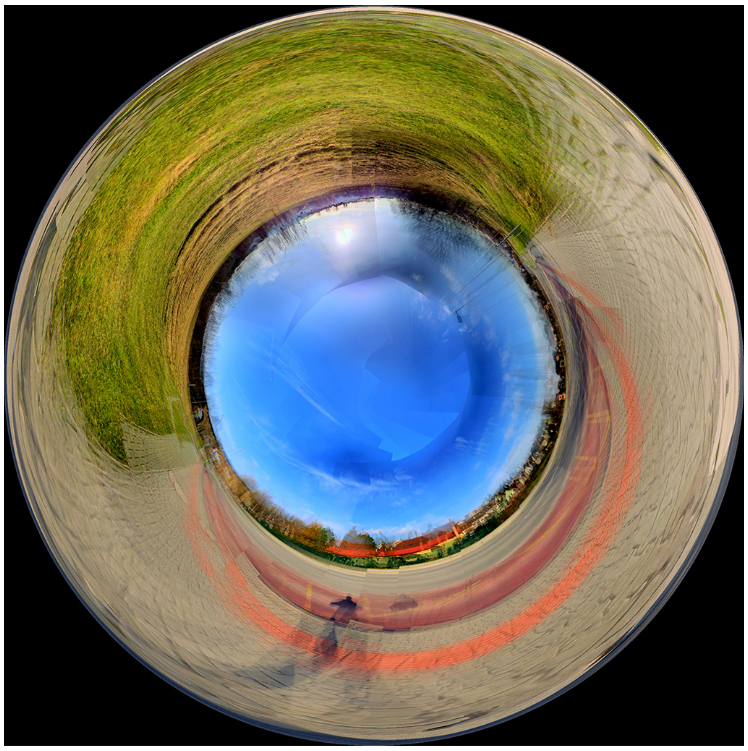


Figure 3.2: An environment map captured with Kán's technique using a commodity mobile device. Reprinted from [Kán15]

CHAPTER 4

Irradiance and Reflection Maps Calculation

In this chapter, we present different methods for calculating irradiance and reflection maps. We have implemented three fundamentally different methods, each has its advantages and shortcomings. Speed and quality significantly vary among the different methods.

Our methods can be applied to a variety of use cases, but generally, we aimed for a high quality level. That means, it should be possible to configure methods to produce very accurate results. We targeted the mobile operating system *Android* and the graphics API *OpenGL ES 3.0*.

For all three methods, our main requirement is that real-time frame rates are possible for rendering an AR scene with virtual objects that are illuminated using the calculated irradiance and reflection maps. If the maps cannot be calculated in real-time with a certain method, we calculate them on demand in an intermitting compute step which is either blocking the app or causing it to slow down to interactive frame rates. The implementations are able to handle simultaneous calculation of multiple reflection maps for arbitrary specular shininess coefficients. For our methods, we use the Phong BRDF. Applying the Phong BRDF induces the incident directions for calculating the irradiance map being weighted by the cosine, while the incident directions for calculating the reflection maps are weighted by $\cos^s$, where s is some arbitrary specular shininess coefficient greater than 1.

4.1 Accurate Calculation

The idea behind the accurate method is to calculate accurate reflection maps for a Phong BRDF with a given specular shininess coefficient. Calculating the maps accurately means that for each outgoing direction vector every incoming radiance vector from the base environment map has to be taken into account. In the final texture, every texel contains the radiance or irradiance values for the corresponding direction. In order to calculate one value, we have to iterate over all texels of the environment map, calculate their corresponding incoming radiance directions and add the lighting contribution from each direction weighted by the BRDF. The accurate calculation has quadratic runtime-complexity. For a 1024×1024 sized map, this means more than 1 trillion (10^{12}) steps are needed to calculate the reflection map. This complexity prohibits real-time frame rates. Therefore, we calculate the accurate maps in an intermitting calculation step. Our specific workflow including this calculation step is as follows:

1. An omni-directional image of the environment is captured by pointing the mobile device in different directions until the environment map is complete, representing the full sphere of light.
2. The app calculates the irradiance map and multiple reflection maps in an intermitting compute step which is causing the app to slow down to interactive frame rates
3. The calculated maps are used to render virtual objects in an Augmented Reality scene in real-time.

In the second step, the accurate maps are calculated. It can be triggered by the user and will take some time to complete. Since the app is used on a mobile device and the user is basically blocked from performing other complex tasks during this intermitting compute step due to the high workload, there are several requirements we have to consider:

- The implementation shall be stable and shall reliably complete in finite time.
- The computations shall be performed as fast as possible to reduce the user's waiting time.
- The computations shall be performed in an efficient way since it is desirable to minimize a mobile device's power consumption.
- It is desirable that the application remains responsive during the pre-computation step and indicates progress.

Our implementation fulfills these requirements. As a progress indicator, we display the irradiance and reflection maps which we calculate tile-wise. Tiles which are to be calculated are rendered in black while the already calculated areas already show the final results. Not performing the calculations tile-wise but instead the whole map at once would cause the app to block completely until the calculations have completed.

Algorithm 4.1: Calculation of the radiance or irradiance for one outgoing direction.

```
1  function calculateRadianceAccurate( $\vec{c}$ )
       input  : $\vec{c}$, baseTexSampler, baseTexSize, $N$, specCoeff[$N$], tileOffset, tileScale
       output : $\vec{L_o}[N]$
2      $\vec{L_o}[N] = \{(0,0,0), \ldots, (0,0,0)\}$;
3      $\vec{c_t} = \vec{c} \cdot tileScale + tileOffset$;
4      if $(2 \cdot \vec{c_t}.s - 1)^2 + (2 \cdot \vec{c_t}.t - 1)^2 > 1$ then
           /* Outside of valid sphere map texture area */
5          return $\vec{L_o}$;
6      end
       /* Within valid sphere map texture area */
7      $oneTxl = 1/baseTexSize$;                // size of one texel
8      $halfTxl = oneTxl/2$;                    // size of a half texel
9      $start = 0$;
10     $end = 1 - oneTxl$;
11     $dirOffset = halfTxl$;
       /* Calculate the corresponding (outgoing) direction vector: */
12     $\vec{d}$ = reverseSphereMappingAngleMap($\vec{c_t}.s + dirOffset$, $\vec{c_t}.t + dirOffset$);
13     $\vec{a_\mathtt{w}}[N] = \{0, \ldots, 0\}$;   // stores accumulated weighted lighting contributions
14     for $t = start, start + oneTxl, \ldots, end$ do
15         $b[2]$ = getRowBounds($t$, $oneTxl$, $halfTxl$);
16         for $s = b[1], b[1] + oneTxl, \ldots, b[2]$ do
               /* Get incoming radiance direction: */
17             $\vec{\omega'}$ = normalize(reverseSphereMappingAngleMap($s + dirOffset$,
                $t + dirOffset$));
18             $lambertian = \max(0, \mathrm{dot}(\vec{d}, \vec{\omega'}))$;
               /* Get incoming radiance value as RGB vector: */
19             $\vec{L_i}$ = sampleTexture(baseTexSampler, $s$, $t$);
20             $\mathtt{w}$ = getWeight($s$, $t$);
21             for $k = 1 \ldots N$ do
                   /* Apply the Phong BRDF by taking the cosine to the power of the specular
                      shininess coefficient */
22                 $f$ = pow($lambertian$, $specCoeff[k]$);
23                 $\vec{a_\mathtt{w}}[k] = \vec{a_\mathtt{w}}[k] + f \cdot \mathtt{w}$;
                   /* Calculate the outgoing radiance contribution and accumulate: */
24                 $\vec{L_o}[k] = \vec{L_o}[k] + \vec{L_i} \cdot f \cdot \mathtt{w}$;
25             end
26         end
27     end
28     for $k = 1 \ldots N$ do
29         $\vec{L_o}[k] = \vec{L_o}[k] / \vec{a_\mathtt{w}}[k]$;
30     end
31     return $\vec{L_o}$;
```

Algorithm 4.2: Helper functions `getRowBounds` and `getWeight` for algorithm 4.1

```
1  function getRowBounds(t, oneTxl, halfTxl)
       input  : $t$, $oneTxl$, $halfTxl$
       output : $b[2]$
       /* start and end coordinates lie on a circle */
2      $t' = (t - 0.5) \cdot (t - 0.5)$;
3      $s1 = 0.5 - \sqrt{0.25 - t'}$;
4      $s2 = 0.5 + \sqrt{0.25 - t'}$;
       /* align with texel centers */
5      $m1 = mod(s1, fullTxl)$;
6      $m2 = mod(s2, fullTxl)$;
7      $s1 = s1 + (fullTxl - m1) - halfTxl$;
8      $s2 = s2 - m2 + halfTxl$;
9      $result[2] = \{s1, s2\}$;
10     return result;
11 function getWeight(s, t)
12     $\vec{w}$ = sampleTexture($weightTexSampler$, $s$, $t$);
13     return $\vec{w}.r$;
14 function calcWeight(s, t)
15     $s' = 2s - 1$;
16     $t' = 2s - 1$;
17     $r = \sqrt{s'^2 + t'^2}$;
18     return sinc($\pi \cdot r$);
```

$\vec{c}$	texture coordinates as 2D vector with elements s and t
baseTexSampler	a sampler to access the base texture
baseTexSize	size of the texture accessed by *baseTexSampler* (square texture)
N	number of maps to calculate simultaneously
specCoeff[N]	array containing the specular coefficient for each map
tileOffset	offset of the tile to calculate radiance and irradiance values for
tileScale	size of the tile to calculate radiance and irradiance values for

Table 4.1: Description of the input variables and constants used with algorithm 4.1

Algorithm 4.1 shows our proposed implementation of the accurate calculation, used helper functions are shown in algorithm 4.2. We run these algorithms in a GPU shader which can calculate multiple target textures simultaneously. This is the reason why the algorithm uses an array of specular shininess coefficients of length N as input parameter and consequently returns an array of length N containing the resulting irradiance or radiance values for each specular coefficient that was passed as an input parameter. In an OpenGL implementation, this can be achieved by using the constants

GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1, etc. with the OpenGL functions glFramebufferTexture2D for creating a framebuffer and glDrawBuffers to render into it [Khr14].

The input parameters for algorithm 4.1 are the number of specular shininess coefficients N, a sampler for the base tetxure and its texture size, the parameters *tileOffset* and *tileScale* for specifying a tile, and per fragment the varying texture coordinates $\vec{c}$ as 2D-vector in the range $[0, 1]$, addressing the texel centers. Setting the tiling parameters $\mathit{tileOffset} = 0$ and $\mathit{tileScale} = 1$ will calculate the whole texture at once. Setting them to other values can be used to calculate only one tile: The offset from the border is specified by *tileOffset*, and the square-sized tile's side length is specified by *tileScale*. Valid ranges for both parameters lie in the range $[0, 1]$.

We use the tile properties to calculate the whole maps over multiple consecutive frames. For input textures sized 512×512 or larger, we have observed that our test device (Nvidia Shield) was not able to calculate the maps in one frame using just one tile. The high number of calculations made the device crash, but we were able to adapt our algorithm such that it successfully and reliably calculates the maps using a tile size of 128×128 or smaller.

Algorithm 4.1 performs a check whether the coordinates are outside of the valid circular area of a sphere map in line 4. If they are inside the valid sphere map texture area, the irradiance and outgoing radiance calculations are performed, starting effectively at line 12.

We assume a texture coordinate range of $[0, 1]$. The variables *oneTxl* and *halfTxl* are used to calculate the size of one texel or half a texel, respectively, with regard to the texture size (*baseTexSize*). The variable *dirOffset* is used to specify the offset of a texel's center to its actual coordinates. OpenGL accesses texels at their lower left corner whereas the texel center is calculated by adding the size of half a texel. Also DirectX versions 10 and above address texels in the same way. However, DirectX 9 and lower versions access a texel with their center coordinates. In the case of DirectX 9, the variable values in lines 9-11 would have to be set like follows: $\mathit{start} = \mathit{halfTxl}$, $\mathit{end} = 1 - \mathit{halfTxl}$, and $\mathit{dirOffset} = 0$. [Gra03]

In line 12, the reverse sphere mapping formula is applied which is given by algorithm 2.4, yielding the corresponding direction to the center of the given texel. The loop in line 14 iterates over all rows of the texture. The loop in line 16 iterates over all valid columns for the current row – from the left border of a sphere map's circular image area to the right. Line 17 performs reverse sphere mapping again, this time on the current iteration's texture coordinate. The cosine between the two vectors is calculated in order to apply the Phong BRDF which is finally done in line 22 by taking the cosine to the power of the specular coefficient.

The incoming radiance value is sampled from the environment map in line 19 and used in line 24 for calculating the accumulative outgoing radiance value. As described in section 2.2.1, using angle map projection requires a proper weighting. The weight for the current iteration's texture coordinate is sampled or calculated in line 20, accumulated in line 23, and finally used in line 29 to normalize the accumulated outgoing radiance/irradiance contribution yielding the final result.

The input values to algorithm 4.1 are listed and described in table 4.1. Algorithm 4.2 shows the helper functions *getRowBounds* and *getWeight* which we use as described above. An alternative to *getWeight* is the helper function *calcWeight* which does not sample the weight from a texture but calculates it. This formula applies to sphere maps with angle map projection format and can be found in the implementation by Ramamoorthi and Hanrahan [RH01a].

4.2 MIP-Mapping

Instead of trying to compute physically realistic irradiance and reflection maps, MIP-mapping calculates their approximation by blurring the environment map. MIP-mapping is naturally supported by mobile GPUs. Given an OpenGL texture handle, the MIP-maps of the texture can be calculated by calling the OpenGL function `glGenerateMipmap` [Khr14]. The results of MIP-mapping a light probe are shown in figure 4.1. In accordance with [SNRS12], we use the 4×4 pixel sized MIP-map level as the irradiance map and all MIP-map levels with the same or higher resolution as reflection maps.

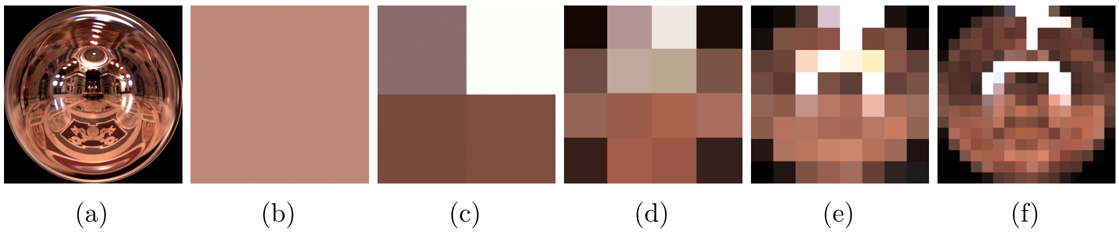


Figure 4.1: Base texture *St. Peters*, adapted from [Deb01], (a) and its highest five MIP-map levels: Image (b) shows the highest level of size 1×1 pixels, the second highest level (c) is sized 2×2 pixels, the next lower level (d) is sized 4×4 pixels, and so on. The color of one pixel in a MIP-map level is calculated as the average pixel color of the four corresponding pixels on the next lower MIP-map level. The lowest level is the base texture. The highest level contains one single pixel which contains the average color of all pixels in the base texture.

During rendering, we calculate the maximum MIP-map level via equation 4.1 which depends on the texture size N.

$$l_{\max} = \log_2(N) \tag{4.1}$$

For a 1024×1024 sized texture, this means that the maximum MIP-map level (the one which contains only a single pixel) is $l_{max} = 10$. From l_{max}, we subtract 2, like shown in equation 4.2, to get the 4×4 sized MIP-map level which we use to sample the irradiance from.

$$l_{\text{irrad}} = l_{\text{max}} - 2 \tag{4.2}$$

The MIP-map level to sample radiance values from is based on the material's specular coefficient like given in equation 4.3:

$$l_{\text{rad}} = l_{\text{max}} - \log_2\left(\min\left(\max(s/2, 4), N\right)\right) \tag{4.3}$$

where l_{rad} is the MIP-map level to sample radiance values from, s is the specular coefficient, and N is the maximum specular coefficient, which is set to the side length of a square texture. s is clamped to values between 4 and N in order to restrict the result to values between 0 and l_{irrad}. The division of s by 2 is scaling to fit the MIP-mapped results better to the accurate results. In a GLSL shader, the `textureLod` function can be used to sample from a specific MIP-map level [Khr14].

Algorithm 4.3: Fragment shader implementation using the MIP-mapping method and Phong BRDF

```
input  : $N$, $texSampler$, $s$, $k_d$, $k_s$
output : $\vec{L}$
1  function shade( $\vec{n}$, $\vec{v}$ )
       /* Calculate MIP-map levels:                                  */
2      $l_{\text{max}} = \log_2(N)$;
3      $l_{\text{irrad}} = l_{\text{max}} - 2$;
4      $l_{\text{rad}} = l_{\text{max}} - \log_2(\min(\max(s/2, 4), N))$;
       /* Calculate sphere map coordinates:                          */
5      $\vec{c_d}$ = sphereMappingAngleMap( $\vec{n}$ );
6      $\vec{r}$ = reflect( $\vec{v}, \vec{n}$ );
7      $\vec{c_s}$ = sphereMappingAngleMap( $\vec{r}$ );
       /* Sample illumination values from the appropriate
          MIP-map levels:                                            */
8      $\vec{i_d}$ = textureLod($texSampler$, $\vec{c_d}$, $l_{\text{irrad}}$);
9      $\vec{i_s}$ = textureLod($texSampler$, $\vec{c_s}$, $l_{\text{rad}}$);
       /* Calculate illumination via Phong BRDF:                     */
10     $\vec{L} = k_d \cdot \vec{i_d} + k_s \cdot \vec{i_s}$;
11     return $\vec{L}$;
```

Algorithm 4.3 shows our implementation of IBL using MIP-mapping. As input, it needs the side length of the square-sized input texture N, a *texSampler* to access it, the material's specular coefficient s, the material's Phong coefficients k_d and k_s, and per fragment the varying parameters surface normal $\vec{n}$ and current view vector $\vec{v}$. In lines 5 and 7 the texture coordinates for texture lookup in the sphere map are calculated for diffuse and specular lighting contribution, respectively. The resulting 2D vectors $\vec{c_d}$ and $\vec{c_s}$ are used for texture sampling in lines 8 and 9. The diffuse lighting value is stored in the 3D vector $\vec{i_d}$ as RGB-value while the 3D vector $\vec{i_s}$ holds the sampled specular lighting RGB-value. The *reflect* function can be implemented like shown in equation 2.7. The Phong BRDF formula 2.6 is finally applied in line 10 and the resulting RGB value is returned in line 11.

4.3 Spherical Harmonics

The SH based method consists of two steps: In the first step the SH coefficients of an input environment map are calculated (algorithm 4.5 and helper functions in algorithm 4.4). In the second step, irradiance and reflection maps are calculated in SH frequency space by spherical convolution with the Phong BRDF's SH coefficients and then transformed into a texture.

The SH coefficients calculation step can be triggered once the environment map has been captured. The environment map serves as input parameter $hdr[n, n][3]$ to the *calculateShCoeffs* function in algorithm 4.5. It is specified as a two-dimensional square-sized array, representing an HDR texture with side length n. Each element is an array of length three representing the three different RGB color values per pixel. Parameter o represents the SH-order which determines the number of SH basis functions to be used – which are all those on the frequency bands from 0 up to and including $(o - 1)$. The first few frequency bands are illustrated in figure 2.16. Algorithm 4.5 calculates one SH coefficient for each SH basis function, leading to $o^2 \times 3$ coefficients, since each color channel is calculated separately. The parameter $coeffs[o^2][3]$ is an output parameter which will contain the SH coefficients of all $(o - 1)$ bands for all three color channels.

Algorithm 4.5 is our implementation of equation 2.29. We have divided it into two parts: one is represented by the function *calculateShValues*, the second by the function *sumForCoeffs*, both shown in algorithm 4.4. The reason for this is that both of these parts can be implemented in a highly parallel fashion, making them well suited for a GPU accelerated implementation. The instructions in the nested `for`-loop in function *calculateShValues* can be implemented in a straight-forward way in a fragment shader or with a compute framework. Function *sumForCoeffs* is not as well suited for highly parallel execution as *calculateShValues* but applying what is commonly known as the *reduce*-pattern, its run-time could be reduced from the current linear complexity to a logarithmic complexity.

In Algorithm 4.5, we iterate over all SH frequency bands and all its functions in lines 3 and 4. In line 5 we calculate the SH coefficient index c before calling *calculateShValues* with the parameters texture side length n, SH band index l, and SH basis function index m. It returns an array of size $n \times n$ containing the corresponding SH function's values multiplied with the sphere map projection weights. In line 7, the function *sumForCoeffs* is called with the HDR texture as a parameter and the SH values. This calculates the values for the RGB coefficients at index c.

Function *calculateShValues* in algorithm 4.4 computes the values of a specific SH basis function (identified by l and m) for each pixel of the $n \times n$ sized texture. A pixel's corresponding texture coordinates are calculated, brought to a range of $[-1, 1]$, and shifted to the pixel centers in lines 6 and 7. If the respective coordinate lies within the circular region of the sphere map (checked in line 9), the corresponding azimuth and elevation angles are calculated in lines 12 and 13. Since the angle map projection format is used, a weight is calculated based on θ in line 14. Finally, the basis function's value is

Algorithm 4.4: Functions *calculateShValues* and *sumForCoeffs*, used by function *calculateShCoeffs* in algorithm 4.5

```
1  function calculateShValues (n, l, m)
       input  : $n$, $l$, $m$
       output : $shValues[n,n]$
2      $onePxl = 1/n$;                // relative size of one light probe image's pixel
3      $halfPxl = onePxl/2$;          // relative size of a half light probe image's pixel
4      for $i = 0, i = i+1, \ldots, n-1$ do
5          for $j = 0, j = j+1, \ldots, n-1$ do
6              $v = 2 \cdot (i + halfPxl)/n - 1$;  // bring to range $[-1,1]$ and shift to pixel center
7              $u = 1 - 2 \cdot (j + halfPxl)/n$;  // bring to range $[-1,1]$ and shift to pixel center
8              $r = \sqrt{(u^2+v^2)}$;             // distance to the light probe's center
               /* Only regard valid pixels inside the circular area of a light probe image    */
9              if $r > 1$ then
10                 $shValues[i,j] = 0.0$;
11             end
               /* Calculate azimuth and elevation angles corresponding to the current texel:  */
12             $\theta = \pi \cdot r$;
13             $\phi = \mathrm{atan2}(v,u)$;
               /* Calculate the weight of the current texel which depends on both, the texture
                  size and the sphere map projection format (in this case sinc($\theta$) because of the
                  angle map projection format):                                                 */
14             $w = \left(\frac{2\pi}{n}\right)^2 \cdot \mathrm{sinc}(\theta)$;
               /* Calculate the SH basis function's value and multiply it with the projecton
                  format weight                                                                 */
15             $shValues[i,j] = \mathrm{SH}(l,m,\theta,\phi) \cdot w$;
16         end
17     end
18     return shValues;
19 function sumForCoeffs( $hdr[n,n][3]$, $shValues[n,n]$)
       input  : $hdr[n,n][3]$, $shValues[n,n]$
       output : $sum[3]$
20     $sum[3] = 0,0,0$;
21     for $i = 0, i = i+1, \ldots, n-1$ do
22         for $j = 0, j = j+1, \ldots, n-1$ do
23             $c[0] = c[0] + hdr[i,j][0] \cdot shValues[i,j]$;
24             $c[1] = c[1] + hdr[i,j][1] \cdot shValues[i,j]$;
25             $c[2] = c[2] + hdr[i,j][2] \cdot shValues[i,j]$;
26         end
27     end
28     return sum;
```

calculated by calling the *SH* function from algorithm 2.6, multiplied with the weight, and returned.

The function *sumForCoeffs* calculates the values of an SH coefficient for each color channel from a given HDR texture and the previously calculated SH values. It iterates over all $n \times n$ values of the input light probe and multiplies the color values with the corresponding SH values. The coefficients are calculated separately for each color channel red (line 23), green (line 24), and blue (line 25).

Algorithm 4.5: Calculation of a given light probe's SH coefficients

```
1  function calculateShCoeffs( n, hdr[n,n][3], o )
       input  : $n$, $hdr[n,n][3]$, $o$
       output : $coeffs[o^2][3]$
2      $coeffs[o^2][3] = \{(0,0,0),\dots,(0,0,0)\}$;
3      for $l = 0, l = l+1, \dots, l_{max}$ do
4          for $m = -l, m = m+1, \dots, l$ do
5              $c = (l+1) \cdot l + m$;
               /* Computes the SH base function values multiplied by the projection format
                  values:                                                              */
6              $shValues[n,n]$ = calculateShValues($n$, $l$, $m$);
               /* Multiplies the SH-values with the corresponding light probe data in order to
                  compute their contribution to SH coefficient c:                      */
7              $cSum[3]$ = sumForCoeffs($hdr$, $shValues$);
8              $coeffs[c][0] = cSum[0]$;
9              $coeffs[c][1] = cSum[1]$;
10             $coeffs[c][2] = cSum[2]$;
11         end
12     end
13     return $coeffs$;
```

While the first part is about the calculation of the SH-coefficients, the second parts uses them to generate irradiance and reflection maps via spherical convolution, described by equation 2.31 in section 2.4. Algorithm 4.6 shows the implementation of equation 2.33, which calculates the SH coefficients of the Phong BRDF. It can be used for both, irradiance and reflection maps. Parameter s is the specular shininess coefficient. For the irradiance map, s has to be assigned the value 1 in order to get the SH-coefficients corresponding to $\cos^1$. The second parameter for *calcPhongBrdfShCoeff* is the SH band index l. The Phong BRDF SH-coefficients do not depend on m since they belong to the class of zonal harmonics which have only one non-zero coefficient per SH band, as we have described in section 2.4.

Algorithm 4.7, finally, describes the SH-based irradiance and reflection maps calculation. The implementation is tailored to being used in a GPU shader program, which is why the SH coefficients are sampled from a texture and N specular shininess coefficients are calculated in *calculateRadianceSH*, which can be performed in parallel on a GPU.

Algorithm 4.6: Calculate the Phong BRDF coefficients $\Lambda_l \rho_l$.

```
1  function calcPhongBrdfShCoeff( l, s )
       input  : SH level $l$, specular coefficient $s$
       output : SH coefficient for the Phong BRDF on level $l$ and specular coeff. $s$
2      $num = 1$;
3      $den = 1$;
4      $start = 1$; // for odd values of l
5      if $l$ is even then
6          $start = 2$; // for even values of l
7      end
8      for $i = start, i = i + 2, \ldots, l$ do
9          $num = num \cdot (s - i + 2)$;
10         $den = den \cdot (s + i + 1)$;
11     end
12     return $\frac{num}{den}$;
```

The function *calculateRadianceSH* takes as input parameters the SH-order O, the texture containing the spherical harmonics coefficients *shCoeffTexSampler* and its size *shCoeffTexSize*, the number of specular shininess coefficients N, the specular shininess coefficients via the array *specCoeff*$[N]$, the tiling parameters *tileOffset* and *tileScale* if only a part of the whole texture is to be calculated in the respective pass, and the varying texture coordinates $\vec{c}$ in the range $[0, 1]$, addressing the texel centers. The output is an array of length N containing the outgoing irradiance or radiance for each specular coefficient at the current texture position.

Line 3 handles tiled calculation by shifting the texture coordinates. The check in line 7 ensures that calculations are performed only inside the circular region where the sphere map contains illumination information. In lines 10 and 11 the azimuth and elevation angles are calculated that correspond to the texture coordinates $\vec{c}$. In order to sample precisely from the texels of the SH-coefficients texture, the size of one texel is calculated in line 12.

Lines 15 and 16 iterate over all SH basis functions which coefficients are available for – from the first SH band index 0 up to the band index $(O - 1)$ where O is the SH-order. Lines 17 to 20 calcuate the sampling coordinates for the i-th coefficient which we assume to be stored in the texture row-wise in ascending order from left to right. In line 22, the Y_{lm} value is calculated according to equation 2.27, and it is used in line 24 to calculate the respective SH coefficient's contribution to the current texel by calculating equation 2.31 for all color channels. It is added to the final result for the current texel in line 34.

Lines 25 to 33 prevent that `NaN`-values are included in the result. Without these checks, we experienced issues on mobile GPUs that led to incorrect results or the result texture being entirely black. With explicit `NaN` checks we were able to reliably compute the correct results.

Algorithm 4.7: Calculate irradiance and radiance maps via spherical harmonics.

```
function calculateRadianceSH( $\vec{c}$ )
    input  : $\vec{c}$, $O$, $shCoeffTexSampler$, $shCoeffTexSize$, $N$, $specCoeff[N]$, $tileOffset$,
             $tileScale$
    output : $\vec{L_o}[N]$
    $\vec{L_o}[k] = (0,0,0),\dots,(0,0,0)$;        // Initialize outgoing radiance/irradiance values
    $\vec{c_t} = \vec{c} \cdot tileScale + tileOffset$;
    $\mathtt{v} = 2 \cdot \vec{c_t}.t - 1$;
    $\mathtt{u} = 1 - 2 \cdot \vec{c_t}.s$;
    $\mathtt{r} = \sqrt{(\mathtt{u}^2 + \mathtt{v}^2)}$;
    if $r > 1$ then
        /* Outside of valid sphere map texture area                                  */
        return $\vec{L_o}$;
    end
    /* Within valid sphere map texture area                                          */
    $\theta = \pi \cdot \mathtt{r}$;
    $\phi = \mathrm{atan2}(\mathtt{v}, \mathtt{u})$;
    /* SH coefficients are sampled from a texture => calculate the size of one texel:  */
    $oneTxl = 1/shCoeffTexSize$;          // size of one texel of the SH-coeffs texture
    $halfTxl = oneTxl/2$;                 // size of a half texel of the SH-coeffs texture
    $l_{max} = O - 1$;
    for $l = 0..l_{max}$ do
        for $m = -l..l$ do
            $i = (l+1) \cdot l + m$; /* Calculate the coordinates for the i-th coefficient:  */
            $\mathtt{d} = i/shCoeffTexSize$;
            $t = oneTxl \cdot \mathtt{d}$;
            $s = oneTxl \cdot (i - shCoeffTexSize \cdot \mathtt{d})$;
            $\vec{L_i} = \mathrm{sampleTexture}(shCoeffTexSampler, s, t)$;
            $Y_{lm} = \mathrm{SH}(l, m, \theta, \phi)$;
            for $k = 1..N$ do
                $\vec{L_t} = \vec{L_i} \cdot \mathrm{calcPhongBrdfShCoeff}(l, specCoeff[k]) \cdot Y_{lm}$;
                if $\vec{L_t}.r$ is NaN then
                    $\vec{L_t}.r = 0$;
                end
                if $\vec{L_t}.g$ is NaN then
                    $\vec{L_t}.g = 0$;
                end
                if $\vec{L_t}.b$ is NaN then
                    $\vec{L_t}.b = 0$;
                end
                $\vec{L_o}[k] = \vec{L_o}[k] + \vec{L_t}$;
            end
        end
    end
    return $\vec{L_o}$;
```

CHAPTER 5

Realistic Rendering in Augmented Reality

In order to increase visual coherence and the realism of AR rendering results, we are proposing to combine image-based lighting using irradiance and reflection maps (described in section 5.3.1) with the real-time rendering effects light mapping (described in section 5.3.2), normal mapping (described in section 5.3.3), and the obligatory tone mapping (described in section 5.3.4) to properly process high dynamic range image data to display it on the low dynamic range display of a mobile device.

5.1 Mobile Application Structure

We have developed a mobile application for the operating system Android and included all of the methods and techniques mentioned above and described in the previous chapters. As graphics-API we have used OpenGL ES 3.0 [Khr14]. An Android application can be organized by dividing it into separate units which are called *Activities*. Our application consists of two such units: The environment capturing Activity, and the rendering Activity. In the environment capturing Activity, the technique described in chapter 3 is implemented. After a user has captured an HDR environment map like the one shown in figure 3.2, it can be stored to a file and passed to the rendering Activity. The rendering Activity loads the captured HDR environment map, upload it to the GPU and immediately perform the SH pre-processing. After the initialization work has completed, the rendering Activity constantly looks for the tracking target via Vuforia AR tracking framework[14], renders the virtual objects atop the camera image if the tracking target is detected by Vuforia, and provides the options to start computing the accurate irradiance and reflection maps, to start computing the SH irradiance and reflection maps,

[14]Vuforia Developer Portal, PTC Inc., `https://developer.vuforia.com`

or to switch between the already computed maps of the different methods: accurate, MIP-mapping, or SH. The steps of our application's render-loop, which are executed in each frame, are like follows:

1. Get pose matrix from AR tracker
2. Update transformations of objects, camera, etc.
3. If the accurate method or the SH method is in progress or has been triggered, calculate one tile of the irradiance and reflection maps (depending on configuration).
4. Render video background
5. Render 3D models and a mask for tone mapping
6. Apply tone mapping to the rendered objects
7. Render the user interface

These are common steps for a real-time rendering application. Updates of game object transformations have to be applied before rendering in order to prevent a delay between the transformations and the rendering. Since this is an AR application, the updates depend on the pose matrix which we get from the AR tracking framework in our application. We use the pose matrix as view matrix when rendering the virtual objects.

Step 3 is an optional step which is only active if an irradiance/reflection map calculation is currently in progress or has been triggered by the user in the current frame. We usually only calculate a part of one or multiple maps in this step to retain interactive frame rates during the lengthy calculations. How much work is done in step 3 can be configured in our application by setting a tile size. For input environment maps sized 512×512 or 1024×1024, we usually set this tile size to 64×64 or to 128×128. Larger tile sizes can introduce a high delay in step 3 for accurate calculations for SH calculations depending on the SH-order.

In step 4, the Vuforia framework is instructed to render the video background. Since the video background is already LDR, we not only render the virtual objects fully shaded to a frame-buffer in step 5 but also a monochrome mask of these virtual objects to another frame-buffer. We use this mask to apply tone mapping only to the virtual objects which are illuminated with HDR irradiance and reflection maps (i.e. to all frame buffer regions where the mask is white).

5.2 Augmented Reality Setup

Our application is split into two parts:

1. Capture a light probe from the current device's position.
2. Compute irradiance and reflection maps from the captured light probe and use them for shading 3D objects rendered atop the real-world camera image.

In the first step, an omni-directional image of the environment is captured like described in chapter 3. The origin of this omni-directional image is at the position where the user starts to capture the environment. Ideally, this should correspond with the origin of the virtual scene rendered in the second step, otherwise errors are introduced due to the spatial offset.

In the second part of our application, an image target is used to define the origin of our scene. Once the image target is recognized by Vuforia tracking SDK, we get a pose matrix and use it to position the virtual 3D objects relative to the origin of the image target.

When the user changes from part one to part two, there is be a waiting time due to a mandatory SH pre-calculation step which computes the SH coefficients of the environment map captured in step 1. The duration varies with different SH-orders. After the pre-calculation step, we do not automatically trigger accurate and SH irradiance and radiance map calculation to avoid further waiting time. Instead, the user can already use the AR application in real-time and look for the tracking target. At this point, only the MIP-mapping based irradiance and reflection maps are available since they can be calculated in real-time. Accurate maps and SH maps calculation can be triggered manually by the user. These calculations imply a significant amount of work and can take up to several minutes to complete. They are performed in step 3 of our application's render-loop and cause the application to slow down to interactive frame rates until the calculations have completed.

5.3 Real-Time Rendering Techniques

In the AR real-time rendering part of our application, we join multiple real-time rendering techniques. We use IBL with irradiance and reflection maps, generated from an environment map containing real-world illumination, for visually coherent shading of virtual objects blended with real-world images. As pointed out by Kán and Kaufmann in [KDB+14], adding further illumination effects usually helps to increase the sense of presence in an AR scene and increases visual coherence. We have chosen to add the illumination effects *light mapping* (section 5.3.2), *normal mapping* (section 5.3.3), and *tone mapping* (section 5.3.4) to our application.

5.3.1 Image-Based Lighting with Irradiance and Reflection Maps

IBL rendering with irradiance and reflection maps requires sampling a given map with a direction vector and using the resulting color in illumination calculation for the current fragment. Irradiance maps are sampled using the surface normal while reflection maps are sampled using the reflection vector (see figures 2.11, and 2.12).

The implementation is similar to algorithm 4.3, but we do not sample a specific MIP-map level but instead we use standard image sampling. Algorithm 5.1 shows the implementation.

Algorithm 5.1: Fragment shader implementation for image-based lighting with the Phong-BRDF, using an irradiance map and a reflection map.

```
   input  : irrTexSampler, reflTexSampler, $k_d$, $k_s$
   output : $\vec{L_o}$
1  function shade( $\vec{n}$, $\vec{v}$ )
       /* Calculate sphere map coordinates:                         */
2      $\vec{c_d}$ = sphereMappingAngleMap( $\vec{n}$ );
3      $\vec{r}$ = reflect( $\vec{v}, \vec{n}$ );
4      $\vec{c_s}$ = sphereMappingAngleMap( $\vec{r}$ );
       /* Sample illumination values from the irradiance and
          from the reflection map:                                  */
5      $\vec{L_d}$ = texture(irrTexSampler, $\vec{c_d}$);
6      $\vec{L_s}$ = texture(reflTexSampler, $\vec{c_s}$);
       /* Calculate illumination via Phong BRDF:                    */
7      $\vec{L_o} = k_d \cdot \vec{L_d} + k_s \cdot \vec{L_s}$;
8      return $\vec{L_o}$;
```

The inputs for algorithm 5.1 are the side length of the square-sized input texture N, two texture samplers *irrTexSampler* and *reflTexSampler* to access the irradiance map and the appropriate reflection map for the material's specular coefficient, the material's Phong coefficients k_d and k_s, and per fragment the varying parameters surface normal $\vec{n}$ and current view vector $\vec{v}$. In lines 2 and 4 the texture coordinates for texture lookup are calculated. The resulting 2D vectors $\vec{c_d}$ and $\vec{c_s}$ are used for texture sampling in lines 5 and 6. The diffuse lighting value is assigned to the 3D vector $\vec{L_d}$ as RGB-value, and the specular lighting RGB-value is assigned to the 3D vector $\vec{L_s}$. The *reflect* function can be implemented like shown in equation 2.7. The Phong BRDF formula 2.6 is finally applied in line 7 and the resulting RGB value is returned in line 8.

Kronander [KBG$^+$15] created an in-depth survey and provides classification for AR-related methods. Lighting conditions for AR can be divided into two categories: measured lighting conditions and estimated lighting conditions. Our technique falls under the first of these two categories since capturing real-world illumination in the form of a light probe means capturing a physically accurate model of the lighting conditions in a scene and recreating it in the virtual scene [KBG$^+$15].

5.3.2 Light Mapping

Light mapping is a technique to add static lighting or shadows to the shading of a virtual object. In our application, we use pre-calculated light maps. Light maps can be calculated offline and high-quality rendering methods can be used. During shading, the static lighting information is sampled from the light map. Since light maps have no directional dependence, they can only be used for the diffuse part of the surface [AMHH08].

Theoretically, the lighting information in a light map could be combined with the diffuse texture of an object in a single texture, there are many practical reasons against this approach: Color information can change with high frequency across a surface of an object. Contrary, diffuse light information often varies slowly over the surface. By splitting color information from lighting information, an appropriate texture resolution can be chosen for each to store the data in an appropriate precision. While color information can often be reused across a surface (e.g. when using a tile-able texture to render grass or walls), light information can usually not be reused easily. Furthermore, splitting enables light information to be treated separately from the color information by making its contribution lighter or stronger globally or in a certain region (e.g. when simulating the effect of illuminating a light mapped wall with a beam of light from a torch) [AMHH08].

Algorithm 5.2 shows our implementation of light mapping combined with irradiance and reflection mapping from algorithm 5.1.

The inputs for algorithm 5.2 are the same as for algorithm 5.1 with an additional light map texture sampler *lmTexSampler* and the varying light map texture coordinates $\vec{c_{lm}}$ which are used to sample from *lmTexSampler* in line 7. In line 8, the static diffuse light information $\vec{i_{lm}}$ is used to modify the diffuse irradiance value $\vec{i_d}$ by a simple component-wise multiplication.

Figure 5.1 shows the effects of light mapping in an AR scene. The virtual dragon model is shown with and without a pre-calculated ambient occlusion light map. The light map contributes to a realistic look and helps to improve the perception of the dragon's spatial alignment.

Algorithm 5.2: Fragment shader implementation for image-based lighting with Phong-BRDF, irradiance and reflection mapping combined with light mapping.

```
input  : $irrTexSampler$, $reflTexSampler$, $lmTexSampler$, $k_d$, $k_s$
output : $\vec{L_o}$
1  function shade( $\vec{n}$, $\vec{v}$, $\vec{c_{lm}}$ )
       /* Calculate sphere map coordinates:                                  */
2      $\vec{c_d}$ = sphereMappingAngleMap( $\vec{n}$ );
3      $\vec{r}$ = reflect( $\vec{v}, \vec{n}$ );
4      $\vec{c_s}$ = sphereMappingAngleMap( $\vec{r}$ );
       /* Sample illumination values from the irradiance and
          from the reflection map:                                           */
5      $\vec{L_d}$ = texture($irrTexSampler$, $\vec{c_d}$);
6      $\vec{L_s}$ = texture($reflTexSampler$, $\vec{c_s}$);
       /* Sample static diffuse lighting information from the
          light map and modify diffuse illumination.                         */
7      $\vec{L_{lm}}$ = texture($lmTexSampler$, $\vec{c_{lm}}$);
8      $\vec{L_d} = \vec{L_d} \cdot \vec{L_{lm}}$;
       /* Calculate illumination via Phong BRDF:                             */
9      $\vec{L_o} = k_d \cdot \vec{L_d} + k_s \cdot \vec{L_s}$;
10     return $\vec{L_o}$;
```

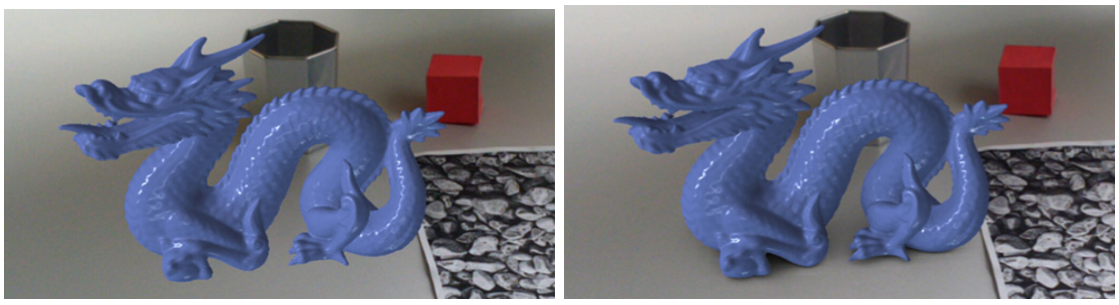

(a) (b)

Figure 5.1: Rendering of an AR scene showing a virtual dragon, a real cup, and a real cube. In the image (a), the virtual dragon is rendered without ambient occlusion light maps, whereas in image (b), the dragon is rendered with static ambient occlusion light maps which were pre-calculated in an offline high-quality ambient occlusion calculation [KUK15]. Adapted from [KUK15].

5.3.3 Normal Mapping

Normal mapping is a technique to add small-scale detail to a virtual object during shading. Normal mapping defines a surface normal per fragment using information fetched from a texture. As described by Akenine-Möller et al., normal mapping techniques are modifying perceived detail on the meso-geometry level, which contains detail that is too complex to be efficiently rendered using geometry, but is large enough that it can not be modelled by a shading model at the micro-geometry level [AMHH08].

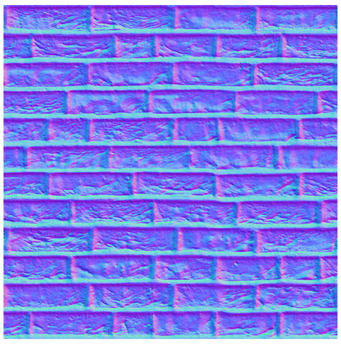


Figure 5.2: A normal map texture in tangent space which can be used to render a flat surface with the appearance of a brick wall on the meso-geometry level. Reprinted from[15].

While normal maps can be stored in world space or in object space, it is most practical to store them in tangent space [AMHH08]. Figure 5.2 shows how a normal map stored in tangent space looks like. Each color channel of a pixel is a surface normal coordinate of a normalized normal. To store a normal map in a texture, these coordinates, however, are transformed by equation 5.1 to map them from range $[-1, 1]$ to the range $[0, 1]$. This enables them to be stored in common 8-bit textures which can only store integer values between 0 and 255 mapped to the range $[0, 1]$.

$$\vec{n_t} = \left(\frac{\vec{n}_x}{2} + 0.5 \ , \ \frac{\vec{n}_y}{2} + 0.5 \ , \ \frac{\vec{n}_z}{2} + 0.5 \right) \tag{5.1}$$

The inverse transformation from $\vec{n_t}$ to $\vec{n}$ is given by equation 5.2.

$$\vec{n} = (2\,\vec{n}_{tx} - 1 \ , \ 2\,\vec{n}_{ty} - 1 \ , \ 2\,\vec{n}_{tz} - 1) \tag{5.2}$$

Advantages of normal maps in tangent space include that they are versatile, can be re-used and have better compression properties compared to normal maps stored in other spaces due to the usually reduced range of the z component [AMHH08].

During shading, a normal is sampled from the normal map texture. In order to calculate the lighting, all relevant vectors have to be transformed into the same space. In our

[15]OpenGameArt.Org, rubberduck, 154_norm.jpg, `http://opengameart.org/node/21131`

application, we use irradiance and reflection maps in world space and also a view vector in world space. Therefore, we transform the sampled normal direction into world space by constructing a transformation matrix which transform from tangent space into world space like follows: The surface tangent $\vec{t_o}$, surface bitangent $\vec{b_o}$ and surface normal $\vec{n_o}$ are transformed from object space into world space (t_w, b_w, and n_w) via the normal transformation matrix N_m. In world space, they form a coordinate system and can be used to construct the matrix shown in 5.3, which transforms from tangent space into world space.

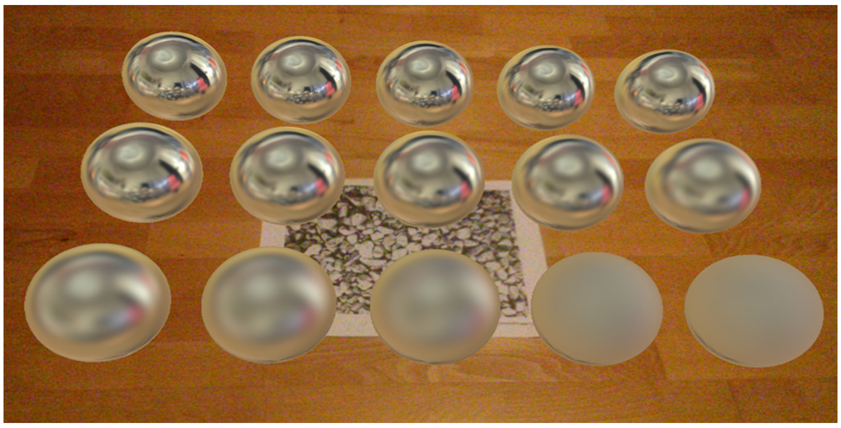
$$\begin{pmatrix} t_{w_x} & b_{w_x} & n_{w_x} & 0 \\ t_{w_y} & b_{w_y} & n_{w_y} & 0 \\ t_{w_z} & b_{w_z} & n_{w_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

Any vector of those three, $\vec{t_w}$, $\vec{b_w}$, and $\vec{n_w}$, can be calculated via the the cross product of the other two. Therefore, it is sufficient to transmit only two vectors from the vertex shader to the fragment shader and calculate the third there.

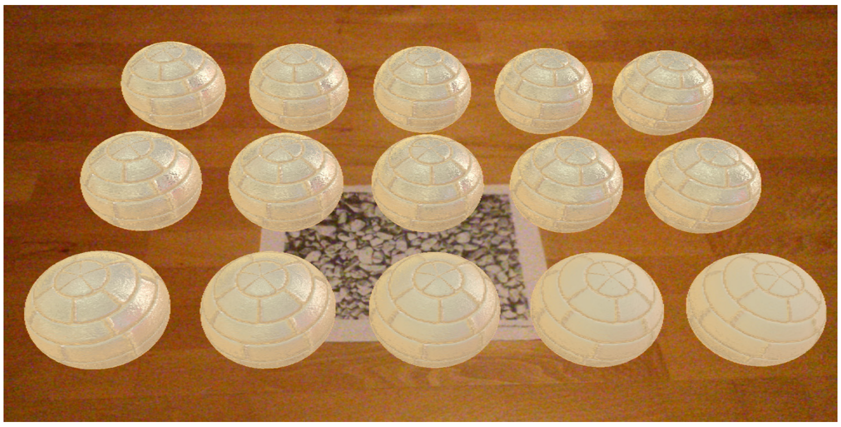
Algorithm 5.3: Vertex shader implementation which calculates tangent, bitangent and normal vectors in world space and passes them on to a fragment shader where they can be used for normal mapping.

```
input  : $M_m$, $M_v$, $M_p$, $N_m$
output : $\vec{p_o}$, $\vec{n_W}$, $\vec{t_W}$
1 function vert( $\vec{p_o}$, $\vec{t_o}$, $\vec{n_o}$ )
      /* Transform into world space using the normal matrix  */
2     $\vec{n_W} = \mathtt{mat3}(N_m) \cdot \vec{n_o}$;
3     $\vec{t_W} = \mathtt{mat3}(N_m) \cdot \vec{t_o}$;
      /* It is sufficient to transform only two vectors since
         the third can be calculated by the cross product    */
4     $\vec{b_W} = \vec{n_W} \times \vec{t_W}$;
      /* Calculate the vertex position and pass the varying
         values down the graphics pipeline                    */
5     $\vec{p_p} = M_p \cdot M_v \cdot M_m \cdot \vec{p_o}$;
6     return $\vec{p_p}, \vec{n_W}, \vec{t_W}$;
```

Algorithms 5.3 and 5.4 show how normal mapping can be integrated into our image based lighting technique. The vertex shader gets as input parameters the vertex' position, its tangent, and normal, each in object space. The tangent, bitangent, and normal form a basis to transform vectors into tangent space. By transforming this basis into world space (lines 2 and 3 in algorithm 5.3), we can create a basis to transform from tangent space into world space as described above. The vertex shader gets several matrices as input: M_m is the model matrix, M_v is the view matrix, M_p is the projection matrix, and N_m is the



(a)



(b)

Figure 5.3: These images illustrate how strongly normal mapping can change the appearance of an object. The geometry of the spheres is exactly the same in both images. Whereas in image (a) plain IBL is applied using different reflection maps for the various spheres, image (b) uses the same shading with only one modification: The normal is perturbed per fragment using a normal map similar to the one shown in figure 5.2.

Algorithm 5.4: Fragment shader implementation for image-based lighting with the Phong-BRDF, using an irradiance map and a reflection map.

```
   input  : $irrTexSampler$, $reflTexSampler$, $nmTexSampler$ $k_d$, $k_s$
   output : $\vec{L_o}$
1  function $shade( \vec{n_W}, \vec{t_W}, \vec{v}, \vec{c_{nm}} )$
       /* Sample the normal from the normal map:                      */
2      $\vec{n_T}$ = texture( $nmTexSampler$, $\vec{c_{nm}}$ ) ; // get normal in tangent space
3      $\vec{n_T} = \vec{n_T} \cdot 2 - 1$ ;             // transform using equation 5.2
       /* Get or calculate the base vectors for the matrix to
          transform from tangent space into world space:              */
4      $\vec{n_W}$ = normalize( $\vec{n_W}$ );
5      $\vec{t_W}$ = normalize( $\vec{t_W}$ );
6      $\vec{b_W} = \vec{n_W} \times \vec{t_W}$;
       /* Construct the matrix from the three base vectors:           */
7      $M_{TW}$ = mat3($\vec{t_W}$, $\vec{b_W}$, $\vec{n_W}$);
       /* Transform the normal from tangent space into world
          space:                                                      */
8      $\vec{n} = M_{TW} \cdot \vec{n_T}$;
       /* The remaining code is just the same as in 5.1:              */
9      $\vec{c_d}$ = sphereMappingAngleMap( $\vec{n_W}$ );
10     $\vec{r}$ = reflect( $\vec{v}$,$\vec{}$);
11     $\vec{c_s}$ = sphereMappingAngleMap( $\vec{r}$ );
12     $\vec{L_d}$ = texture($irrTexSampler$, $\vec{c_d}$);
13     $\vec{L_s}$ = texture($reflTexSampler$, $\vec{c_s}$);
14     $\vec{L_o} = k_d \cdot \vec{L_d} + k_s \cdot \vec{L_s}$;
15     return $\vec{L_o}$;
```

normal matrix to transform normals into world space. A normal is usually transformed by taking the upper left 3×3 part of the inverse transposed transformation matrix. In code, we denote this as `mat3(M4by4)` with `M4by4` being an arbitrary 4×4 matrix. Per vertex, the shader receives the parameters $\vec{p_o}$ which is the vertex position in object space, and $\vec{t_o}$, $\vec{b_o}$, and $\vec{n_o}$ which are the tangent, bitangent, and normal vectors, respectively. As varying output parameters, the vertex shader returns the position transformed into projection space, and two of the tangent space basis vectors to the fragment shader.

The fragment shader implementation in algorithm 5.4 is based on the implementation in 5.1 but has as an additional parameter the normal map texture sampler *nmTexSampler*. As varying input parameters, it receives the view vector $\vec{v}$, two of the base vectors for transforming from tangent space to world space $\vec{n_W}$ and $\vec{t_W}$, and texture coordinates $\vec{c_{nm}}$ to sample from the normal map which is done in line 2. In line 3 equation 5.2 is applied to bring all components of the normal to the range $[-1, 1]$. In lines 4 and 5, two of the basis vectors are retrieved and used to calculate the third via their cross product in line 6. Matrix 5.3 is constructed in line 7 and used to transform the normal n_T, which was sampled from the normal map texture, from tangent space into world space in line 8. The remaining code is exactly the same as in 5.1 since only the normal has been modified by normal mapping.

5.3.4 Tone Mapping

The irradiance and reflection maps which we capture like described in chapter 3 consist of high dynamic range (HDR) values. We are utilizing floating point textures to store them. Furthermore, we use floating point framebuffers which the virtual objects are rendered into, shaded via IBL using the aforementioned irradiance and reflection maps.

In the final stage of our application's rendering pipeline we need to convert the HDR values in our framebuffer to a low dynamic range (LDR) in order to display them on the smartphone's or tablet's LDR screen.

The idea behind a *tone mapping operator* is to map a HDR range to the LDR range of $[0, 1]$ so that all areas become visible. A good tone mapping operator takes the human visual system's logarithmic response to light intensities into account. An overview of tone mapping operators can be found in [Luk07]. One of them is the *Reinhard's operator*, introduced by Reinhard et al. in [RSSF02]. We have implemented a simplified version of it.

The Reinhard's operator requires the average scene luminance $\overline{L}_w$. While this should be calculated as the log-average luminance over all pixels as described in [RSSF02], we are simplifying this step in order to save performance. In our implementation we take the single value of the framebuffer's highest MIP-map level as the value for $\overline{L}_w$ which is the average luminance. Calculating the log-average luminance would require another render pass, but we get the average luminance value for minimal performance cost since MIP-mapping is implemented in hardware.

Reinhard et al. describe two versions of their tone mapping operators. The first is given by equation 5.4, and the second is given by equation 5.5,

$$L_d(x, y) = \frac{L(x, y)}{1 + L(x, y)} \tag{5.4}$$

$$L_d(x, y) = \frac{L(x, y)\left(1 + \frac{L(x,y)}{L^2_{\text{white}}}\right)}{1 + L(x, y)} \tag{5.5}$$

where $L_d(x, y)$ is the tone mapped luminance value, L_{white} is the smallest luminance that is mapped to pure white, and $L(x, y)$ is the scaled luminance given by equation 5.6,

$$L(x, y) = \frac{a}{\overline{L}_w} L_w(x, y) \tag{5.6}$$

where $L_w(x, y)$ is the outgoing luminance resulting from previous IBL rendering for pixel (x, y), $\overline{L}_w$ is the average scene luminance as stated above, and a is the so-called "key value" which can be varied depending on the average luminance [RSSF02]. Figure 5.4 shows the effects of different key values.

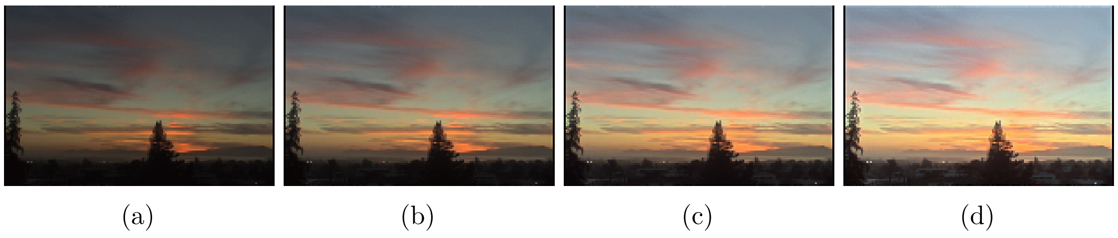


Figure 5.4: These images show the effects of different key values used in equation 5.6 as parameter a. Image (a) has been generated with a key value of 0.09, (b) with a key value of 0.18, (c) with a key value of 0.36, and (d) with a key value of 0.72. All images are reprinted from [RSSF02].

The second version of Reinhard's operator (equation 5.5) allows high luminance values above a certain threshold to be mapped to 1 whereas the first version (equation 5.4) brings all luminance values within displayable range [RSSF02]. We have included a variation of equation 5.5 in our technique. Algorithm 5.5 shows our fragment shader implementation.

The inputs for algorithm 5.5 are the color framebuffer texture sampler *texSamplerIllu*, a texture sampler for a mask texture *texSamplerMask*, the texture size of the color texture *texSize*, the key value of Reinhard's operator a, L_{white}, and the varying texture coordinates $\vec{c}$.

The mask texture is often used in AR applications and we use it for applying tone mapping only to the regions where the color framebuffer contains HDR values. The video background is rendered in LDR into the color framebuffer, therefore we only need to tone map the pixels which contain HDR values from image-based lighting like described in section 5.3.1. The monochrome mask texture is sampled in line 4 of algorithm 5.5 and it contains only the values 0 or 1.

The sampled color value $\vec{C_w}$ is converted into a luminance value in line 3 by the dot product with the vector $(0.299, 0.587, 0.114)$ which roughly corresponds to the ratio how sensitive the human eye is to the different color channels red, green, and blue.

The average luminance value is sampled from the highest MIP-map level of the color framebuffer and calculated in line 6. Reinhard's operator is applied in line 8. Line 9 ensures that tone mapping is only applied to fragments with a corresponding mask value of 1.

Algorithm 5.5: Fragment shader implementation based on a simplified version of Reinhard's tone mapping operator like given by equation 5.5.

```
  input  : $texSamplerIllu$, $texSamplerMask$, $texSize$, $a$, $L_{\text{white}}$
  output : Tone mapped LDR pixel color
1 function $toneMapping(\vec{c})$
      /* Sample the world illumination and the mask value $m$: */
2     $\vec{C_w} = texture(texSamplerIllu, \vec{c})$;
3     $\vec{L_w} = \text{dot}(\vec{C_w}, (0.299, 0.587, 0.114))$;
4     $m = texture(texSamplerMask, \vec{c})$;          // will be 0 or 1 always
      /* Get the average luminance from the highest MIP-map level: */
5     $l_{\max} = \log_2(texSize)$;
6     $\overline{L}_w = textureLod(texSamplerMask, (0.5, 0.5), l_{\max}) \cdot (0.299, 0.587, 0.114)$;
      /* Calculate the scaled luminance value: */
7     $L_{xy} = a \cdot L_w / \overline{L}_w$;
      /* Calculate the tone mapped value: */
8     $L_d = L_{xy}\left(1 + L_{xy}/L^2_{\text{white}}\right)/(1 + L_{xy})$;
      /* Apply tone mapping only if $m = 1$ */
9     return $\vec{C_w} \cdot (1 + (L_d - 1) \cdot m)$;
```

5.4 Realistic Rendering on Mobile Devices

The novelty of our application is that all steps can be executed on a single mobile device: from capturing the full sphere of light, to calculating multiple irradiance and reflection maps, and using the calculated maps for realistic image-based lighting. This, however, poses several challenges on the implementation.

The main limitation on mobile devices is their relatively low computation power compared to desktop computers. The SH pre-computation step and the calculation of accurate or SH maps require a high number of instructions and need up to several minutes to complete, depending on the configuration. We have implemented the calculation of the maps in tiles over multiple frames where only one tile is calculated in a frame. The reasons for implementing this feature are two-fold: Firstly, it enables us to maintain interactive frame rates during the calculations, which improves the user experience. Secondly, there are technical reasons. On our test devices, we observed that more than a certain number of GPU instructions executed within a single frame causes the application to crash. By splitting the maps into tiles and only creating one at a time, we were able to reliably avoid this behavior.

At the time of writing, the graphics APIs OpenGL ES 1 and 2 were the most widely supported standards on mobile devices. There was also a wide support for OpenGL ES 3 and most SoC manufacturers supported this graphics API in their most recent chips. Since OpenGL ES 3 has some important advantages compared to OpenGL ES 2, we defined OpenGL ES 3 as a minimum requirement for our application. For our application, the most important of the new features in OpenGL ES 3 are the support for four or more render targets, filtered 16-bit floating point textures and full support for integer and floating point operations in the shading language [Lip13]. These enabled us to implement the tiled maps calculation algorithms very efficiently.

While we could have used OpenGL ES for the SH pre-calculation step as well, we decided to use a different framework which is provided by the Android platform: *RenderScript*[16]. RenderScript is a compute framework similar to CUDA or OpenCL for general compute tasks. The code is organized in so-called kernels which can be programmed in a C99-derived language. If such a kernel is run on a device, it will be executed on the device's GPU if it supports it. Otherwise, the kernel will be executed on the CPU. The main reason for choosing RenderScript was its easy setup and neat integration in the Android platform. Our SH pre-calculation code runs reliably on the GPU in release mode and thus achieves high performance, while in debug mode RenderScript executes our kernels on the CPU to enable step-by-step debugging.

[16]RenderScript API Guide, Android Developers, Google Inc., `https://developer.android.com/guide/topics/renderscript/compute.html`

CHAPTER 6

Results and Optimizations

The three different methods which we have described in chapter 5 have very different characteristics. The accurate method calculates the accurate solution with brute force. The MIP-mapping based method is very fast but calculates only an approximation to the accurate solution. The SH-based method performs the calculations in frequency space with a finite number of SH basis functions. In this chapter, we compare the running times and visual results of the different methods. We can control the results of the SH method by calculating them with different SH-orders, which has an impact on the running time and the visual results. The accurate method can be adjusted by applying a bounds optimization. We also sketch an approach to vary performance between the accurate method and the MIP-mapping based method.

6.1 Results

We have implemented all three methods described in chapter 4 in a mobile application: the accurate computation, the MIP-mapping based approximation, and the SH-based method. All of these methods have different characteristics, advantages and shortcomings. In this chapter, the resulting irradiance and reflection maps from each method are shown and analyzed.

We have deployed our application to a Nvidia Shield tablet and used it to calculate the results. While the visual results are the same on other devices, the computation times may vary across devices with different SoC. We chose the Nvidia Shield tablet for our experiments because it fully supports OpenGL ES 3.0 and various compute frameworks including RenderScript. The device supports the GPU-accelerated version of the latter which means that all relevant computations are executed on the GPU.

We use the four environment maps shown in figure 6.1 to analyze the three different methods in terms of quality. The computation times do not depend on the contents of light probes but only on their sizes. We used textures of size 512 × 512 pixels for our evaluations.

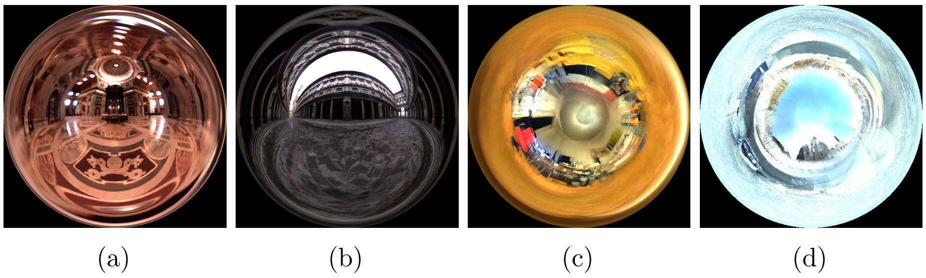


Figure 6.1: The light probes which we use to analyze our methods: The light probe "St. Peters" by Debevec adapted from [Deb01] shown in image (a), the light probe "Uffizi" by Debevec adapted from [Deb01] shown in image (b), the light probe "Room" captured with our app on a Nvidia Shield tablet in image (c), and likewise the light probe "Karlsplatz" shown in image (d) captured with our app on a Nvidia Shield.

Figures 6.2 and 6.3 show examples how irradiance and reflection maps of varying glossiness can be used in rendering. The appearance of the knight's armor can be altered by only changing the irradiance or reflection maps of the metallic material. We adapted a shader described in [Lam13] for these renderings and used the "Challenger" 3D model[17].

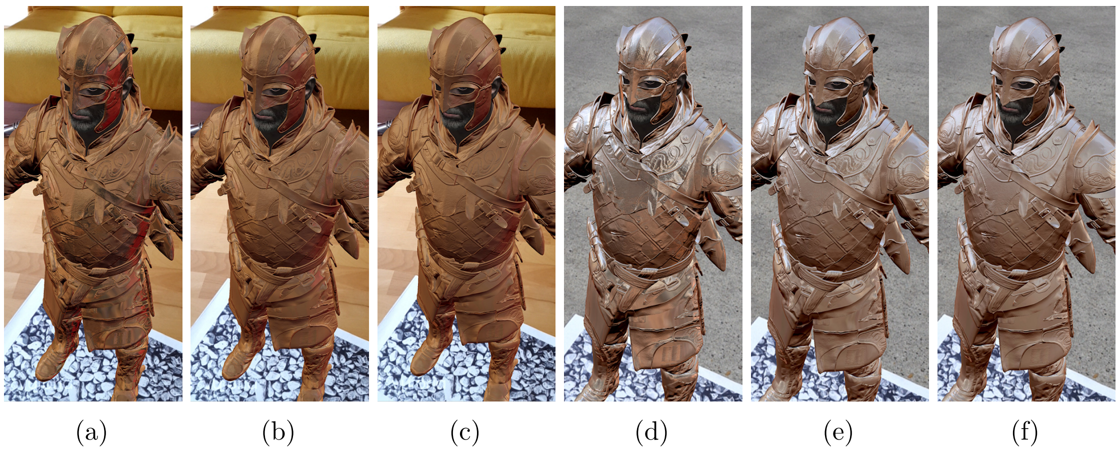


Figure 6.2: AR rendering results of a knight model[16] with metallic IBL material which uses different reflection maps generated with the accurate method. The maps used in images (a), (b), and (c) were generated from the "Room" base map. The maps used in images (d), (e), and (f) were generated from the "Karlsplatz" base map.

[17]3D Model *Challenger* by Unity Technologies, *The Blacksmith: Characters*, Unity Asset Store, http://u3d.as/hjn

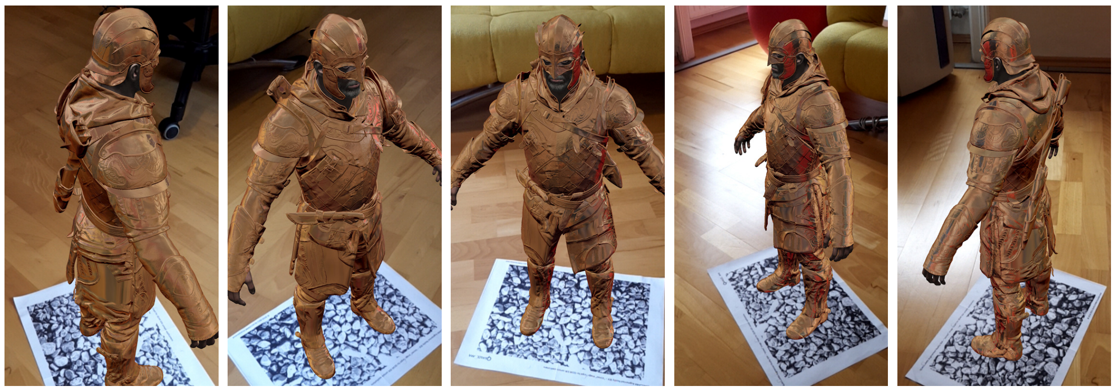


Figure 6.3: AR rendering results of a knight model[16] with a metallic IBL material, viewed from different perspectives. The irradiance and reflection maps were generated from the "Room" base map. The reflections of some real objects in the room can be seen well. For instance, the yellow couch and the red exercise ball can be seen as rather faint reflections in image (a). The more intense red regions in images (b), (c), (d), and (e) are reflections of a red chair.

Figures 6.4, 6.5, 6.6, and 6.7 show the results of calculated irradiance and reflection maps with the different methods: accurate, MIP-mapping, and spherical harmonics with an SH-order of 35. The accurate results are shown in the middle row in each of them. This helps comparing the accurate results to the results of the other two methods: MIP-mapping in the top row, and SH in the bottom row.

As expected, the MIP-mapping results are very inaccurate. This can be explained by the fact that they are not based on the Phong BRDF but are only a very crude approximation of the accurate results, created by averaging neighbouring pixel values. Especially with low specular powers, there is quite a high error, while for high specular powers, the error can get very small.

The SH results show oppositional characteristics. The results are very close to the accurate results for specular shininess coefficients up to 160 while for high specular shininess coefficients very obvious artifacts can occur. The results for low specular shininess coefficients are practically indistinguishable from the accurate results. There are no visual differences for all light probes up to specular shininess coefficients of 80. The results for specular shininess coefficients 320 and 1280 show very obvious ringing artifacts for the maps "St. Peters" and "Uffizi". Interestingly, the maps which we have captured with our app on a mobile device, "Room" and "Karlsplatz", are still very close to the accurate results, even for high specular shininess coefficients. Compared to the maps "St. Peters" and "Uffizi", the captured maps appear to have fewer high-frequency image details like sharp edges and appear blurrier, which is an explanation for this bahavior.

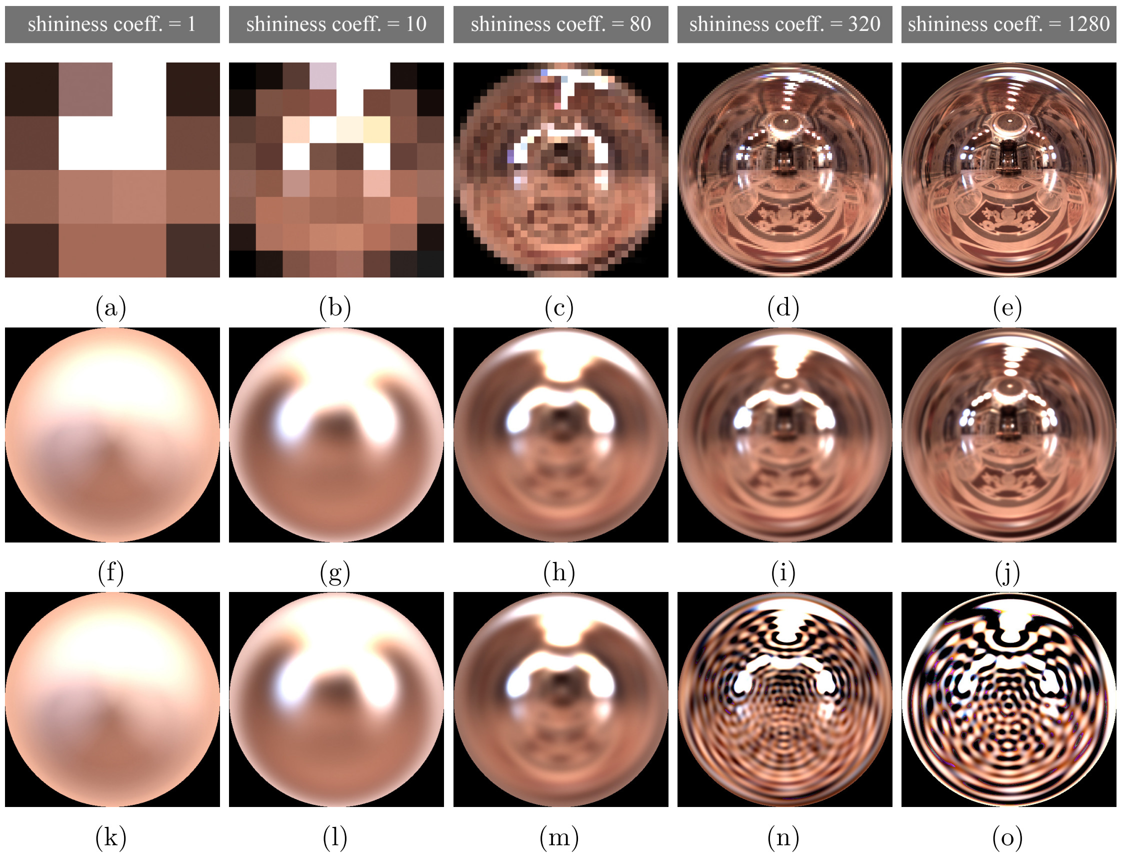


Figure 6.4: Results generated with our three different methods: MIP-mapping (top row), accurate (middle row), and SH (bottom row) for the “St. Peters” light probe. The base map is provided by Debevec [Deb01]. The irradiance maps are shown in the first column: (a), (f), (k); the reflection maps for specular shininess coefficient 10 are shown in the second column: (b), (g), (l); the reflection maps for specular shininess coefficient 80 are shown in the third column: (c), (h), (m); the reflection maps for specular shininess coefficient 320 are shown in the fourth column: (d), (i), (n); and the reflection maps for specular shininess coefficient 1280 are shown in the fifth column: (e), (j), (o).

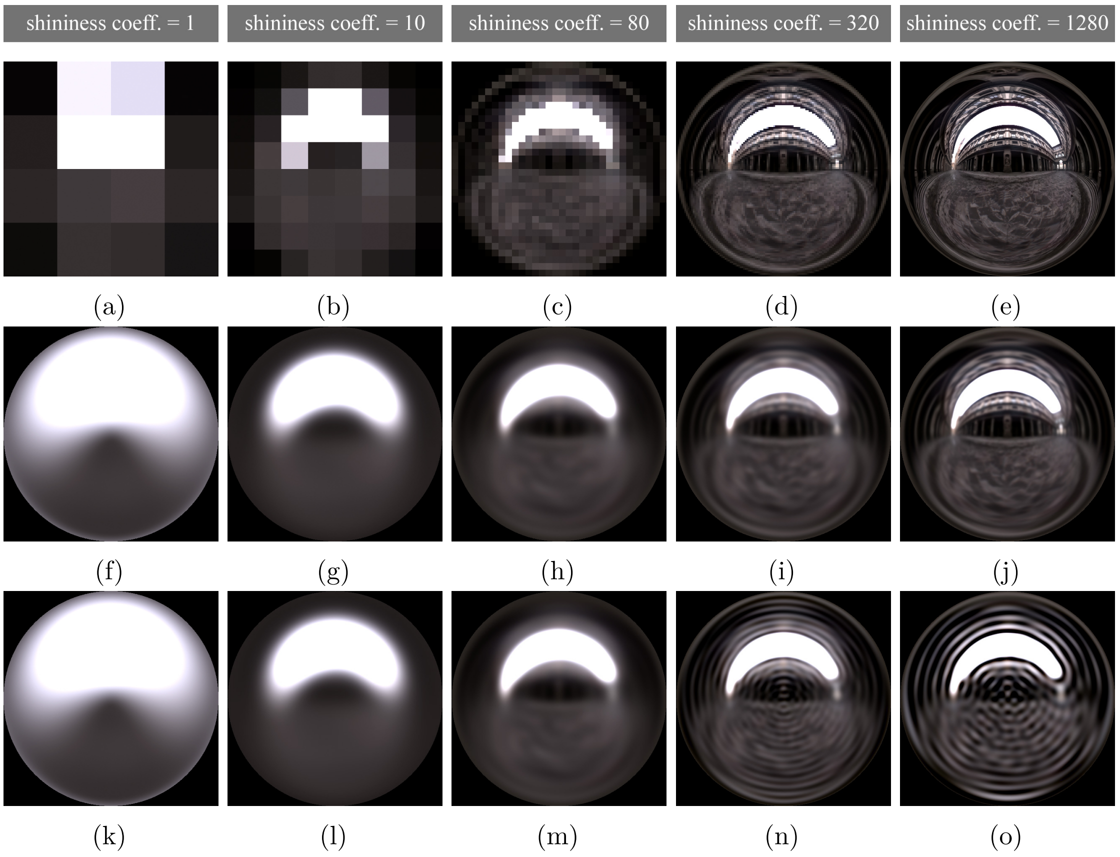


Figure 6.5: Results generated with our three different methods: MIP-mapping (top row), accurate (middle row), and SH (bottom row) for the “Uffizi” light probe. The base map is provided by Debevec [Deb01]. The irradiance maps are shown in the first column: (a), (f), (k); the reflection maps for specular shininess coefficient 10 are shown in the second column: (b), (g), (l); the reflection maps for specular shininess coefficient 80 are shown in the third column: (c), (h), (m); the reflection maps for specular shininess coefficient 320 are shown in the fourth column: (d), (i), (n); and the reflection maps for specular shininess coefficient 1280 are shown in the fifth column: (e), (j), (o).

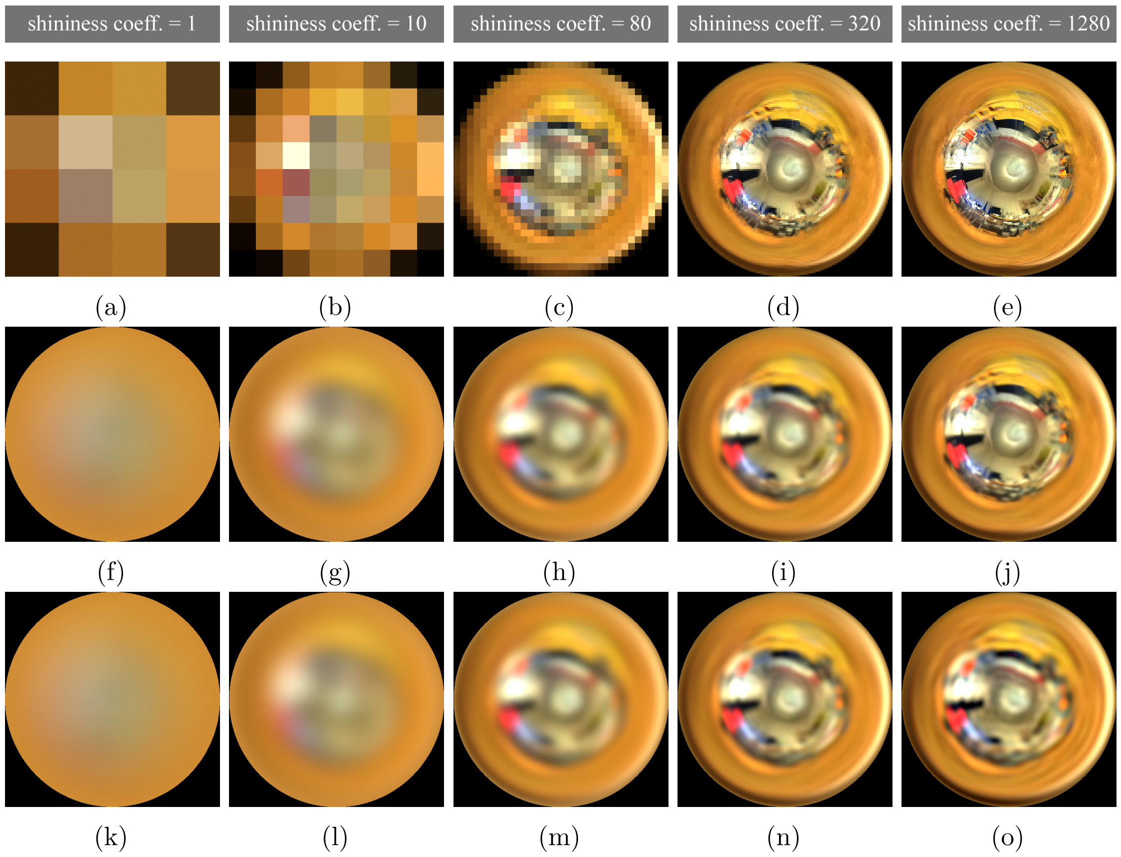


Figure 6.6: Results generated with our three different methods: MIP-mapping (top row), accurate (middle row), and SH (bottom row) for the “Room” light probe. The base map has been captured with our application on a mobile device. The irradiance maps are shown in the first column: (a), (f), (k); the reflection maps for specular shininess coefficient 10 are shown in the second column: (b), (g), (l); the reflection maps for specular shininess coefficient 80 are shown in the third column: (c), (h), (m); the reflection maps for specular shininess coefficient 320 are shown in the fourth column: (d), (i), (n); and the reflection maps for specular shininess coefficient 1280 are shown in the fifth column: (e), (j), (o).

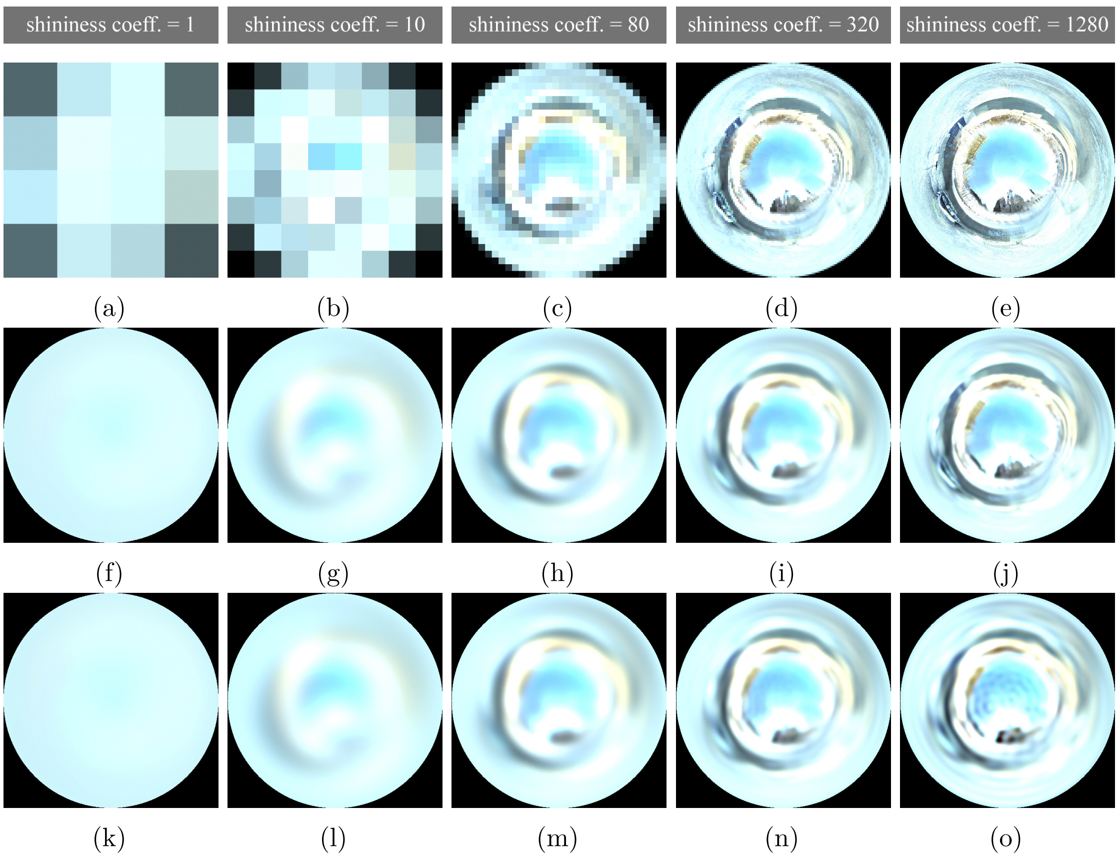


Figure 6.7: Results generated with our three different methods: MIP-mapping (top row), accurate (middle row), and SH (bottom row) for the "Room" light probe. The base map has been captured with our application on a mobile device. The irradiance maps are shown in the first column: (a), (f), (k); the reflection maps for specular shininess coefficient 10 are shown in the second column: (b), (g), (l); the reflection maps for specular shininess coefficient 80 are shown in the third column: (c), (h), (m); the reflection maps for specular shininess coefficient 320 are shown in the fourth column: (d), (i), (n); and the reflection maps for specular shininess coefficient 1280 are shown in the fifth column: (e), (j), (o).

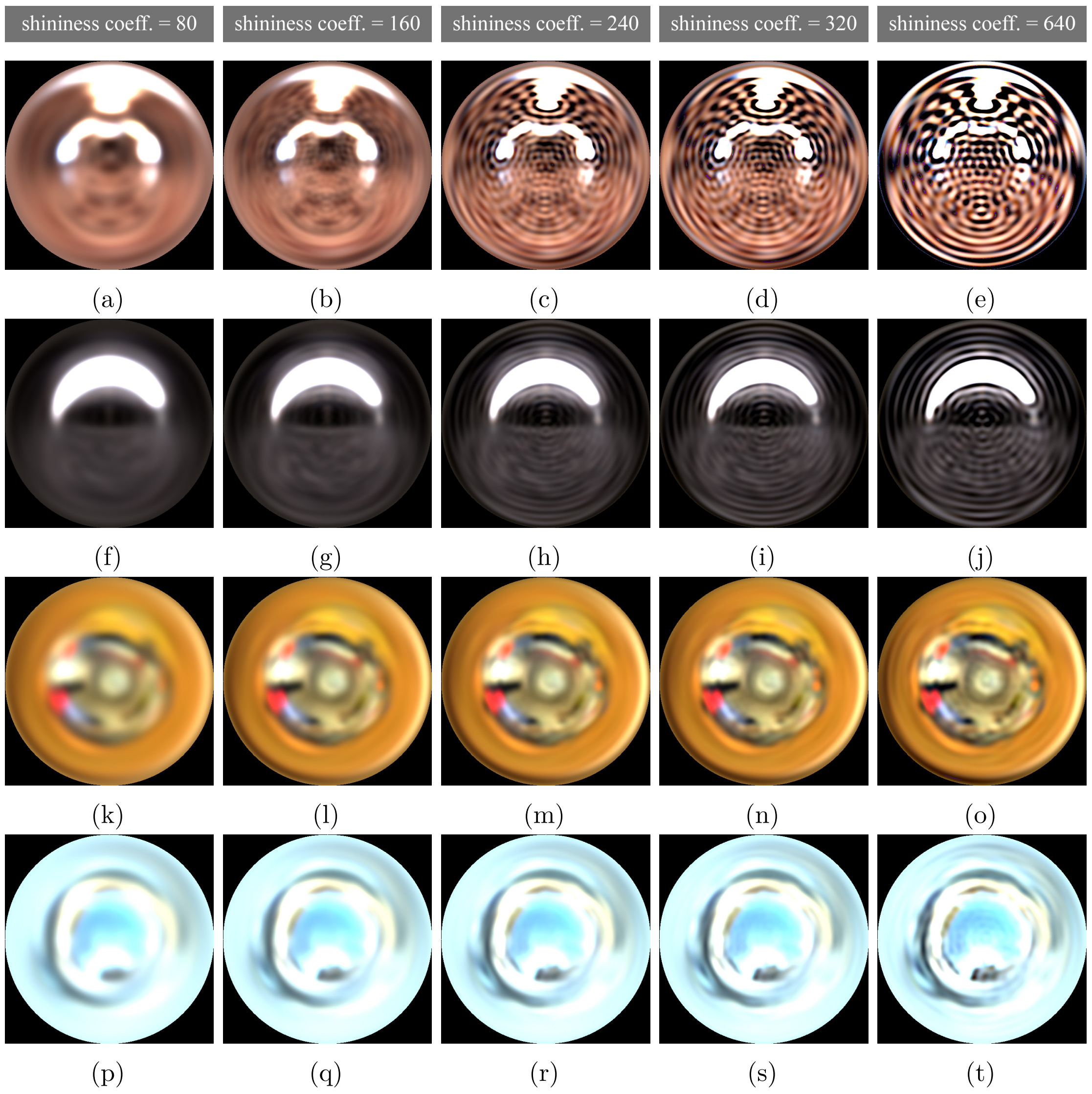


Figure 6.8: SH results for the specular shininess coefficients 80, 160, 240, 320, and 640 for all four maps: the "St. Peters" light probe (top row), "Uffizi" light probe (second row), "Room" light probe (third row), and "Karlsplatz" light probe (bottom row). The reflection maps for specular shininess coefficient 80 are shown in the first column: (a), (f), (k), (p); the reflection maps for specular shininess coefficient 160 are shown in the second column: (b), (g), (l), (q); the reflection maps for specular shininess coefficient 240 are shown in the third column: (c), (h), (m), (r); the reflection maps for specular shininess coefficient 320 are shown in the fourth column: (d), (i), (n), (s); and the reflection maps for specular shininess coefficient 640 are shown in the fifth column: (e), (j), (o), (t). The base maps for "St. Peters" and "Uffizi" are provided by Debevec [Deb01], the base maps for "Room" and "Karlsplatz" have been captured with our application on a mobile device.

Basis functions on higher SH-orders are able to capture higher frequency image details. Irradiance changes very slowly, which can be captured by the low order SH basis functions as pointed out by Ramamoorthi and Hanrahan [RH01a]. An explanation for this is sketched by Akenine-Möller et al.: The SH coefficients of the clamped cosine function, which are used with SH convolution to calculate the irradiance map, have very small values beyond the first nine coefficients [AMHH08]. The reflection maps can contain high frequency image details with increasing specular shininess coefficients which means that the SH coefficients for the corresponding clamped cosine-power function have relevant values beyond SH band level 2. Equation 2.16 describes the reconstruction of a target function, i.e. the reflection map in this case. In the cases of the results shown in figures 6.4, 6.5, 6.6, and 6.7 we are stopping the reconstruction after $n = 1225$ basis functions, corresponding to SH-order 35. Interestingly, while the "St. Peters" and "Uffizi" reflection maps can not be reconstructed without noticeable ringing artifacts for specular shinines coefficients of 240 or higher, the light probes captured with our app show very little ringing artifacts in their reconstructed reflection maps. It can be argued that our captured maps do not change as fast as Debevec's light probes over neighboring pixels and thus contain fewer high frequency image details. Debevec's light probes appear to be containing finer details whereas the captured light probes are blurrier, which are explanations for the higher amount of artifacts in the corresponding reflection maps which were generated from Debevec's light probes with the SH method.

Analyzing the reflection maps generated with the SH method in figure 6.8, it seems like ringing artifacts start to emerge between specular shininess coefficients 160 and 240 with the used SH-order of 35. While the artifacts can already be glimpsed at specular shininess coefficient 160 with the "St. Peters" reflection maps, they appear to emerge a little later with the "Uffizi" map at specular coefficient 240 and are barely visible with the "Room" and "Karlsplatz" reflection maps even up to specular shininess coefficient 640.

While the figures show the visual results, tables 6.1 and 6.2 contain the error rates for the MIP-mapping and the SH method using SH-order 35 in comparison to the accurate solution. The error rates showed in all tables are mean and maximum absolute percentage errors. The mean percentage errors are calculated via equation 6.1,

$$E_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{a_i - v_i}{a_i + \epsilon} \right| \tag{6.1}$$

where n is the number of all pixels and color channel values in a light probe, a_i is the light intensity value of the accurate result, and v_i is the light intensity value of the map which is to be compared with the corresponding accurate value. The maximum percentage errors are calculated via equation 6.2. An ϵ value is added to avoid divisions by zero.

$$E_{\max} = \max \left| \frac{a_i - v_i}{a_i + \epsilon} \right| \forall j \leq n \tag{6.2}$$

shininess coeff.	**stpeters**		**uffizi**		**room**		**karlsplatz**	
	MM	**SH**	**MM**	**SH**	**MM**	**SH**	**MM**	**SH**
1	55.17%	1.56%	55.98%	1.18%	23.22%	0.00%	16.83%	0.00%
2	52.05%	1.56%	60.74%	1.18%	22.42%	0.00%	16.65%	0.00%
10	48.48%	1.61%	139.46%	1.19%	21.61%	0.00%	18.09%	0.00%
20	38.60%	1.64%	45.69%	1.20%	19.28%	0.00%	13.49%	0.00%
40	34.17%	1.67%	25.66%	1.21%	16.62%	0.01%	11.49%	0.00%
80	29.58%	1.69%	24.62%	1.21%	14.57%	0.01%	9.71%	0.00%
160	28.62%	4.17%	28.51%	2.04%	12.67%	0.09%	8.86%	0.07%
240	24.88%	23.96%	25.56%	13.54%	11.09%	0.73%	7.91%	0.60%
320	26.10%	39.12%	29.36%	22.86%	10.89%	1.33%	8.50%	1.07%
640	22.51%	96.79%	29.06%	55.49%	9.42%	5.04%	8.60%	3.44%
1280	18.24%	142.09%	27.54%	104.34%	8.69%	7.82%	9.89%	6.05%
2560	13.48%	165.53%	23.10%	206.65%	7.19%	12.55%	8.88%	8.40%
5120	9.51%	188.30%	19.19%	2585.76%	5.95%	14.31%	7.97%	10.53%
10240	6.41%	211.39%	15.74%	162837.52%	4.85%	18.22%	7.06%	12.93%
20480	4.10%	225.20%	12.27%	171329.41%	3.77%	20.75%	6.00%	14.32%

Table 6.1: Mean absolute percentage error of the irradiance and reflection maps resulting from the MIP-mapping based method in relation to the accurate results in the “MM” labeled columns, and the maps resulting from the SH-based method in relation to the accurate results in the “SH” labeled columns for four different light probes. Each row represents all maps calculated for the respective specular coefficient, where the first row represents the irradiance maps and all other rows represent reflection maps. SH-order 35 has been used to generate the SH results.

shininess coeff.	**stpeters**		**uffizi**		**room**		**karlsplatz**	
	MM	**SH**	**MM**	**SH**	**MM**	**SH**	**MM**	**SH**
1	375.36%	3.16%	709.14%	2.04%	91.94%	2.63%	74.82%	1.09%
2	358.74%	3.06%	1344.90%	1.91%	98.33%	2.78%	74.27%	1.18%
10	517.39%	3.57%	2961.62%	1.92%	158.70%	3.23%	89.19%	2.33%
20	724.39%	3.80%	2033.33%	2.03%	215.62%	3.85%	109.68%	2.63%
40	1432.71%	3.75%	653.93%	2.11%	151.61%	4.55%	102.87%	1.89%
80	867.11%	3.90%	253.33%	2.24%	173.91%	4.76%	105.41%	2.44%
160	956.00%	50.71%	202.27%	34.37%	143.01%	5.26%	128.31%	2.86%
240	896.23%	283.67%	204.97%	441.93%	133.80%	21.51%	124.00%	10.85%
320	1545.71%	506.99%	197.30%	987.62%	190.07%	47.33%	168.47%	20.16%
640	1766.67%	1698.16%	269.01%	4983.56%	515.38%	454.35%	199.07%	82.85%
1280	1738.14%	4266.35%	465.13%	29396.84%	798.55%	227.27%	528.07%	165.75%

Table 6.2: Maximum absolute percentage error of the irradiance and reflection maps resulting from the MIP-mapping based method in relation to the accurate results in the “MM” labeled columns, and the maps resulting from the SH-based method in relation to the accurate results in the “SH” labeled columns for four different light probes. Each row represents all maps calculated for the respective specular coefficient, where the first row represents the irradiance maps and all other rows represent reflection maps. SH-order 35 has been used to generate the SH results.

The error rates confirm our observations of the visual results. The mean error rates with MIP-mapping are decreasing with increasing specular shininess coefficient, while the maximum error rates are constantly very high for all maps and all specular shininess coefficients. Generally, the mean error rates are lower for the light probes captured on the device compared to Debevec's light probes.

The SH error rates generally stay very low as long as the SH basis functions of the first 34 frequency bands (SH-order 35) are able to capture the image details of a reflection map for a specific specular coefficient. The threshold seems to depend on the map. While the maximum error of "St. Peters" is already at 50.71% with a specular shininess coefficient of 160, "Uffizi" is at 34.37% for the same coefficient, and the captured maps – "Room" and "Karlsplatz" – are much lower with 5.26% and 2.86%, respectively. Their mean error rates stay below 8% up until specular shininess coefficients of 1280 while the mean error rates of "St. Peters" and "Uffizi" are rising to high levels starting at specular shininess coefficient 240. This corresponds to our observations of the visual results, where ringing artifacts start to emerge at this specular shininess coefficient. Depending on the input light probe's details and the SH reconstruction quality, above a certain threshold the SH error rates measure mostly the results of the ringing artifact.

With regard to comparing the SH-based method with the accurate method in section 6.2.2, we have to estimate which specular shininess coefficients can be represented by a specific SH-order. The mean and maximum error rates from tables 6.1 and 6.2 suggest the *representable specular shininess coefficient* for SH-order 35 to lie between specular shininess coefficients 80 and 160. The mean absolute percentage errors are very small for all maps. The maximum absolute percentage errors start to rise between those two specular shininess coefficients for some light probes. In section 6.2.1 we analyze SH results in more detail and elaborate on how different SH-orders affect the results.

6.2 Optimizations and Performance Measurements

We have analyzed our method in terms of quality, but especially on a mobile device, also the calculation time of an algorithm is an important factor. In this section, we elaborate on how our methods can be optimized with respect to performance and quality. We analyze calculation times for all methods except for the MIP-mapping based method since MIP-mapping is usually implemented in hardware and calculates practically instantly on the GPU. Calculation times for the SH and the accurate methods can be optimized – for SH, the number of used basis functions can be varied; for the accurate method, the texture lookup bounds can be narrowed depending on the specular shininess coefficient. All *maps calculation times* we specify in this section have been measured on a Nvidia Shield tablet, calculating eight maps in parallel on the GPU.

6.2.1 Spherical Harmonics Order

Calculating maps accurately takes 01:07.080 minutes for the resolution of 512×512. Calculating the maps via SH frequency space requires two steps. Both of them have to be performed after capturing an environment map. The first step computes an environment map's SH coefficients and takes 36.597 seconds for SH-order 35. The second step computes the actual irradiance and environment maps from the SH coefficients and takes 5.737 seconds for SH-order 35. If pre-calculated SH function values are utilized, the computation time for the first step can be reduced to 8.458 seconds.

The calculation times with different SH-orders are shown in table 6.3. The calculation times for the first step are shown in the column labeled "SH coefficients calculation time". The calculation times for the second step are shown in the column labeled "maps calc. time from SH coeff.". The column with the abbreviated labeling for "SH coefficients calculation with pre-calculated SH values" shows the calculation times for the equivalent calculations to column "SH coefficients calculation time" but using pre-calculated SH function values, and column "pre-calculated SH value size" shows the storage requirements for pre-calculated SH function values.

Pre-calculating SH function values (values of the *SH* function from algorithm 2.6) is a possibility to significantly speed up computation of a light probe's SH coefficients for a specific texture size. The function value depends on the indices for a specific SH basis function l and m, and on the azimuth and elevation angles θ, and ϕ. Those two angles depend on the number of pixels in a texture, which is the reason why a specific texture size has to be assumed. The *pre-calculated SH values sizes* in table 6.3 refer to a texture size of 512×512. For each pixel, the *SH* function's values have to be computed for every basis function which is the reason for the high storage requirements. For instance, the uncompressed storage requirements for SH-order 35 are calculated by multiplying the texture size of 512×512 with the number of basis functions, which is 1225, resulting in a total of 3.21×10^8 bytes. The high storage requirements pay off by significantly lower computation times for SH coefficients calculation which are stated in the sixth column of the table. These computation times have been created by a linear summation (see implementation of function *sumForCoeffs* in algorithm 4.4). By implementing a logarithmic reduce approach, further performance increase can be expected.

SH-order	SH coefficients calculation time	maps calc. time from SH coeff.	sum of SH coeff. calc. and maps calc.	pre-calculated SH values size	SH coeff. calc. w/ pre-calc. SH val.
86	06:01.132	01:03.270 *	07:04.402	1.94×10^9 bytes	00:47.681
75	04:13.349	00:45.252 *	04:58.601	1.47×10^9 bytes	00:38.145
63	02:38.833	00:26.213	03:05.046	1.04×10^9 bytes	00:27.529
50	01:27.075	00:14.223	01:41.297	6.55×10^8 bytes	00:18.834
35	00:36.597	00:05.737	00:42.334	3.21×10^8 bytes	00:08.458
25	00:16.301	00:02.754	00:19.055	1.64×10^8 bytes	00:04.766
15	00:04.422	00:00.927	00:05.349	5.9×10^7 bytes	00:01.461
10	00:01.251	00:00.358	00:01.609	2.62×10^7 bytes	00:00.690
7	00:00.561	00:00.132	00:00.693	1.28×10^7 bytes	00:00.429
5	00:00.265	00:00.081	00:00.346	6.55×10^6 bytes	00:00.212
3	00:00.124	00:00.013	00:00.137	2.36×10^6 bytes	00:00.081

Table 6.3: This table shows computation times and sizes for computing irradiance and reflection maps with different SH-orders. The second column states the SH coefficients calulation times of a light probe via algorithm 6.3. The times for calculating actual irradiance and reflection maps from the resulting SH coefficients are stated in the third column. The fourth column is the sum of both, the SH coefficients calculation time and the maps calculation time. The calculation times were generated with input envrionment maps sized 512 × 512. During the maps calculation step (third column), eight irradiance and reflection maps are calculated in parallel. Columns five and six show relevant data for a performance optimization where the values of the *SH* function from algorithm 2.6 are pre-calculated. The required uncompressed data storage sizes are listed in column five. The computation times for the SH coefficients calculation step with pre-calculated SH function values are listed in column six. Compared with the computation times in column one, they indicate big performance improvements. All calculation times are specified in minutes, seconds and milliseconds.

For the results presented in section 6.1 we used maps generated with SH-order 35 although we calculated even higher SH-orders. The reason for not using higher SH-orders can be seen in tables 6.4 and 6.5. Error rates of reconstructed maps do not change for SH-orders higher than 35 compared to the error rates of SH-order 35. Likewise, the visual results do not change. We attribute this to precision problems which occur even with double precision floating-point arithmetic.

We strived to perform all computations on the device in double precision since double precision is essential for the correctness of the SH results. SH frequency domain transformations as we have presented them in section 2.4 require high factorial values in their intrinsic calculations. The hard limit for SH calculations in double precision is SH-order 86 which requires factorials up to 170!, which is the highest factorial representable in double precision. Already 171! can not be represented in double precision anymore. Assuming a decimal precision of approximately 16 decimal digits for a double precision data type, already factorials higher than 18! induce inaccuracies in the computations.

Factorials higher than 18! are required for SH-orders higher than 10. In addition to very high values, very low values occur as well during SH calculations causing additional inaccuracies on the other end of the floating point range. As an example, contributions to specific SH-coefficients are summed up in the *sumForCoeffs* function of algorithm 4.4. They can become very small – especially for the SH basis functions on higher frequency bands. Adding a value smaller than the smallest precisely representable floating point number to a double precision variable might not change the result. These inaccuracies contribute to increasing error rates for higher SH-orders as can be examined in table 6.4 for the "St. Peters" light probe and in table 6.5 for the "Karlsplatz" light probe. Interestingly, the "Karlsplatz" light probe shows relatively small error rates up to specular shininess coefficient 20480 compared to the "St. Peters" light probe. The "Karlsplatz" light probe does not change very abruptly between pixels and is blurrier than the "St. Peters" light probe. The SH basis functions on the lower frequency bands are obviously able to capture the light probe's data very well and basis functions on higher frequency bands seem to be less important for representing it. The "St. Peters" light probe, on the other hand, shows higher error rates. For a better SH frequency space representation of it, it would require the SH basis functions on higher frequency bands to better capture its image details.

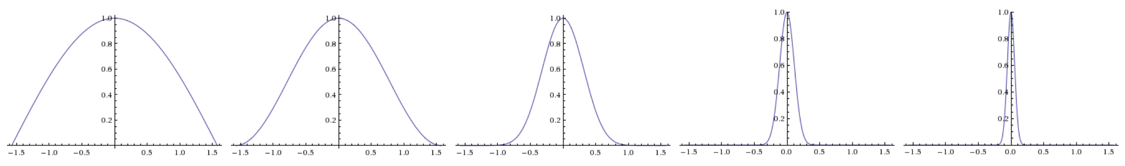
Despite the floating point precision issues, results for small specular shininess coefficients can be very close to the accurate versions and they can be computed faster than the accurate solutions with SH-order 35. In the next section we propose an approach to increase performance of the accurate calculation and compare the results with the SH results presented in this section.

shininess	**St. Peters**							
coefficient	**50**	**35**	**25**	**15**	**10**	**7**	**5**	**3**
1	0.55%	0.55%	0.56%	0.57%	0.65%	0.83%	1.20%	2.59%
2	0.65%	0.65%	0.65%	0.65%	0.66%	0.69%	1.15%	6.09%
10	0.92%	0.92%	0.92%	0.92%	0.92%	4.64%	17.64%	39.44%
20	0.99%	0.99%	0.99%	0.99%	5.58%	19.51%	37.45%	64.38%
40	1.03%	1.03%	1.03%	5.24%	23.88%	42.03%	62.47%	91.98%
80	1.08%	1.08%	2.85%	25.38%	48.54%	72.54%	92.43%	119.21%
160	4.29%	4.29%	22.18%	52.46%	77.93%	107.18%	122.51%	143.16%
240	25.41%	25.41%	41.79%	69.80%	97.33%	127.19%	138.78%	155.15%
320	41.14%	41.14%	56.49%	83.16%	111.76%	140.90%	149.61%	162.93%
640	92.51%	92.51%	92.80%	118.98%	147.34%	171.97%	173.49%	179.87%
1280	133.42%	133.42%	133.42%	157.02%	181.09%	199.55%	194.23%	194.60%
2560	171.17%	171.17%	171.17%	190.07%	208.32%	221.19%	210.64%	206.56%
5120	200.60%	200.60%	200.60%	213.56%	227.07%	236.21%	222.39%	215.63%
10240	229.76%	229.76%	229.76%	228.00%	238.62%	245.80%	230.06%	222.13%
20480	241.79%	241.79%	241.79%	236.60%	245.56%	251.84%	234.93%	226.71%

Table 6.4: Mean absolute percentage error rates for calculated irradiance and reflection maps of Debevec's "St. Peters" light probe with varying number of SH-orders compared to the accurate solution. The first column indicates which specular shininess coefficients have been used to generate the results. Each row shows the resulting error rates of one specific specular shininess coefficient for a variety of different SH-orders.

shininess	**Karlsplatz**							
coefficient	**50**	**35**	**25**	**15**	**10**	**7**	**5**	**3**
1	0.00%	0.00%	0.00%	0.01%	0.05%	0.09%	0.17%	0.58%
2	0.00%	0.00%	0.00%	0.00%	0.01%	0.07%	0.21%	1.92%
10	0.00%	0.00%	0.00%	0.00%	0.02%	0.98%	3.43%	9.00%
20	0.00%	0.00%	0.00%	0.01%	0.63%	3.76%	6.97%	12.52%
40	0.00%	0.00%	0.00%	0.33%	2.65%	7.73%	10.76%	16.02%
80	0.00%	0.00%	0.08%	1.63%	5.67%	11.77%	14.37%	19.35%
160	0.07%	0.07%	0.66%	3.88%	8.79%	15.20%	17.53%	22.32%
240	0.60%	0.60%	1.41%	5.34%	10.42%	16.86%	19.12%	23.83%
320	1.07%	1.07%	2.10%	6.37%	11.47%	17.89%	20.13%	24.79%
640	3.44%	3.44%	4.06%	8.64%	13.67%	19.99%	22.21%	26.80%
1280	6.05%	6.05%	6.05%	10.60%	15.48%	21.69%	23.95%	28.48%
2560	8.40%	8.40%	8.40%	12.23%	16.98%	23.10%	25.42%	29.90%
5120	10.53%	10.53%	10.53%	13.56%	18.20%	24.26%	26.64%	31.09%
10240	12.93%	12.93%	12.93%	14.65%	19.20%	25.20%	27.65%	32.08%
20480	14.32%	14.32%	14.32%	15.65%	20.11%	26.03%	28.58%	32.99%

Table 6.5: Mean absolute percentage error rates for calculated irradiance and reflection maps of a captured light probe "Karlsplatz" with varying number of SH-orders compared to the accurate solution. The first column indicates which specular shininess coefficients have been used to generate the results. Each row shows the resulting error rates of one specific specular shininess coefficient for a variety of different SH-orders.



(a) (b) (c) (d) (e)

Figure 6.9: Cosine lobes for the functions cos (a), $\cos^2$ (b), $\cos^{10}$ (c), $\cos^{80}$ (d), and $\cos^{320}$ (e). Images are reprinted from[18].

6.2.2 Bounds Optimization for Accurate Computation

In this section, we sketch an approach to speed up the accurate method significantly, depending on the specular shininess coefficient. The accurate algorithm from section 4.1 iterates over all directions to add their respective illumination contributions. However, considering that higher cosine powers tend to zero very quickly as can be seen in figure 6.9, the idea is to not iterate over all directions but instead over a narrower region. To exemplarily attest the benefits of this optimization, we selected specific specular shininess coefficients and manually defined the narrow bounds area for each of them like described in figure 6.10.

Following this practice, we manually defined optimized bounds for five selected specular shininess coefficients and used them to measure the performance gains. At the end of this section, we compare the run-times of the accurate method with bounds optimization to the run-times of varying SH-order, which allows us to make recommendations regarding usage scenarios for these two methods.

The bounds in uv-space which we have defined manually are like follows: 0.36 for specular shininess coefficient 40, 0.24 for specular shininess coefficient 80, 0.18 for specular shininess coefficient 160, 0.14 for specular shininess coefficient 320, and 0.0666 for specular shininess coefficient 1280. For the calculation of an irradiance map, the bounds cannot be narrowed since the light directions are weighted by the cosine which requires all directions to be taken into account, as can be seen in figure (6.9a). This means that for the specular shininess coefficient 1, the bounds have to encompass the full texture and be set to 1.0 which corresponds exactly to the accurate method without bounds optimization.

The implementation of the bounds optimization for specular shininess coefficient 80 is shown in algorithm 6.1 and it can be applied by modifying algorithm 4.1 to call *getNarrowRowBounds80* instead of *getRowBounds*. *getNarrowRowBounds80* regards all incoming light directions within the region marked with a red square in images (6.10a), (6.10b), (6.10c), (6.10d), and (6.10e). Our goal was to regard all directions with a cosine contribution greater than 0.03. The bounds optimization achieves that for many cases, but not for all. If a reflection direction gets mapped to the sphere map's border where the sphere map has its singularity, the relevant directions are spread across the entire texture.

[18]Wolfram Alpha LLC, WolframAlpha computational knowledge engine
`http://www.wolframalpha.com`

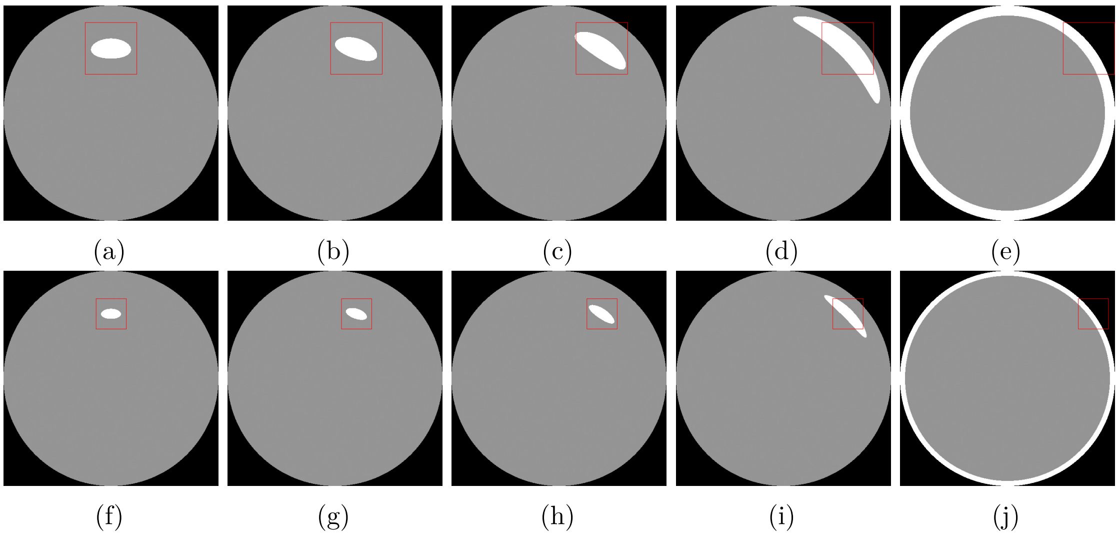


Figure 6.10: These images illustrate the region which we consider for the bounds optimization with specular shininess coefficient 80 in the top row, and for specular shininess coefficient 320 in the bottom row. The region which our shader implementation considers is marked with a red square in each image and has been defined manually. The white areas represent regions where the corresponding cosine-power has values higher than 0.03. The bounds were chosen to include the white area completely in images (a), (b), (c), (f), (g), (h) and partially in images (d), (e), (i), (j). The partial fit is caused by increasing size of cosine projection towards the borders, where a sphere map has its singularity.

In order to create a fast GPU shader implementation, we decided not to introduce extra handling of such cases and just sample inside the red square region regardless of the direction. Interestingly, the results are very good in terms of quality and show small error rates.

We analyze the quality of the results generated with the bounds optimization for the specular shininess coefficients 80, 320, and 1280 for the light probes “Karlsplatz” and “St. Peters” for a variety of specular shininess coefficients, i.e. we use a specific bounds optimization to calculate the results for a wide range of specular shininess coefficients including those which would actually require wider bounds in the strict sense of this optimization as we have described it above. The expectation is that for a specific bounds optimization, the error rates should be low for all specular shininess coefficients greater or equal to the specific bounds optimization’s target specular shininess coefficient.

Tables 6.7, and 6.8, show the mean and maximum error rates for a specific bounds optimization. Actually, in the sense of the bounds optimization, the error rates are relevant only starting from the specular shininess coefficient which the optimization has been targeted at. For instance, relevant error rates for the *getNarrowRowBounds80* would be specular shininess coefficient 80 and all which are greater. For all relevant

Algorithm 6.1: getNarrowRowBounds80 for specular power 80

```
1 function getNarrowRowBounds80(s, t, oneTxl, halfTxl)
      input  : $s$, $t$, $oneTxl$, $halfTxl$
      output : $b_n[2]$
2     $b = \text{getRowBounds}(t, oneTxl, halfTxl)$;
3     $b_n[0] = \max(s - 0.12, b[0])$;
4     $b_n[1] = b_n[0] + 0.24$;
5     if $b_n[1] > 1.0$ then
6         $b_n[1] = \min(s + 0.12, b[1])$;
7         $b_n[0] = b_n[1] - 0.24$;
8     end
9     return $b_n$;
```

specular shininess coefficients, the respective bounds optimizations yield results with mean absolute percentage errors of 0.46% or less for the "Karlsplatz" light probe, and 5.40% or less for the "St. Peters" light probe. The relevant maximum absolute percentage errors are 7.07% or less for the "Karlsplatz" light probe, and go up to 181.18% for the "St. Peters" light probe.

The performance gains using bounds optimization are significant as shown in table 6.6. These optimizations allowed us to calculate the reflection maps in one tile which is infeasible for the unoptimized version that causes the device to crash due to the high number of computations in a single frame. Figure 6.11 finally shows how artifacts could look like if the bounds are chosen too narrow with sphere mapping.

Optimization	**Tiled**	**Calculation time**
none	yes, 16 tiles	01:07.860
bounds for spec. coefficients 80 and above	yes, 16 tiles	00:16.818
bounds for spec. coefficients 320 and above	yes, 16 tiles	00:10.600
bounds for spec. coefficients 1280 and above	yes, 16 tiles	00:05.345
none	no	*infeasible*
bounds for spec. coefficients 80 and above	no	00:15.795
bounds for spec. coefficients 320 and above	no	00:09.488
bounds for spec. coefficients 1280 and above	no	00:04.906

Table 6.6: Calculation times of the accurate method's bounds optimization compared to the times of the unoptimized accurate method with different configurations (tiles, and bounds range).

coeff	Karlsplatz			St. Peters		
	80	**320**	**1280**	**80**	**320**	**1280**
1	4.55%	7.57%	9.88%	34.61%	44.88%	53.09%
2	3.73%	6.87%	9.25%	32.28%	43.18%	51.02%
10	1.78%	3.83%	6.51%	21.03%	30.77%	38.54%
20	1.13%	2.37%	4.93%	15.15%	22.95%	30.28%
40	0.72%	1.33%	3.37%	9.72%	15.50%	21.86%
80	0.46%	0.74%	2.01%	5.40%	9.40%	14.32%
160	0.28%	0.42%	1.05%	2.85%	5.44%	8.91%
240	0.20%	0.30%	0.71%	1.99%	3.99%	6.82%
320	0.16%	0.24%	0.54%	1.56%	3.19%	5.67%
640	0.11%	0.16%	0.32%	0.89%	1.78%	3.52%
1280	0.11%	0.13%	0.22%	0.57%	0.97%	2.04%
2560	0.13%	0.12%	0.17%	0.45%	0.58%	1.10%
5120	0.19%	0.15%	0.14%	0.48%	0.45%	0.61%
10240	0.32%	0.22%	0.14%	0.56%	0.44%	0.39%
20480	0.64%	0.38%	0.17%	0.73%	0.50%	0.30%

Table 6.7: Mean absolute percentage error rates for irradiance and reflection maps calculated with the bounds optimization for specular shininess coefficients 80, 320, and 1280 in relation to the unoptimized accurate results. The error rates were calculated using the three bounds optimizations with the light probes "Karlsplatz" and "St. Peters". Each row represents the results for a specular shininess coefficient calculated with the specific column's bounds optimization. The first row represents the irradiance maps and all other rows represent reflection maps.

coeff	Karlsplatz			St. Peters		
	80	**320**	**1280**	**80**	**320**	**1280**
1	19.44%	28.71%	41.41%	209.33%	389.36%	759.57%
2	15.33%	26.67%	43.56%	230.86%	414.85%	775.47%
10	10.53%	17.16%	43.68%	282.35%	425.93%	625.66%
20	9.31%	11.30%	38.18%	292.66%	419.66%	642.86%
40	8.30%	9.26%	27.82%	264.81%	366.67%	555.46%
80	7.07%	8.39%	19.76%	181.18%	245.61%	354.29%
160	5.32%	6.15%	9.37%	82.72%	117.61%	185.92%
240	4.95%	5.32%	6.91%	61.29%	98.11%	164.52%
320	4.90%	5.32%	6.50%	53.19%	90.65%	154.78%
640	4.26%	4.92%	6.50%	44.83%	61.97%	121.62%
1280	3.23%	4.48%	6.98%	37.41%	45.33%	89.40%
2560	4.51%	5.04%	7.75%	25.47%	32.26%	56.86%
5120	9.39%	5.45%	7.05%	30.89%	25.32%	38.00%
10240	23.50%	15.49%	7.19%	57.62%	54.76%	31.48%
20480	77.78%	55.56%	14.41%	101.98%	100.00%	33.53%

Table 6.8: Maximum absolute percentage error rates for irradiance and reflection maps calculated with the bounds optimization for specular shininess coefficients 80, 320, and 1280 in relation to the unoptimized accurate results. The error rates were calculated using the three bounds optimizations with the light probes "Karlsplatz" and "St. Peters". Each row represents the results for a specular shininess coefficient calculated with the specific column's bounds optimization. The first row represents the irradiance maps and all other rows represent reflection maps.

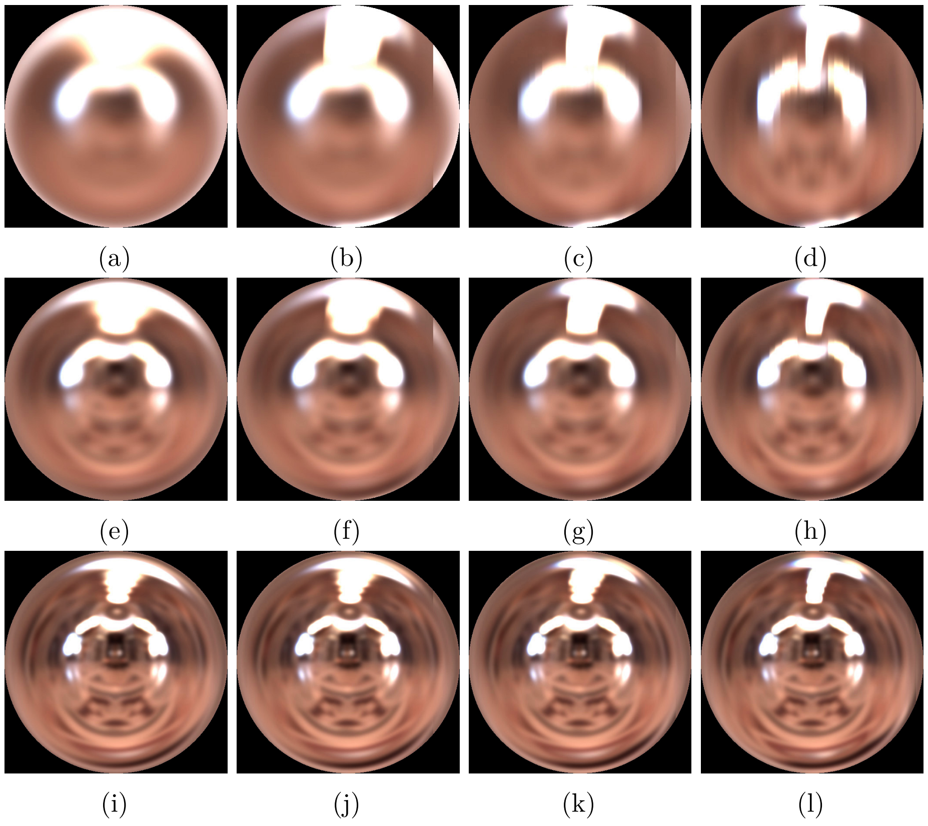


Figure 6.11: Results of accurate calculation with bounds optimization: The top row shows the maps for specular shininess coefficient 20, the middle row shows the maps for specular shininess coefficient 80, and the bottom row shows the maps for specular shininess coefficient 320. Images (a), (e), and (i) have been calculated accurately without optimization; images (b), (f), and (j) have been generated with optimized bounds for specular shininess coefficient 80; images (c), (g), and (k) have been generated with optimized bounds for specular shininess coefficient 320; and images (d), (h), and (l) have been generated with optimized bounds for specular shininess coefficient 1280. In the strict sense of the bounds optimization, maps with a specular shininess coefficient lower than the optimization's target coefficient would not be generated with that specific optimization but instead with a more suitable one. For illustration purposes however, we show which artifacts emerge if that would be done. We have used bounds optimizations for specular shininess coefficients 80, 320, and 1280 for generating these images. This means, in practice we would only calculate the maps from images (f), (j), and (k) with those optimizations, for all the other maps, we would use bounds optimizations for smaller specular shininess coefficients.

The bounds optimization has a lot of potential performance-wise, but the singularity of the sphere map is a flaw which cannot be overcome with a performance-optimized shader implementation like shown in algorithm 6.1. It is the main reason for the high maximum errors in table 6.8. In section 2.2, we have described alternative environment map formats such as cube mapping and dual paraboloid mapping. They would be suited much better for the bounds optimization since they do not have any singularities. In our case, light probes would have to be converted from the sphere map format into cube map format or into dual paraboloid format. Thanks to their properties, even smaller bounds could be chosen for the narrow region around a specific direction. Since dual paraboloid mapping has more uniform texel sampling than cube mapping, it would probably be the best suited environment mapping format for this kind of optimization.

Figures 6.12 and 6.13 show diagrams comparing the computation times of the SH based method with varying SH-order to the accurate method with bounds optimization. It is obvious that the strengths of the two methods lie at the exactly opposite sides of specular shininess range. While the SH method can be used to efficiently calculate irradiance maps and low glossy maps, the accurate method with bounds optimization is much faster with calculating maps of higher specular shininess coefficients. The diagrams show "representable specular shininess coefficient" on the x-axes by which we refer to specular shininess coefficients that have been calculated with mean percentage errors of $< 3\%$.

The diagram in figure 6.12 has been generated based on data for the "St. Peters" light probe by combining the error rates from table 6.4 with the corresponding calculation times from table 6.3 for the SH results and adding the results from accurate calculation with bounds optimization by combining the error rates from table 6.7 with the calculation times from table 6.6.

Likewise, the diagram in figure 6.13 has been generated based on data for the "Karlsplatz" light probe by combining the error rates from table 6.5 with the corresponding calculation times from table 6.3 for the SH results and adding the results from accurate calculation with bounds optimization by combining the error rates for the "Karlsplatz" light probe from table 6.7 with the calculation times from table 6.6.

Both methods are able to calculate the "Karlsplatz" light probe – which was captured with our application on a mobile device – with much lower error rates compared to the "St. Peters" light probe. Since we have created the diagrams based on the error values, the resulting curves are a bit different. Both methods, accurate with bounds optimization and SH-based with varying SH-orders, produce higher error rates for the "St. Peters" light probe. The point of this comparison is to determine the spot where the two different methods balance each other in total computation time. This allows us to tell which of the two methods is better suited for calculating reflection maps of certain specular shininess coefficients per light probe.

For the "St. Peters" results, both methods yield approximately the same error rates and take approximately the same amount of time to compute the results for specular shininess coefficient 92 with total SH calculation time. Considering SH maps calculation time

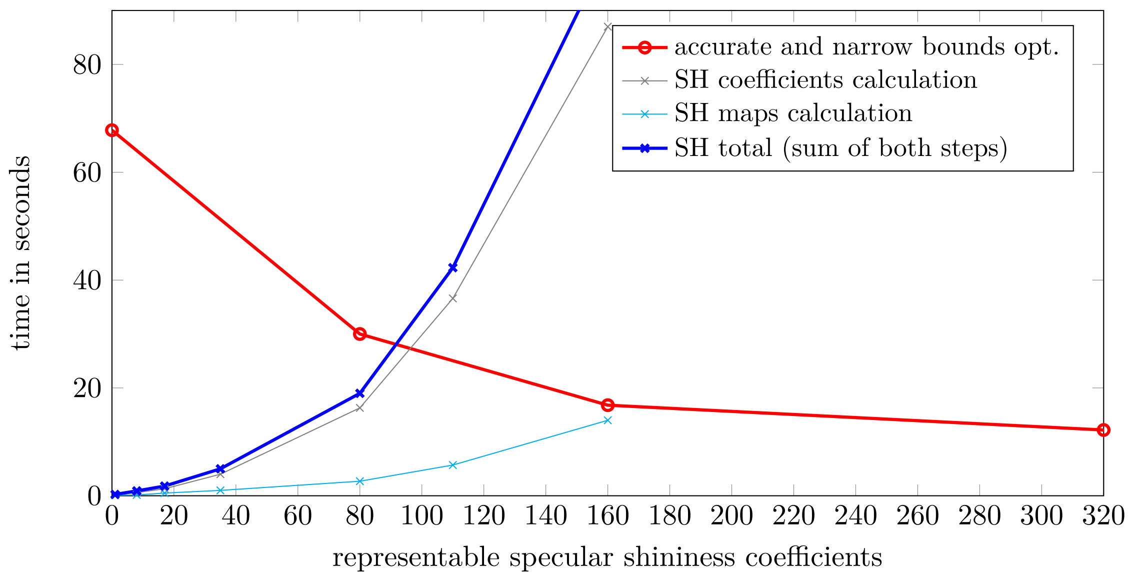


Figure 6.12: This diagram shows the calculation times in seconds per representable specular shininess coefficient for the SH-based method compared to the accurate method with bounds optimization based on the error rates of the "St. Peters" results. The calculation times of the SH-based method are split into the *SH coefficients calculation* step and the *SH maps calculation* step. The two steps are shown as separate curves in addition to the curve of total SH calculation times. The representable specular shininess coefficients on the x-axis have been set to values where the corresponding error rates for the "St. Peters" light probe are $< 3\%$ in table 6.5 for the SH results, and $< 3\%$ in table 6.7 for the accurate computation with bounds optimization results, respectively.

only, the spot shifts towards specular shininess coefficient 160. For all higher specular shininess coefficients, the accurate method with bounds optimization calculates the result more efficiently. A suitable choice to calculate reflection maps with specular shininess coefficients of 92 would be around SH-order 30 in this case.

The "Karlsplatz" diagram shows different curves and a different balance point. The overall curve tendencies, however, show a similar form. Both methods yield approximately the same error rates and take approximately the same amount of time to compute the results for specular shininess coefficient 173 with total SH calculation time. Considering SH maps calculation time only, the spot shifts towards specular shininess coefficient 440. SH perform better in this case because the basis functions are able to better capture the light probe's image details. For the "Karlsplatz" light probe, suitable choices for the specular shininess coefficients 173 and 440 would be SH-orders of around 18 and 32, respectively.

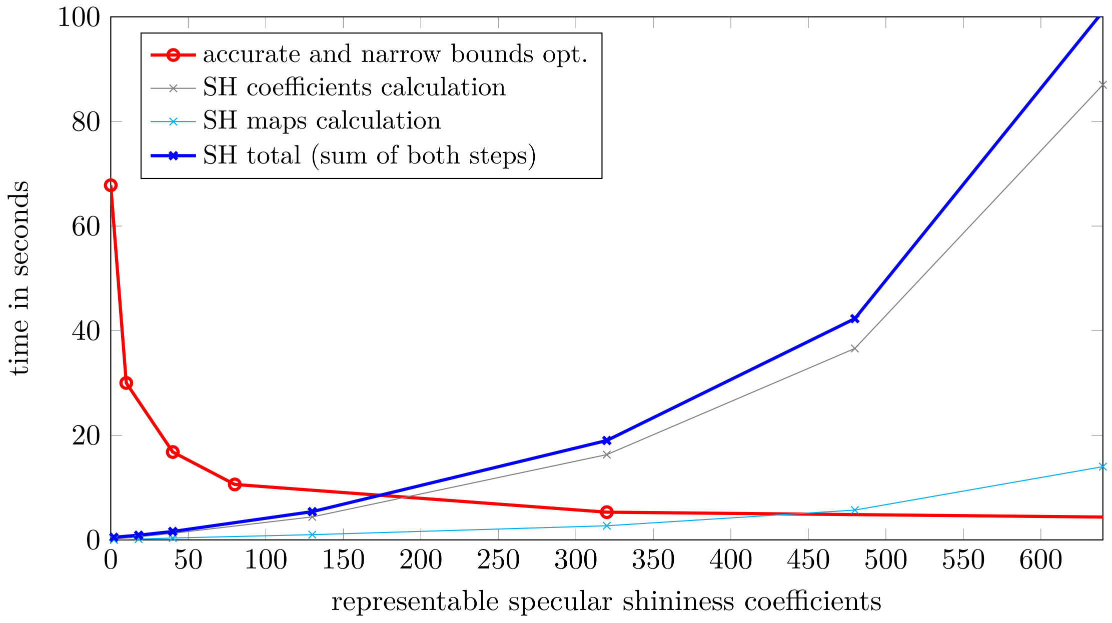


Figure 6.13: This diagram shows the calculation times in seconds per representable specular shininess coefficient for the SH-based method compared to the accurate method with bounds optimization based on the error rates of the "Karlsplatz" results. The calculation times of the SH-based method are split into the *SH coefficients calculation* step and the *SH maps calculation* step. The two steps are shown as separate curves in addition to the curve of total SH calculation times. The representable specular shininess coefficients on the x-axis have been set to values where the corresponding error rates for the "Karlsplatz" light probe are $< 3\%$ in table 6.5 for the SH results, and $< 3\%$ in table 6.7 for the accurate computation with bounds optimization results, respectively.

6.2.3 Extension of MIP-Mapping Based Computation

Since the MIP-mapping based method has high error rates, is not based on the Phong BRDF, and generally computes almost instantly, we did not include it in diagrams 6.12 and 6.13. However, it can support a final optimization strategy. In section 2.3, we have described a technique by Colbert and Křivánek called "GPU-based importance sampling" [CK07]. As we have explained, they are using Monte Carlo quadrature with up to 40 random sample directions, importance sampling, and MIP-mapping based filtering for a real-time solution to generating irradiance and reflection maps. Based on our analyses, we can state that the errors for the MIP-mapping based results are getting lower with higher specular shininess coefficients. Furthermore, we can state that a higher amount of sample directions leads to a lower error rate. Based thereon, we can infer that a method could be developed which can be seen as a middle ground between the accurate method with bounds optimization and the MIP-mapping based method by utilizing Monte Carlo quadrature, importance sampling, and MIP-mapping based filtering as proposed in [CK07]. The number of samples could be varied according to quality or performance demands between the accurate method with bounds optimization – which represents the upper bound in that regard – and the pure MIP-mapping based method – which calculates the results practically instantly and represents the lower bound.

Comparing this approach with the SH-based method in terms of the quality-performance ratio, we think that SH will be unmatched for low glossy maps up to a SH-order of approximately 15 and remain very competitive up to SH-orders of 30. As we have observed, SH are able to represent the illumination information captured with the technique described in chapter 3 on a mobile device extraordinarily well. For higher specular shininess coefficients, an approach based on the accurate method with bounds optimization will probably better fulfill demands in terms of performance and quality. It is also suitable for an additional performance increase by the approach described in this section.

CHAPTER 7

Conclusion and Future Work

In this thesis we have addressed the subject of realistic rendering on mobile devices. Our novel approach allows us to capture an omni-directional HDR light probe with a commodity mobile device and calculate irradiance and reflection maps, which can be used in image-based lighting for realistic rendering. We have presented efficient implementations for three different methods to calculate these maps and use them to render virtual objects realistically in an AR scene, using light information from the real world. Capturing the real world illumination, processing it into irradiance and reflection maps, and rendering the AR scene with that lighting information is all combined into a single mobile application and does not require any additional hardware.

The three different methods to calculate the irradiance and reflection maps are: A method performing accurate calculation, a MIP-mapping based approximation method, and a method which performs the computations in spherical harmonics frequency space. We have analyzed the quality and performance of these methods and described the strengths and weaknesses of each. The accurate method naturally yields the best quality results but takes long time to compute. Its bounds optimized version and the SH-based method, however, are capable of producing comparable quality depending on the specular shininess coefficient while computing the results faster. The SH-based method is able to calculate diffuse or low glossy illumination very efficiently. The accurate method with bounds optimization has its strengths with higher specular shininess coefficients. The MIP-mapping based method is faster than both, but it delivers inferior quality results.

There are quite a few starting points for future work building upon our technique. A direct consequence of our findings in section 6.2.2 would be to convert a captured environment map into a different format like a cube map or a dual paraboloid map and adapt the bounds optimization to it. We are very confident that this would result in an additional performance benefit since the bounds can be better optimized with an environment map format which has fewer distortions and no singularity. In addition to that, the approach described in section 6.2.3 should be implemented. It could be used to smoothly scale between the MIP-mapping based results and the results from the accurate method with bounds optimization in terms of both, speed and quality.

As far as our frequency space based method is concerned, reasonable optimizations for future work would be a simplification of the shader calculations like done in [RH01a] in the context of irradiance maps. Furthermore, evaluating different sets of basis functions – like the $\mathcal{H}$-basis [HW10] – could lead to more efficiency as well. A mathematical challenge would be to investigate whether the precision problems (described in section 6.2.1) with respect to spherical harmonics computations could be rewritten in a way that handles the concerned computations smartly. This would lead to higher SH orders being utilizable for our technique.

Environment map capturing is currently a separate process from irradiance and reflection map calculation. By trying to merge these two steps, a more user-friendly and more practical solution could be developed which would optimally calculate irradiance and reflection maps concurrently to environment map capturing. By additionally trying to speed up environment map capturing, this technique could enable new areas of application by providing near real-time irradiance and reflection maps for AR applications on commodity mobile devices.

With increasing computational power in mobile devices and AR applications getting more and more popular, the demand for realistic rendering within AR can be expected to rise accordingly. Realistic and visual coherent rendering in AR requires capturing real-world illumination information. This thesis describes a novel technique to capture environment maps with a commodity mobile device, and gives in-depth information for choosing the proper method of calculating irradiance and reflection maps for realistic image-based lighting. Detailed implementation instructions are provided along with starting points for future research based thereon, and background information on the most relevant topics from several fascinating areas of visual computing.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Add13] Neil Addison. Spherical harmonic lighting, 2013. The University of Warwick.

[AMHH08] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-time rendering*. CRC Press, 2008.

[Azu97] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

[Bjo04] Kevin Bjorke. *Image-Based Lighting*, chapter 19. Pearson Higher Education, 2004.

[BN76] James F Blinn and Martin E Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, 1976.

[CK07] Mark Colbert and Jaroslav Křivánek. *GPU-Based Importance Sampling*, chapter 20. Addison-Wesley Professional, 2007.

[Deb98] Paul Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 1998, pages 189–198, New York, NY, USA, 1998.

[Deb01] Paul Debevec. Light probe image gallery. In *Proceedings of SIGGRAPH*, volume 98, 2001.

[Deb05] Paul Debevec. Image-based lighting. In *ACM SIGGRAPH 2005 Courses*, page 3. ACM, 2005.

[DWWW08] Xiaoming Deng, Fuchao Wu, Yihong Wu, and Chongwei Wan. Automatic spherical panorama generation with two fisheye images. In *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pages 5955–5959. IEEE, 2008.

[FGR93] Alain Fournier, Atjeng S Gunawan, and Chris Romanzin. Common illumination between real and computer generated scenes. In *Graphics Interface*, pages 254–254. CANADIAN INFORMATION PROCESSING SOCIETY, 1993.

[FK03] Randima Fernando and Mark J Kilgard. *The Cg Tutorial: The definitive guide to programmable real-time graphics.* Addison-Wesley Longman Publishing Co., Inc., 2003.

[Fra13] Tobias Alexander Franke. Delta light propagation volumes for mixed reality. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pages 125–132. IEEE, 2013.

[Fra14] Tobias Alexander Franke. Delta voxel cone tracing. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 39–44. IEEE, 2014.

[GEM07] Thorsten Grosch, Tobias Eble, and Stefan Mueller. Consistent interactive augmentation of live camera images with correct near-field illumination. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pages 125–132. ACM, 2007.

[GKD07] Paul Green, Jan Kautz, and Frédo Durand. Efficient reflectance and visibility approximations for environment map rendering. In *Computer Graphics Forum*, volume 26, pages 495–502. Wiley Online Library, 2007.

[GM00] Simon Gibson and Alan Murta. *Interactive rendering with real-world illumination.* Springer, 2000.

[Gra03] Kris Gray. *Microsoft DirectX 9 programmable graphics pipeline.* Microsoft Press, 2003.

[Gre86] Ned Greene. Environment mapping and other applications of world projections. *Computer Graphics and Applications, IEEE*, 6(11):21–29, 1986.

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

[Gro05] Thorsten Grosch. Differential photon mapping: Consistent augmentation of photographs with correction of all light paths. In *Eurographics*, pages 53–56, 2005.

[Hal04] Michael Haller. Photorealism or/and non-photorealism in augmented reality. In *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 189–196. ACM, 2004.

[Han05] Pat Hanrahan. Radiometry. University Lecture, 2005.

[HDH03] Michael Haller, Stephan Drab, and Werner Hartmann. A real-time shadow approach for an augmented reality application using shadow volumes. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 56–65. ACM, 2003.

[HFJS09] Timothy J Hattenberger, Mark D Fairchild, Garrett M Johnson, and Carl Salvaggio. A psychophysical investigation of global illumination algorithms used in augmented reality. *ACM Transactions on Applied Perception (TAP)*, 6(1):2, 2009.

[HS98] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 39–ff. ACM, 1998.

[HS99] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 171–178. ACM Press/Addison-Wesley Publishing Co., 1999.

[HW10] Ralf Habel and Michael Wimmer. Efficient irradiance normal mapping. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 189–195. ACM, 2010.

[ICG86] David S Immel, Michael F Cohen, and Donald P Greenberg. A radiosity method for non-diffuse environments. In *ACM SIGGRAPH Computer Graphics*, volume 20, pages 133–142. ACM, 1986.

[Kaj86] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[Kán14] Peter Kán. *High-Quality Real-Time Global Illumination in Augmented Reality.* PhD thesis, Institute of Software Technology and Interactive Systems, 2014.

[Kán15] Peter Kán. Interactive hdr environment map capturing on mobile devices. In *EUROGRAPHICS 2015 Short papers*, pages 29–32. Eurographics Association, 2015.

[KBG$^+$15] Joel Kronander, Francesco Banterle, Andrew Gardner, Ehsan Miandji, and Jonas Unger. Photorealistic rendering of mixed reality scenes. In *Computer Graphics Forum*, volume 34, pages 643–665. Wiley Online Library, 2015.

[KDB$^+$14] Peter Kán, Andreas Dünser, Mark Billinghurst, Christian Schönauer, and Hannes Kaufmann. The effects of direct and global illumination on presence in augmented reality. In *Challenging Presence - Proceedings of 15th International Conference on Presence (ISPR 2014)*, pages 223–230. Facultas Verlags- und Buchhandels AG, 2014.

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

[KHFH11] Kevin Karsch, Varsha Hedau, David Forsyth, and Derek Hoiem. Rendering synthetic objects into legacy photographs. In *ACM Transactions on Graphics (TOG)*, volume 30, page 157. ACM, 2011.

[Khr14] Khronos Group. OpenGL ES Software Development Kit. https://www.khronos.org/opengles, 2014. [Online; accessed 11-May-2016].

[Kin05] Gary King. *Real-Time Computation of Dynamic Irradiance Environment Maps*, chapter 10. Addison-Wesley Professional, 2005.

[KK12a] Peter Kán and Hannes Kaufmann. High-quality reflections, refractions, and caustics in augmented reality and their contribution to visual coherence. In *Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 99–108. IEEE Computer Society, 2012.

[KK12b] Peter Kán and Hannes Kaufmann. Physically-based depth of field in augmented reality. In *Proceedings of EUROGRAPHICS 2012*, pages 89–92. Eurographics Association, 2012.

[KK13a] Paul Kan and Hannes Kaufmann. Differential irradiance caching for fast high-quality light transport between virtual and real worlds. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pages 133–141. IEEE, 2013.

[KK13b] Peter Kán and Hannes Kaufmann. Differential progressive path tracing for high-quality previsualization and relighting in augmented reality. In George Bebis, editor, *ISVC 2013, Part II, LNCS 8034*, pages 328–338. Springer-Verlag Berlin Heidelberg, 2013.

[KTM$^+$10] Martin Knecht, Christoph Traxler, Oliver Mattausch, Werner Purgathofer, and Michael Wimmer. Differential instant radiosity for mixed reality. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 99–107. IEEE, 2010.

[KUK15] Peter Kán, Johannes Unterguggenberger, and Hannes Kaufmann. High-quality consistent illumination in mobile augmented reality by radiance convolution on the gpu. In *International Symposium on Visual Computing*, pages 574–585. Springer, 2015.

[Lam13] Kenny Lammers. *Unity shaders and effects cookbook*. Packt Publishing Ltd, 2013.

[Lew94] Robert R Lewis. Making shaders more physically plausible. In *Computer Graphics Forum*, volume 13, pages 109–120. Wiley Online Library, 1994.

[Lip13] Benj Lipchak. *OpenGL ES Version 3.0.2*. The Khronos Group Inc., 2013.

[Luk07] Christian Luksch. Realtime hdr rendering. *Graduation Project. Institute of Computer Graphics and Algorithms, TU Vienna*, 2007.

[LW94] Eric P Lafortune and Yves D Willems. *Using the modified phong reflectance model for physically based rendering.* Citeseer, 1994.

[LZ12] Sébastien Lagarde and Antoine Zanuttini. Local image-based lighting with parallax-corrected cubemaps. In *ACM SIGGRAPH 2012 Talks*, page 36. ACM, 2012.

[Mac28] TM MacRobert. Spherical harmonics. an elementary treatise on harmonic functions. *Bull. Amer. Math. Soc. 34 (1928), 779-780 DOI: http://dx. doi. org/10.1090/S0002-9904-1928-04648-7 PII*, pages 0002–9904, 1928.

[Mac48] Thomas Murray MacRobert. *Spherical harmonics: An elementary treatise on harmonic functions, with applications.* Dover Publ., 1948.

[Mat03] Wojciech Matusik. *A data-driven reflectance model.* PhD thesis, Massachusetts Institute of Technology, 2003.

[MBGN98] Tom McReynolds, David Blythe, Brad Grantham, and Scott Nelson. Advanced graphics programming techniques using opengl. *Computer Graphics*, pages 95–145, 1998.

[MBGN99] Tom McReynolds, David Blythe, B Grantham, and S Nelson. Advanced graphics programming techniques using opengl. siggraph course notes, 1999.

[McA04] David McAllister. *Spatial BRDFs*, chapter 18. Pearson Higher Education, 2004.

[MH84] GENES Miller and C Robert Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. In *SIGGRAPH 84 Advanced Computer Graphics Animation seminar notes*, volume 190, 1984.

[MLH02] David K McAllister, Anselmo Lastra, and Wolfgang Heidrich. Efficient rendering of spatial bi-directional reflectance distribution functions. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 79–88. Eurographics Association, 2002.

[MPBM03] Wojciech Matusik, Hanspeter Pfister, Matthew Brand, and Leonard McMillan. Efficient isotropic brdf measurement. 2003.

[PH03] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 340–349. ACM, 2003.

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

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[PMDS06] Voicu Popescu, Chunhui Mei, Jordan Dauble, and Elisha Sacks. Reflected-scene impostors for realistic reflections at interactive rates. In *Computer Graphics Forum*, volume 25, pages 313–322. Wiley Online Library, 2006.

[PML$^+$10] Saulo Pessoa, Guilherme Moura, Joao Lima, Veronica Teichrieb, and Judith Kelner. Photorealistic rendering for augmented reality: A global illumination and brdf solution. In *Virtual Reality Conference (VR), 2010 IEEE*, pages 3–10. IEEE, 2010.

[RBS99] Mark A Robertson, Sean Borman, and Robert L Stevenson. Dynamic range improvement through multiple exposures. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 3, pages 159–163. IEEE, 1999.

[RH01a] 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.

[RH01b] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 117–128. ACM, 2001.

[RH02] Ravi Ramamoorthi and Pat Hanrahan. Frequency space environment map rendering. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 517–526. ACM, 2002.

[RSSF02] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Transactions on Graphics (TOG)*, 21(3):267–276, 2002.

[Sch05] Volker Schönefeld. Spherical harmonics. *Computer Graphics and Multimedia Group, Technical Note. RWTH Aachen University, Germany*, 2005.

[SG$^+$09] Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1.* Pearson Education, 2009.

[SKAL08] László Szirmay-Kalos, Barnabás Aszódi, and István Lazányi. Ray tracing effects without tracing rays. *shaderX*, 4, 2008.

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

[SNRS12] Daniel Scherzer, Chuong H Nguyen, Tobias Ritschel, and Hans-Peter Seidel. Pre-convolved radiance caching. In *Computer Graphics Forum*, volume 31, pages 1391–1397. Wiley Online Library, 2012.

[SS97] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 251–258. ACM Press/Addison-Wesley Publishing Co., 1997.

[Ste05] Mauro Steigleder. *Pencil Light Transport*. PhD thesis, University of Waterloo, 2005.

[War92] Gregory J Ward. Measuring and modeling anisotropic reflection. *ACM SIGGRAPH Computer Graphics*, 26(2):265–272, 1992.

[WD02] Jamie Waese and Paul Debevec. A real-time high dynamic range light probe. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques: Conference Abstracts and Applications*, page 247, 2002.

[Wil83] Lance Williams. Pyramidal parametrics. In *ACM Siggraph Computer Graphics*, volume 17, pages 1–11. ACM, 1983.

[WMLS10] Daniel Wagner, Alessandro Mulloni, Tobias Langlotz, and Dieter Schmalstieg. Real-time panoramic mapping and tracking on mobile phones. In *Virtual Reality Conference (VR), 2010 IEEE*, pages 211–218. IEEE, 2010.

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

[WWL07] Liang Wan, Tien-Tsin Wong, and Chi-Sing Leung. Isocube: Exploiting the cubemap hardware. *Visualization and Computer Graphics, IEEE Transactions on*, 13(4):720–731, 2007.

[WWLL06] Tien-Tsin Wong, Liang Wan, Chi-Sing Leung, and Ping-Man Lam. Real-time environment mapping with equal solid-angle spherical quad-map. *Shader X4: Lighting & Rendering, W. Engel, ed., Charles River Media*, 2006.

[ZDB08] Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 193–202. IEEE Computer Society, 2008.