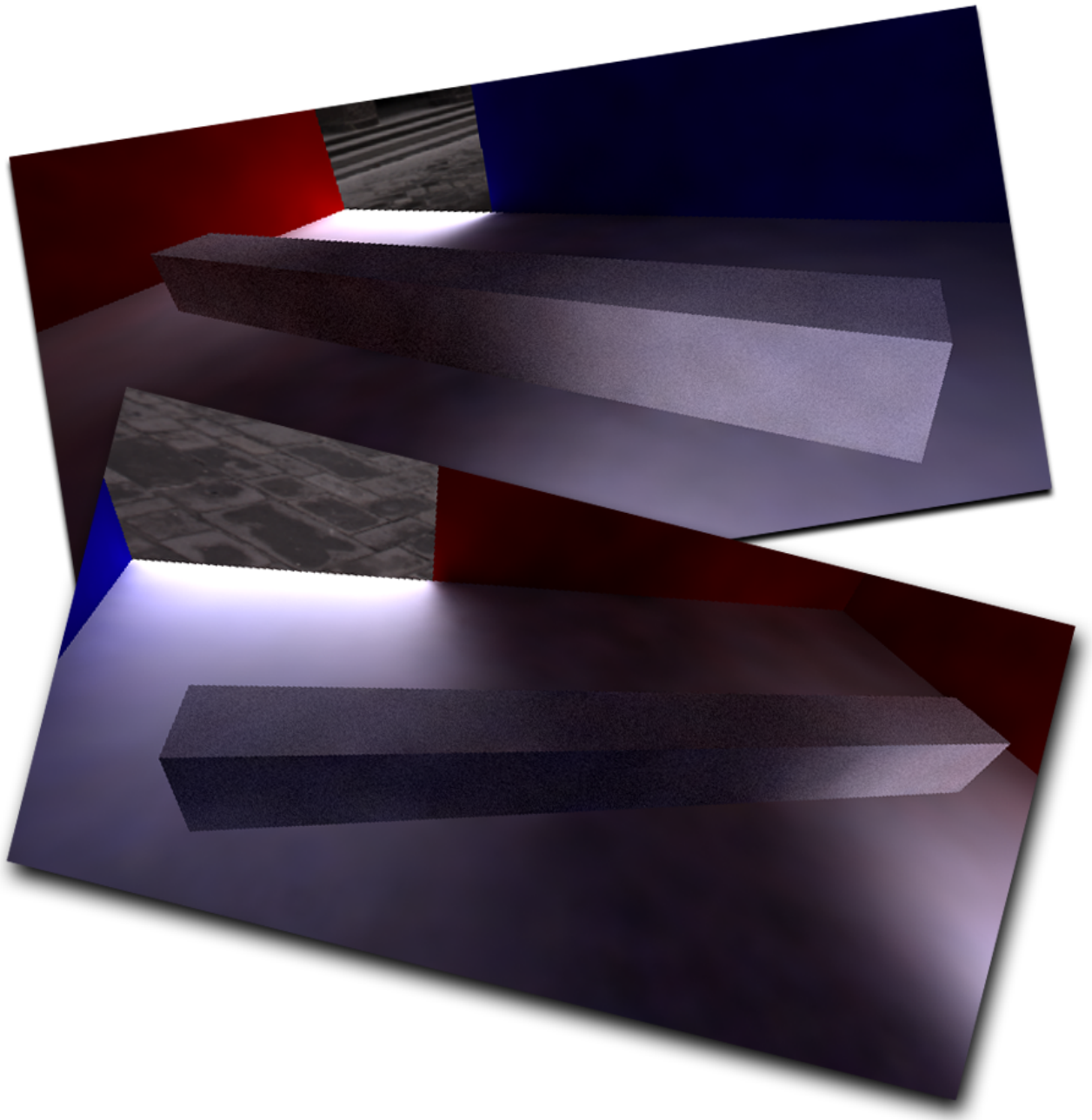
GDIBL

Implementation and verification
of geometrically derived
image based lighting



Kasper Vestergaard

Aalborg Universitet, 2008

AALBORG UNIVERSITET

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Ø
Telephone: (+45) 9940 9940
Telefax: (+45) 9940 9798
http://www.cs.aau.dk

Title:
GDIBL: Implementation and results

Topic:
Image-based illumination of synthetic objects

Project period:
INF8, summer 2008 (extended)
April 13th - November 1st

Project group:
N/A

Members of the group:
Kasper Vestergaard

Supervisor:
Claus B. Madsen

Number of copies: 3

Pages: 57

Synopsis:

This report documents the implementation and further studies in succession to the preliminary research and theory outline of the GDIBL method presented in the spring of 2008. Geometrically Derived Image Based Lighting reference multiple light probes through geometric intersection for illuminating a synthetic scene in 3D computer graphics.
The implementation is presented in such technical detail as is needed to confirm a valid application of the method previously described only in theory.
This report presents both the implementation and the results to validate the effect of the proposed method through successful implementation.
The results of both the GDIBL rendering method and the native benefits of the method are presented through this report.

Preface

This report documents the preliminary research and theory outline for the subsequent master's thesis which was carried out at the 10th semester of Informatics at Aalborg University, during the summer and fall of 2008.

The reader is reminded that this is the second of two, where the first one described preliminary research and theoretic overview of the Geometrically derived image based lighting (GDIBL) method.

Supervision was provided by Claus B. Madsen from the department of Computer Vision and Graphics.

Reading guidelines

While equations are kept in coherence with mathematical standard notation, all algorithms are in C# or C# like pseudo-code in order to document the actual implementation. It is therefor favorable if the reader is familiar with the this language, but given the easily accessible nature of the C# language, the meaning and generic algorithm should be derivable.

References to sources are presented in a Harvard like fashion and throughout the report marked with author initials and year, e.g. [DM97]. The four character reference ID can then be found in the actual literature listing at page 51. Additionally a glossary can be found on page 56.

The enclosed disk

The enclosed disk contains the report at hand in PDF format for hypertext reading and digital image quality. It also includes source code for the GDIBL implementation in it's current state, as well as scene files and hight dynamic range image (HDRI) versions of the presented renderings.

Author

Kasper Vestergaard

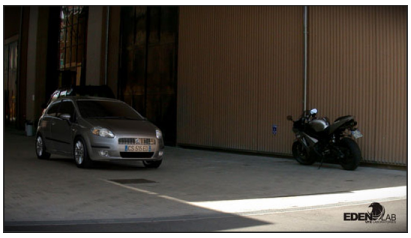
Contents

Chapter 1

Introduction

When the GDIBL method was first outlined [Ves08], the goal was to develop a method for embracing spatial variance in real world lighting for the use in computer graphics. The method is based on image based lighting (IBL) and utilize a rough geometric model of the local environment to derive correct lighting from multiple, arbitrarily scattered light samples.

As IBL has heightened the standard for realism in synthetic object rendering, the issue to address is it's lack of support for spatial diversity in the lighting conditions.



(a) The car is in the shade which means a visually uniform lighting environment from all points on the surface of the car model.

(b) The car moves gradually out of the shade and is no longer uniformly illuminated.

Figure 1.1: EDEN Lab have rendered this car for a commercial while also demonstrating the aspects of spatially variant lighting conditions.

Lighting characteristics may vary little or much depending on the scene. In some cases it will not be noticeable, but in cases like the one presented by EDEN Lab, Figure 1.1 the visual difference is very obvious.

The GDIBL method was laid out in theory and promised to derive location depending lighting, like the example above, requiring only a sparse number of light samples and a simple geometric approximation of the environment from which the samples are obtained. The work behind the report at hand is my effort to prove by practical implementation and discussed examples the validity of the GDIBL method.

1.1 Basis for implementation

The task of implementing the GDIBL method was chosen for two reasons. Mainly I had a genuine desire to validate the method and show that it was able to produce the expected result. In this reason was also included the curiosity to further investigate the method, it's theoretical limitations and practical workings. Secondly I have a B.sc. in informatics and have thus far only been superficially trained in computer graphics and practical programming. As a personal competence building project I wanted to focus on the implementation, on working with both theoretical and practical aspects of computer graphics and challenge my background to extend to the field that fascinates me the most.

1.2 Goals

Through this implementation project I wished to gain knowledge about the practical approach to three dimensional computer graphics and lighting. This was sought through the implementation and verification of the GDIBL method, previously described in theory.

Implementation from scratch

As the crucial aspect of the programming exercise was centered around the actual programming and structuring of a graphics application like the GDIBL implementation, the use of external libraries are kept at a minimum. This means a practical approach to the basic structure and foundational implementation of concepts, which I have so far only touched on a theoretical level, such as notions of space, synthetic objects and color.

Verification of the GDIBL method

The programming exercise was to implement, verify and demonstrate the GDIBL method. In order to properly verify the effect of the method, a valid implementation must be documented.

This is done through algorithmic documentation as well as actual source code examples and linking this to the relevant theory of both illumination in general and the theoretical definition of the GDIBL method.

Additionally the results of the methods treatment of test scenarios must produce the expected results. Such results can take both graphical and numerical form as to best describe the validity of the calculations.

1.3 Structure and contents

Initially the theory and a method overview is presented along with a short walk through of relevant terminology, related technologies and previous

work.

Then follows an overview of the GDIBL method and the algorithms used in the implementation before a description of the actual application. The method section also outlines various additional issues which has been discovered through the work carried out through this project.

Application structures and actual implementation are presented in such technical detail as is needed to confirm a valid application of the method previously described only in theory.

Subsequently, results and further discussions provide both visual and technical confirmation of the effect of the GDIBL method. The discussion includes a series of suggestions on further development and application areas are pointed out along with the final conclusion on this second part of the GDIBL project.

Chapter 2

Theory

In this chapter the theoretical basis for the subsequent descriptions and methods is described.

Illumination in computer graphics is described with the purpose of presenting the terminology and illumination models related to real world light distribution and illumination calculation in 3D computer graphics.

Then IBL is presented focusing on the theoretical foundation and practical application as well as the limitation sought to be solved through the GDIBL method.

Lastly the challenge of reconstructing spatial variation in complex lighting conditions is discussed. The focus of this section is to point out the issues and theoretical challenge that lie lighting conditions that vary spatially.

2.1 Illumination

In order to keep the goal of realistic lighting in mind, this section review some of the relevant theory and notation of modeling real world lighting and the relevance of those models to global illumination (GI) and IBL rendering. In the following section light interaction is described with two purposes; one is to clarify the phenomena the GDIBL method approximates and secondly to specify exactly which light properties the actual application implements.

As light source, material and surface geometry properties are all relevant to an illumination model, there are various assumptions that can be made in order to drastically simplify the implemented model while still providing sufficient data for validating the effect of the GDIBL method. Illumination models can initially be separated into two categories; local and global illumination models. While local illumination models only deal with light arriving directly from the light source, global illumination models also consider light that is reflected between the surfaces of objects in the scene.

Initially this section describes the basic definitions of lighting and illumination effects. It then turns its focus towards the scattering and distribu-

tion of light as well as models used to describe relevant surface properties.

As this project is based on IBL parameters of classic light sources such as point and area lights are omitted. However, the use of images as a source of lighting is described in section 2.2.

2.1.1 Basic definitions

As many different sources of light occur in nature and the world we have build around us, all these light sources share some characteristics that enable us to model them for both physics and graphical purposes. Traditionally computer graphics has been concerned with three types of light sources; point, spot light and directional.

The point light source emits light uniformly in all directions while a spot light only emits within a cone in some given direction. While these two light source types are both based in a single point, directional light emits parallel rays of light as to imitate light from a source very far away, like the sun. More recent light source types additions count area lights, light emissive objects and environment lighting like sky-domes or IBL, described in section 2.2.

Whether scattered from a single point, orthogonally from an abstract plane, from a spatial object or purely directional, light has other attributes of relevance like color and intensity. As color represents the wavelengths emitted from the light source and intensity defines the power or energy level of the light, these attributes basically defines the entire light emission part of the equation. The further traveling, object interaction and behavior of that emitted light depend among others on the following concepts:

Direct light. This term is used about light that arrives on a surface directly from the light source. It means that the light is not reflected off and has not passed through other objects or media (atmosphere usually not included).

Indirect light. All light in the scene that is not direct light. I.e. all light that has been reflected, refracted or in other ways has had previous encounters with objects in the scene. Indirect light is also referred to as ambient light.

Reflected light. As mentioned above is the light that has arrived at a surface and has been reflected back into the scene. This is not necessarily all light; most light models take absorbtion into account in some form or another.

As we cannot see a beam of light itself we are left with observing the effects of light on object surfaces such as skin, wood, broken down through a computer screen or, on a microscopic level, through reflection and refraction in tiny water drops in fog. For this reason it is extremely important how we model the properties and behavior of those surfaces.

Diffuse reflection is where the light is scattered in all directions. A special case of diffuse reflection is Lambertian or perfect diffuse where the light is scattered *uniformly* in all directions. This type of surface is very easy to implement but does not exist to the theoretical definition in the real world.

Specular reflection is where light is reflected directly in the opposite angular direction from which it arrived. This models a mirror like reflection and does occur in the real world, e.g. in metals.

As most materials behave with qualities of both models, they can be represented as a combination of the two.

Other phenomena that affect the dispersion of light throughout a scene is the way the light is affected by the surfaces of the objects.

When the environment can be seen vaguely on the surface of an object, this less than mirror perfect reflection is called glossy. A glossy reflection is a combination of diffuse and specular and results in a blurry reflection. The visual result of a glossy reflection in computer generated imagery (CGI) depends on the implementation of the physical phenomenon where the rays reflected in the surface comes from a more or less narrow cone of incoming angles. This means that in a realistic implementation objects will appear more distinct the closer they get to the reflecting surface.

In the physical properties of a transparent material the light rays are affected by the change of density in the media. So when light arrives at a transparent surface, some of it is reflected back into the scene, some of it is absorbed and some of it is refracted down into the object. This is usually demonstrated with glass spheres, prisms or a glass of water. The refracted angle is determined by the incoming angle and the change in density of the materials. This also means that if the incoming angle is too small and the density change is too high, the refracted ray also reflects back into the scene. This is called Fresnel reflection and also occurs in materials that are not transparent, such as paper and wood. The general effect of this type of reflection is that more light is reflected in shallow angles.

When a ray of light is reflected or refracted in a surface, it might be affected by the color of the surface it interacted with. This is called color bleeding, and despite the macabre term it defines the effect of colors from one objects transmitted more or less visibly to another. This effect is often demonstrated through the Cornell box, Figure 2.1, where the white light is reflected red and blue from each wall respectively.

Another effect demonstrated in Figure 2.1 is the intense light gathering under the glass sphere. This effect is called caustics and defines the phenomenon where multiple light rays are reflected or refracted to the same area, creating a bright shape. All reflective and refractive objects produce these even though they are not always sufficiently intense for us to observe them.

The last of the evident effects demonstrated in the Cornell box example is shadow. Shadows are simply the absence of light but what distinguish

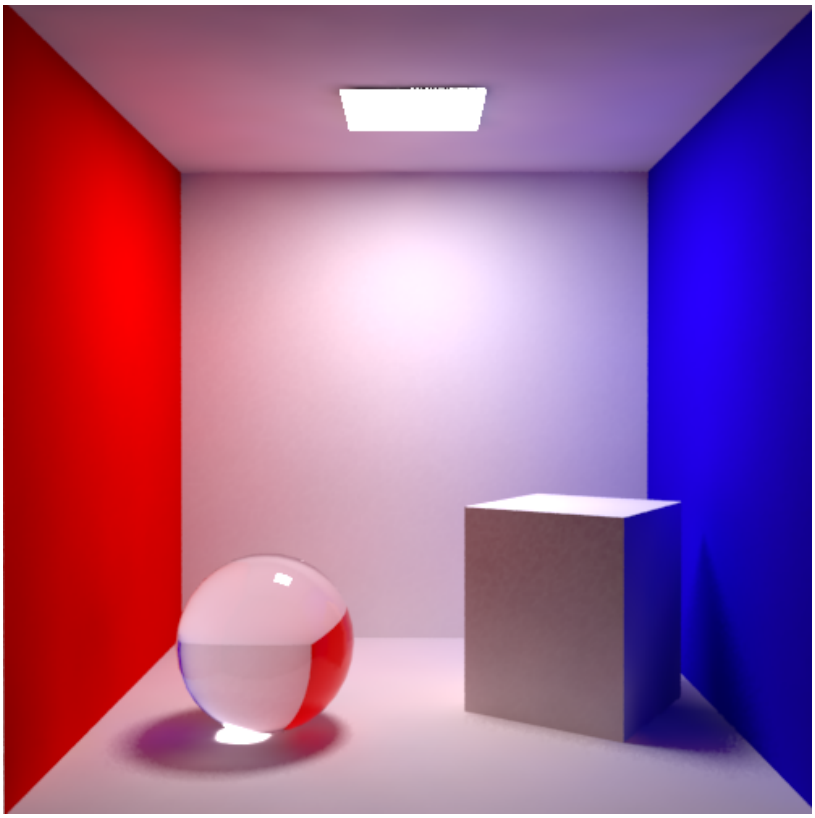


Figure 2.1: The modified Cornell box demonstrates both color bleeding, caustics and shadow phenomena.

shadows from mere darkness is that shadows are located *on* a surface and are defined the occlusion of a light source by an object. This means that there can be a shadow within a shadow and that shadows can add to each other if there are more than one light source that is occluded as shown in Figure 2.2.

The occluding object does not have to be an opaque object as our perception of a shadow is given by the contrast to a more brightly lit area or the idea of such an area. This means that even though a piece of frosted glass does not block all light it still leaves a shadow on the surface below.

Point lights is defined by not having any spatial extent and can an occluding object can therefor not produce soft shadows. If, on the other hand the light source does have a spatial extent, like an area light or a light emitting object, shadows from an occluder consists of two area types; the umbra, which is the area where all light from the given light source is completely blocked and the penumbra where only part of the light is blocked. These areas are illustrated in Figure 2.2 where a, b and c are placed through the areas of the shadow produced by the cylinder in the right. As the hard shadow produced by the box occluding a point light from behind, this shadow does not have any penumbra since light from this light source can not be partially blocked by an opaque object. The

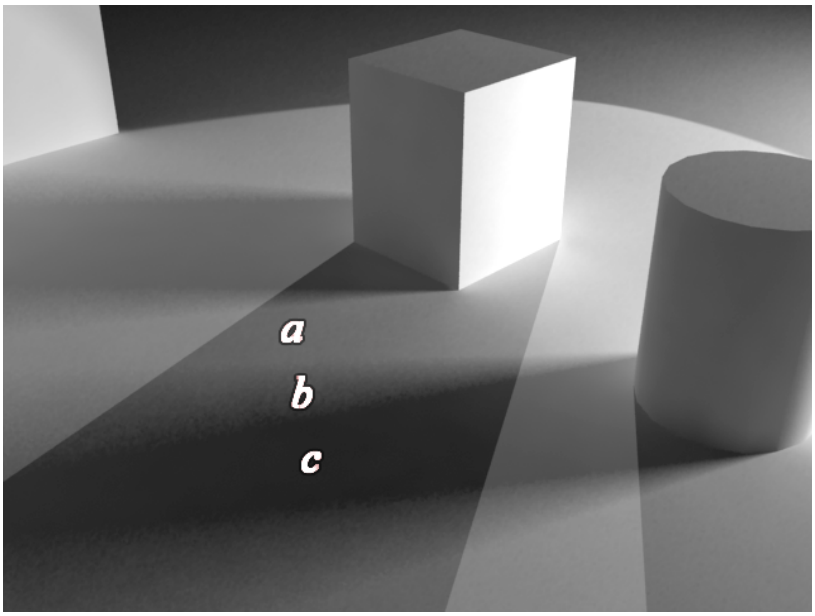


Figure 2.2: Overlapping shadows also showing the difference of hard and soft shadows

light coming from the right is emitted from an area light and does therefor have a spatial extent.

At a the light source is not blocked at all, where as the cylinder begins to block out the light arriving at point b. And finally the entire light emitting plane is blocked an no light from that light source arrives in point c.

2.1.2 Lighting terminology

Though a number of physical models of light has been proposed and published over the years, the ray optics model seems predominant in the field of computer graphics. This model does not include all observed behaviors of light but is far sufficient for simulating most visual effects of light as well as material interactions. It assumes that light travels in a straight line as it passes through a uniform media. Ray optics is therefor the basis for the following descriptions. Radiometry is used to model the radiation and irradiation of light and is well suited to match the ray notion of ray optics.

As [Jen01] and [Wu08] introduce the concepts and terminology of light and light modeling, this section is widely a direct reference to these introductions.

Notation

In ray optics x denotes a location on a surface. $\vec{n}$ is the normal vector at x while $\vec{\omega}_i$ is the unit vector representing the incoming direction of light and $\vec{\omega}$ is the direction of reflected light.

For sampling the environment off a surface the unit hemisphere covers 2π steradian and is noted as $\Omega_{2\pi}$. Thus follows that $\Omega_{4\pi}$ is a unit sphere

covering the entire 4π steradian that is all possible angles from one point in the three dimensional space. In integration over finite solid angles $d\omega$ denotes the differential solid angle.

When working with spheres, hemispheres and solid angles spherical coordinates are handy to use. A direction in spherical coordinates are defined as $\vec{\omega} = (\theta, \phi)$. A subscribed i and r are logically used to indicate the *incoming* and *reflected* version of the given concept. E.g. the above becomes $\vec{\omega}_i = (\theta_i, \phi_i)$ for the incoming direction.

In all the following definitions and equations square brackets [and] are used to encapsulate the unit of the entity or expression.

Radiometry

In radiometry light is seen as electro magnetic energy with a given wavelength. The basic quantum of this energy is called a photon. A photon with wavelength λ has an associated energy e_λ and thus the spectral radiant energy of n photons of the same wavelength λ is defined as $Q_\lambda = n_\lambda e_\lambda$; the radiant energy for a specific wavelength. Symbols, entities and associated units used in radiometry are listed in Table 2.1.

Symbol	Description	Unit
Q	Radiant energy	$[J]$
Q_λ	Spectral radiant energy	$[J/m]$
Φ	Radiant flux	$[W]$
Φ_λ	Spectral radiant flux	$[W/m]$
$B(x)$	Radiosity of surface location x	$[W/m^2]$
$E(x)$	Irradiance at surface location x	$[W/m^2]$
$L(x, \vec{\omega})$	Radiance at surface location x in direction $\vec{\omega}$	$[W/(m^2 \cdot sr]$

Table 2.1: Symbols used in radiometry.

The term, radiant energy defines the quantity of energy arriving at, moving through or is emitted from a surface of given area in a given period of time. The radiant energy is in other words the collective amount of energy of all photons entering or leaving that surface area. We get this value by integrating the spectral radiant energy for all possible wavelengths:

$$Q = \int_0^\infty Q_\lambda d\lambda \quad [J/m] \tag{2.1}$$

Flux is a general term to denote the *rate* of which energy or particles are transferred across a given surface. The radiant flux is therefor the quantity of energy transferring through a surface or region of space per time and is given by;

$$\Phi = \frac{dQ}{dt} \quad [W] \tag{2.2}$$

Radiosity, B, is the light energy leaving a surface, the emitted radiant flux, while the incoming radiant flux, E, is called irradiance. If a surface does not absorb or transmit ligh, like metal, then $B = E$; the *entering* light is *beamed* away again. While irradiance is given by

$$E(x) = \frac{d\Phi}{dA} \quad [W/m^2] \tag{2.3}$$

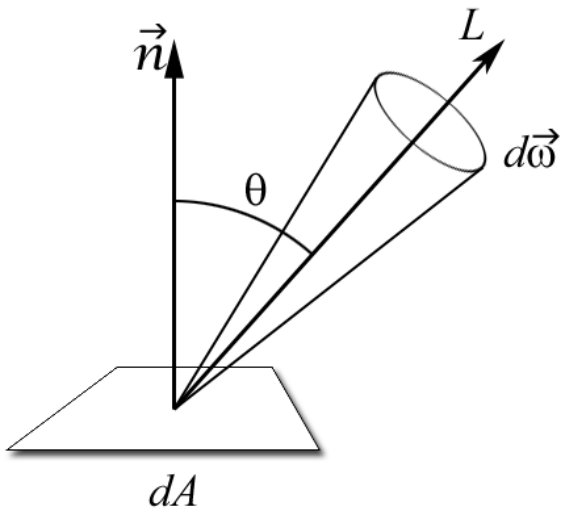


Figure 2.3: Radiance, L, is the radiant flux per unit solid angle, $d\omega$, per unit projected area, dA.

While radiosity is the energy flow at a given surface, radiance is the amount of energy, or number of photons arriving at or leaving the surface in a given direction. In relation to Figure 2.3, radiance is, formally speaking the radiant flux, Φ, per solid angle, $d\vec{\omega}$, per projected area, dA, and is given by

$$L(x, \vec{\omega}) = \frac{dE(x)}{\cos\theta d\vec{\omega}} = \frac{d^2\Phi}{\cos\theta d\vec{\omega} dA} \quad [W/m^2 \cdot sr] \tag{2.4}$$

The phenomenon that the same amount of light is spread over a larger area when more shallow incoming angles are reached (highter θ) is expressed by the cosine factor in the denominator. Radiometry is focused on getting pure light behaviors correct and does not approximate to include observable atmospheric disturbances within the light model itself. An important property of radiance is that in vacuum, it remains constant along a line of sight. This means that photons are not dispersed, they do not loose energy and they never disappear entirely. According to [Jen01] this is used by all ray tracing algorithms. Atmospheric disturbances and electromagnetic interference must therefor be separately implemented if such effects are desired.

2.1.3 Light scattering

With the terms and attributes of light in place follows a description of light-surface interaction. As the appearance of an object depends on the way it reflects light into the eye of the beholder, this area is focused around meaningfully representing and handling the reflection properties of any given material. In computer graphics the bi-directional surface scattering reflectance distribution function (BSSRDF) is as the fundamental tool to describe reflection characteristics. Usually the simplified bi-directional reflectance distribution function (BRDF) is far sufficient to describe and reconstruct any surface that is not visually translucent, while BSSRDF can be utilized to additionally describe the translucent properties of materials like frosted glass, milk, wax and the like. Physically speaking, only metal does not to some degree scatter light beneath the surface, but for graphic reconstruction this property is rarely visible and the BRDF is used for simplicity and render speed. In this overview, I only focus on the BRDF.

The BRDF is a function for describing how much of the incident light is reflected from the surface. This fraction of incident light, which is not absorbed by or transmitted through the material is called the reflectance, $\rho(x)$. The generic BRDF is based on the notion of wavelengths, but in most computer graphics implementations this feature is approximated by using a BRDF for each RGB-component[1].

Formally BRDF defines the relationship between differential reflected radiance and differential irradiance

$$f_r(x, \vec{\omega}_i, \vec{\omega}) = \frac{dL_r(x, \vec{\omega})}{dE(x, \vec{\omega}_i)} = \frac{dL_r(x, \vec{\omega})}{L_i(x, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n})d\vec{\omega}_i} \quad [sr^{-1}] \tag{2.5}$$

The reflection radiance in all directions can be calculated if the incident radiance field at a given surface location is known. This is done by rearranging equation 2.5 and integrating the incident radiance, L_i, over the entire hemisphere;

$$\begin{aligned} L_r(x, \vec{\omega}) &= \int_{\Omega 2\pi} f_r(x, \vec{\omega}_i, \vec{\omega}) dE(x, \vec{\omega}_i) \\ &= \int_{\Omega 2\pi} f_r(x, \vec{\omega}_i, \vec{\omega}) L_i(x, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) d\omega_i \end{aligned} \tag{2.6}$$

Helmholtz's law of reciprocity states that the BRDF is independent of the direction in which the light flows;

$$f_r(x, \vec{\omega}_i, \vec{\omega}) = f_r(x, \vec{\omega}, \vec{\omega}_i) \tag{2.7}$$

This means that the BRDF enables us to trace light in both directions. Another important physical property is energy conservation. Due to energy

[1]Recent development has produced rendering techniques with a more abstract concept of wavelength, usually referred to as spectral rendering[Don06][SFDC01][CPB06]

absorbtion and transmission the energy level can drop but never rise, unless the given surface is emissive in nature. In other words, a surface can not *reflect* more light than it receives.

As described in section 3.2.1, the GDIBL method is thought for lighting rather than providing an environment for specular reflections. For this reason Lambertian surfaces are used to test and demonstrate the validity of the method. For a Lambertian surface reflected radiance is constant in all directions over the hemisphere around the surface normal. This means that the light reflected off a Lambertian surface is independent of view direction and thus gives the constant BRDF

$$f_{r,d}(x) = \rho(x)/\pi \tag{2.8}$$

and by substitution in equation 2.6, the radiance is given by

$$L_r(x) = f_{r,d}(x) \int_{\Omega 2\pi} dE(x, \vec{\omega}_i) = f_{r,d}(x)E(x) \tag{2.9}$$

In section 5.1 and 5.2 a specular reflective material is used to illustrate the relative position of objects in the scene. As specular reflections are outside the scope of this project and since this material simply displays a maximum level of specularity, a simple vector reflection is used in the application.

2.1.4 The rendering equation

The rendering equation is the general equation for computing the radiance at any surface location in the modeled scene.

As stated in section 2.1.3 a surface can not reflect more light than it receives, *unless* it is light emissive in nature. Therefor the rendering equation states that the outgoing radiance, L_o, is the sum of the emitted radiance, L_e and the reflected radiance, L_r, which gives;

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + L_r(x, \vec{\omega}) \tag{2.10}$$

By substituting equation 2.6 for the reflected radiance, we get the rendering equation in the form often used in ray-tracing algorithms.

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_{\Omega 2\pi} f_r(x, \vec{\omega}_i, \vec{\omega}) L_i(x, \vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) d\omega_i \tag{2.11}$$

As this projects does not deal with environment interaction in IBL and also have no other use for emissive surfaces, this first part of the equation is disregarded. $L_i(x, \vec{\omega})$ in the second part of the equation represents the incoming radiance from other surfaces in the scene, and thus indicate the recursive nature of this equation.

2.2 Image Based Lighting

Today's motion picture productions including computer generated imagery has reached a standard where global illumination is required. The realism of the composited graphics must be unquestioned and the classic three point lighting days are over. At the same time the high standards for lighting in computer graphics have increasingly begun being applied to computer games and other real-time applications.

GI generically covers all methods that provide both direct and indirect illumination within a given scene. IBL is not directly categorized with the other GI methods as the indirect illumination is not *necessarily* a part of any IBL implementation. It does however manage to apply the indirect illumination from the surrounding environment of the scene.

As the name indicate, IBL is a technique where an image based sample of irradiance at some point in a real or synthetic scene is used to illuminate any synthetic objects that is rendered into that scene [Deb98]. In order to capture and reconstruct actual lighting conditions the image is required to store a high dynamic range of light. Though several formats provide support for hight dynamic range (HDR), a common property is that they typically store color information in somewhere from 16 to 48 bits per color channel as opposed to the traditional 8-bpc. The most common utilization of HDR images as a means of illuminating synthetic objects, is by having them represent the radiance from the entire scene. For this we need a panoramic image that describes the environment in a full 360° by 180° sphere. Throughout this report, such panoramic irradiation samples are referred to as *light probes*.

IBL represent all indirect light from the given environment, but introduce it to the local scene as direct light. As opposed to the classic light types in computer graphics; point and directional light. Since IBL is based on the notion of being infinitely far away, it is in fact always locally directional. I.e. a given direction is surveyed for light information it will always return the same value independent of the survey *location*. This does not classify IBL as a directional light per say, but rather as another model for irradiance completely.

IBL ensures a close approximation to the actual light in the scene, but has noticeable limitations in respect to flexibility and range of correct lighting conditions. If the synthetic objects move to an area with different lighting conditions than where the light probe recording was made, this will become very visible as seamless blending is only provided within visually identical lighting condition.

This issue is due to the fact that the light is "cast" onto the object from an infinite distance. This makes the IBL method indifferent to relative location as far as the object goes. It is sensitive to changes in relative *orientation*, but no matter where the object is placed, it will still have the exact same relative location to the surrounding environment as this remains at infinite distance ([Gre86], [Deb98]). This restriction in spatial

variance is the main concern for this project.

2.2.1 Acquisition

For HDRI recording several state-of-the-art devices provide fully automated HDR photography. On a lower budget the, by Debevec introduced traditional methods are still in use. As presented in [DM97] a series of low dynamic range (LDR) photographs of varying exposure can be combined into an HDR image. This leaves us with the acquisition of the panoramic aspect.

The mirror ball

A first surface reflective orb can be used to capture a reflection of the environment around it. Due to its spherical nature it reflects everything besides what is directly behind it. As everything usually include the photographer the orb should be photographed from at least two different angles in order be able to edit distortion and photographers. This method is fairly straight forward as, among others, both HDRI Shop[2] and Adobe Photoshop CS2[3] and newer will take care of automatically merging a series of LDR photographs into a variety of HDRIformats.

Fish eye lenses

A fish eye lens is a common term for an extreme wide angle lens providing up to a 180° view capture. Usually some dispersion or precision falloff appears towards the sides of the lens where the extreme angles can influence various wavelengths differently. As presented by [KB04] and used in the [Pro07] project, the output of such wide angle photography can be calibrated and any color distortion computed out. This is a somewhat more labor intensive method but pays back in both precision, resolution and level of detail.

Synthetic scenery

Rendering a full environment panorama can not be done with a classic pin-hole camera, but an angle-based camera is neither hard to comprehend nor implement. The commercial state-of-the-art renderers on the market today, have some variation of dome-camera, spherical rendering option or at least they provide an easy way to set up their standard camera type to provide the six different renderings that is needed for a cubic environment map as described in the following section. Rendering a synthetic scene into an environment map has numerous applications ([UGY06],[ML03],[Pro07]) and has the advantage that any level of detail can be reached as well as

[2] `http://www.debevec.org/hdrshop/`

[3] This feature has been improved through CS3 and now CS4; `http://help.adobe.com/en_US/Photoshop/11.0/WSfd1234e1c4b69f30ea53e41001031ab64-78e5a.html`

a pipeline without compromising photographers, changing lighting conditions or blind angles.

Regardless of acquisition techniques or initial pipeline, the actual mapping of the environment recording should be considered as well.

2.2.2 Environment mapping

In IBL the irradiance map is usually handled in the virtual scene as an environment map. An environment map is by definition [Gre86] mapped to an arbitrary, virtual object placed around the entire scene with its normals facing in towards the scene. This object is for reference rather than relative interaction and is not regarded as a part the rest of the actual scene. The environment is virtually at infinite distance from the objects in the scene, and so Debevec denoted this the difference between the *local scene* and the *distant environment*. This indefeasible distance property remains intact regardless of the mapping technique used to introduce the environment map or light probe t

Angular map

The angular mapping technique was introduced by Paul Debevec as the native format of light probes. The angular map is a circular map with a uniform differential angle from the center to the edge. This means that if the 1:1 aspect angular map is oriented towards the sky, the center of the map represents the direction straight up in the air, the 180◦ circle, located at exactly $\frac{r}{2}$ from the center is the horizontal line and the edge of the circular map, at a full r from the center, represents downward direction. This mapping is illustrated with a horizontal orientation in Figure 2.4

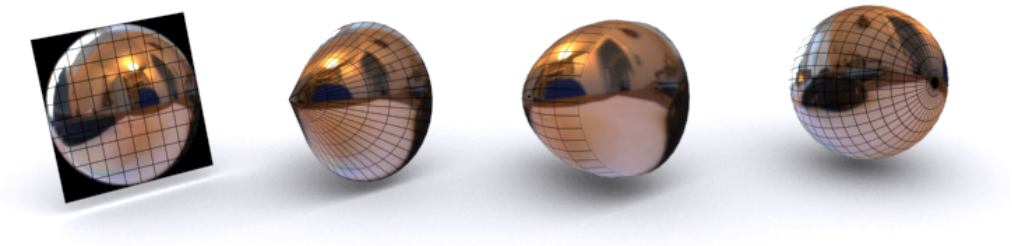


Figure 2.4: The 2D angular is first extended into a cone and then shrink wrapped to a sphere.

This mapping leaves us with no seems and only one pole. As this pole force the distinctive pixels around a far stretch, its singular role makes it deviate visibly from the detail level of the remaining image, followed only by the center of the map where the distinctive pixels are packed very tightly. In these extremes sample quality is either lost to compression or distorted by smearing. In addition the circular nature of the format also

makes it hard for the human eye to relate to the content and thus makes it nearly impossible to manually edit an image of this format.

Spherical coordinates

Spherical coordinates has been adopted from geology where the latitude and longitude coordinates of e.g. the Earth describe the surface traversal of all spherical objects. For this reason, an environment map stored in spherical coordinates are also referred to as *latitude-longitude*-map. The spherical map has an aspect of 2:1. Spherical coordinates, in math and physics notated by (r, ϕ, θ) are easily derived from any vector representation and thus makes a spherical map simple to reference. Another positive side to the spherical map is that the human eye better relates to this visual 2D representation than e.g. the angular map. On the down side the format does produce one seem and two poles in which smearing and loss of detail can occur, see Figure 2.5 for reference.

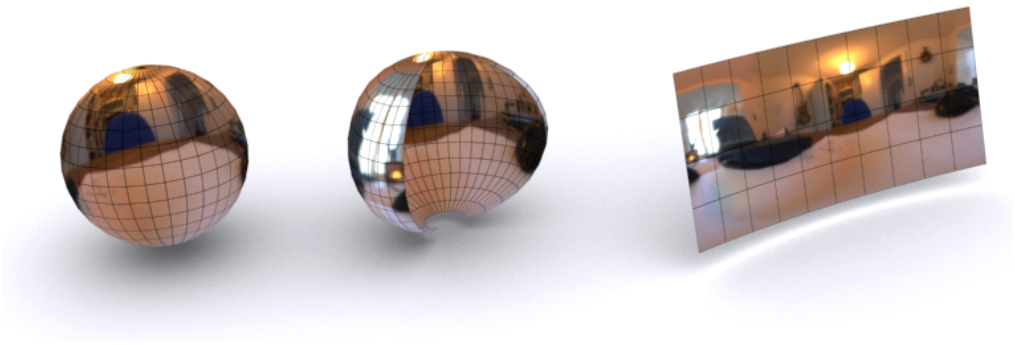


Figure 2.5: The sphere is opened. The small amount of pixels in the top and the bottom of the sphere are stretched to match the width of the 2D map.

The spherical coordinate approach has been supported in both OpenGL[4] and DirectX[5] for some time now and is thus hardware accelerated by all complying graphics cards.

Cubic environment

In a cubic map the environment is intuitively projected out onto a cube, producing six individual sides. When combined on a single 2D image, a common representation of these six sides is presenting them in a vertical or horizontal cross. As these six patches are square in nature, the cubic environment map has an aspect of 3:4 or 4:3 for the vertical and horizontal

[4] `http://www.opengl.org/documentation/specs/version1.1/state.pdf`
[5] `http://msdn.microsoft.com/en-us/library/bb147401(VS.85).aspx`

versions respectively. This mapping is very easy to reference as the vector coordinates directly transfers to the cubic pixel mapping mapping as illustrated in Figure 2.6.

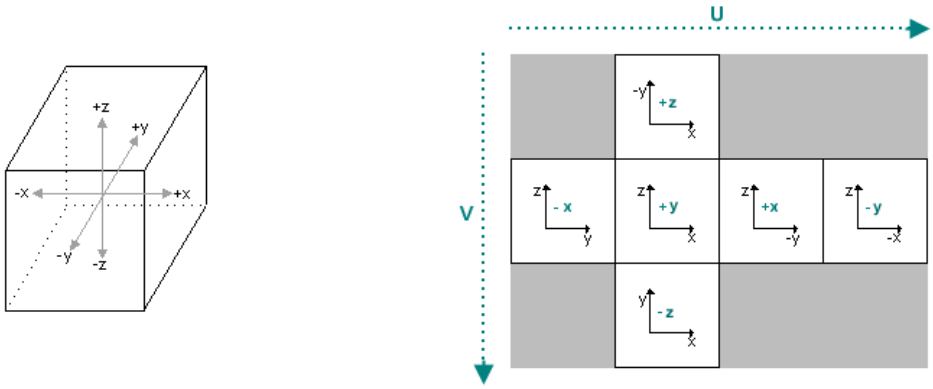


Figure 2.6: Vector coordinates can be directly transferred to the mapping of a cubic environment.

This format does provide eight seems but as it is in all aspects cubic, it has no poles, as illustrated in Figure 2.7. For this reason, in addition to the easy reference by computers, the individual patches of the cube map is also practically without distortion. This property makes it equally easy to view and edit for the human user.

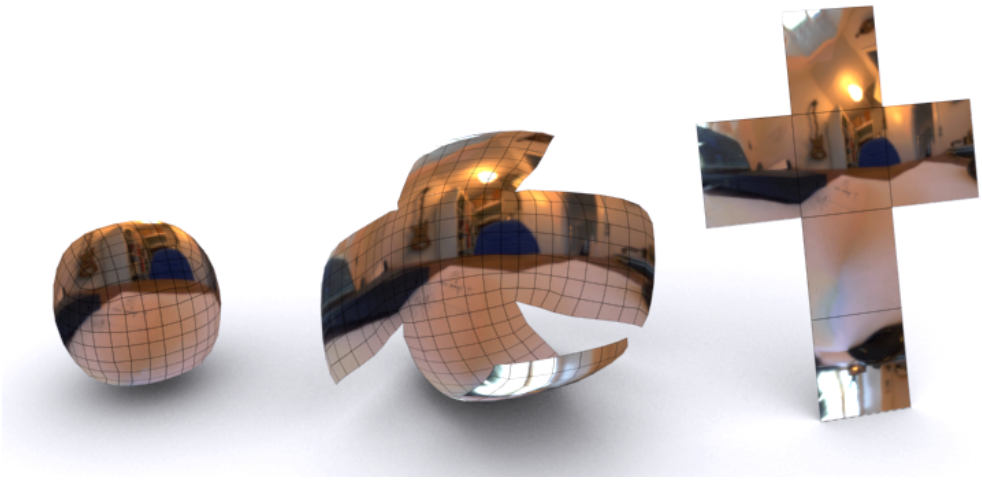


Figure 2.7: The sphere is divided into 6 rectangles, flattened and assembled to a horizontal or vertical cross.

Box or cube environment mapping has got hardware support through both OpenGL[6] and DirectX[7] and thus supported by practically all modern

[6] `http://developer.nvidia.com/object/cube_map_ogl_tutorial.html`

[7] Hardware supported in DirectX 9; `http://msdn.microsoft.com/en-us/library/bb204881(VS.85).aspx`

graphics cards.

2.3 Spatial variation of light

Regardless of the mapping technique used for applying the environment irradiance map to the scene, the issue regarding spatial invariance is not solved by native IBL.

Numerous approaches has been made to solve this issue with methods most of which have to do with densely sampled light fields [UGY06]. This approach has shown to be impractical for actual implementation and use and even raise questions regarding recording precision and spatial detail, as discussed in [Ves08], why also more sparsely populated light fields has been the subject of some work.

Various approached deal with real-time recording and reconstruction from HDR video [HSK+05] where the movement of the camera alters the captured lighting conditions, or [ML03] which focus their efforts on solving specific subcases such as 2D movement.

Also work has been done in the direction of view dependent environment maps [HSL01], [BKM07]. Such maps provide varying lighting conditions for spatial movement and even stave off the need for thousand of individual samples in the dense light field approach.

As all these approaches provide both advances and limitations in regard to the spatial, natural lighting issue, none of them seems to solve the issue in a practically feasible manner, and thus the quest continues.

It has, however, at a late point in the course of this project come to my attention that [Pro07] describes a fairly equivalent project simply called; Spatial Image Based Lighting (SIBL), and thus solves this problem in much the same fashion as proposed in this report. In order to avoid any confusion it is hereby stated that the [Pro07] project has *not* been the basis of this project and is thus explicitly referenced throughout this report whenever it has been used as a resource.

Chapter 3

Overview of the GDIBL method

This chapter presents the GDIBL method in terms of main idea, overall algorithms and issues still pending to be addressed.

Initially an overview of the GDIBL method is described, including the general idea, a step-by-step walk-through and a presentation of the main algorithm.

Subsequently I present a list of known limitations for the method along with possible solutions or arguments for disregarding these issues.

3.1 Overview

This is the section that presents the method derived and described in the first report. Pictures, illustrations, fancy words and possible application and their main concerns for a method like this.

3.1.1 The GDIBL method

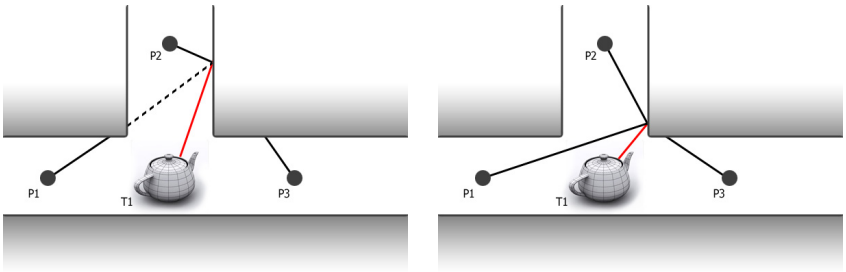
As stated in detail in [Ves08], the preliminary research project for this method proposal, spatial variation under complex lighting conditions is still an issue that challenge both the motion picture industry and the gaming industry alike. While IBL provides a solid solution to sampling and reproducing complex lighting conditions, it natively lacks the ability to provide information about spatial variance in scene lighting.

The GDIBL method introduce a combination of image based lighting and a geometric reference model to reproduce spatially varying lighting conditions in a scene. The way this is accomplished is by validating an arbitrary number of light probes relatively positioned within the scene and deriving the irradiance at any novel location by interpolation from the probe data. In the proposed method, a rough geometric model is used to reference the distance from which the the probe lighting is emitted, but any depth information could be used. Whether geometric reference, depth

information embedded in the light probe format or any other method is utilized, the depth information is used to query valid light probes and reference the direction in which irradiation data is requested.

As the light probes are mapped to a geometric model, the main light sources in the HDR maps has an actual extent and location relative to the object being rendered and thus provide support for hard and soft shadows in accordance with the scene. This means that light is distributed throughout the scene largely in the same way as in the real world. But this method does not rely on heavily precalculated light distribution as this is provided by simple derivation for each point at runtime. As only a few levels of ray-tracing is needed at runtime this method is also plausibly implementable for dynamic real-time applications as well as off-line rendering.

Figure 3.1 illustrates the reference method of the GDIBL method. While gathering the collective irradiance at the rendered point of the tea pot, a geometric model is being used to validate and reference light probe data from probes $P1$, $P2$ and $P3$ respectively.



(a) Both $P1$ and $P3$ are blocked in their line of sight and invalid.

(b) In this case both $P1$ and $P2$ hold valid information about the queried direction.

Figure 3.1: The point referenced in 3.1(a) return data from $P2$ only, while the point referenced in 3.1(b) should be a blend of data from $P1$ and $P2$.

In 3.1(b) both $P1$ and $P2$ has a line of sight (LOS) to the to the environment reference point. This means that irradiance information for the positioned direction, indicated by the red line, should be the blended irradiance at the positions of light probes $P1$ and $P2$ coming from that particular point:

$$E(x) = \int_{\Omega 2\pi} \sum_{i=0}^{n_valid} E(p_i, \vec{\omega}_i) k_b lend \tag{3.1}$$

Where p_i are positions of valid light probes and x_e is the intersection point with the environment geometry by the query direction vector. Further more, the irradiance function has been extended to look up irradiance from one particular direction, and presents as such a mirrored alternative to $L(x_e, \vec{\omega}_i)$; the radiance at surface location x_e in the positioned direction from that point towards p_i. $k_b lend$ is a generic blending coefficient determined by the applied blending method, see section 3.1.2.

Step by step

The GDIBL method is a generic method for querying lighting information from multiple light probes spatially positioned within a given scene. Even though distributed ray tracing, real-time rendering or a number of other frameworks could be used, the method is as proposed based on some level of ray tracing. The method itself only handle the actual irradiance querying of which the flow can be described as follows:

The validation of light probe consists of an additional ray intersection test, namely the environment geometry intersection of a ray from the light probe position to the intersection point of the initial hemisphere ray. If the light probe is not blocked in its line to the intersection point, it is said to have a LOS and is thereby considered valid.

Algorithm 1 describes the process of deriving lighting information for a single point. Since the algorithm only returns the irradiance on a single point, it is equally valid for full off-line ray-tracing and for simplified shader-based real-time rendering, though the need for recursive ray-tracing should be handled differently. See Figure 3.2 for illustrated walk-through.

Input: Set of light probes ($\mathcal{E}$), scene geometry (G_{scene}), environment reference geometry (G_{env}), target point on object surface (P_{target})
Output: Irradiance value (I)for the target input point

```
1  Create set of rays R_hemi to cover hemisphere from target point
   P_target;
2  foreach ray in R_hemi do
3      foreach geo in G_scene do
4          if r intersects with geo then
5              Trace recursively or return final value;
6              ...                          /* framework dependent */
7          end
8      end
9      foreach env in G_env do
10         if ray intersects with env then
11             P_int ← intersectionpoint;
12             E_valid ← hasLineOfSight(E, P_int);
13             if E_valid is empty then
14                 return value of nearest/last successful neighbor;
15             else
16                 return interpolation between all probes of E_valid;
17             end
18         end
19     end
20     if ray did not intersect G_scene or G_env then
21         return interpolated value of probes with LOS in the given
           direction;
22     end
23 end
```

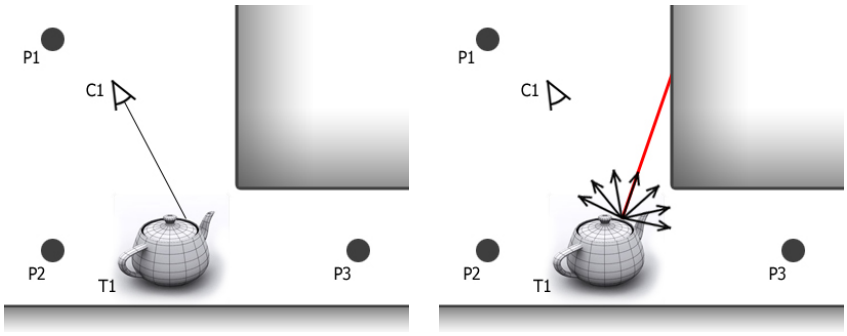
Algorithm 1: Pseudo code algorithm for the GDIBL method.

The reason that only intersection with environment geometry should entail the disqualification of a light probe is that this method queries the scene as is, while leaving further light interaction within the scene to the surrounding framework.

3.1.2 Blending

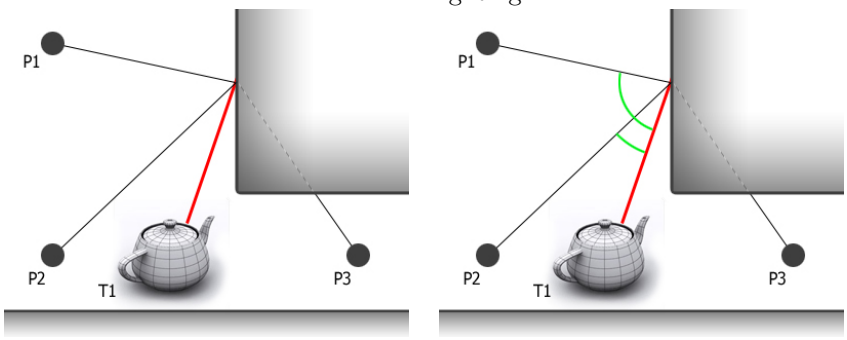
The GDIBL method is based on the derivation of information from multiple light probes. As these light probes does not cover the entire scene, interpolation between the accessible samples is needed in order to derive plausible lighting information for novice positions.

The light probe samples used in the GDIBL method are arbitrarily positioned to improve flexibility in acquisition and setup. Interpolation of



(a) A piece of geometry is seen by the camera.

(b) Rays are sent out in a number of directions to inspect the environment lighting.



(c) Intersection with the environment geometry of GDIBL is used to validate existing light probes.

(d) Valid light probes are blended according to angle. The one to closest approximate the viewing angle is weighted highest.

Figure 3.2: Illustration of the Algorithm 1 in work.

light probe data must therefore accommodate such arbitrary positioning and cannot rely on grid based or in other ways uniformly distributed data.

inverse distance weighted (IDW) interpolation is such an interpolation method. IDW adds importance to samples closer to the interpolated point than samples far away. The interpolation carried out in Algorithm 1 line 16, returns a novel irradiation value based on all light probes in the scene considered valid for the reference point in question. While a HDR light sample is a recording of irradiation from all directions at the location of the light probe, this is also a recording of radiance from all visible surfaces towards this one point.

The IDW interpolation is defined as

$$\acute{r} = \sum_{i=0}^{n} \frac{idw(p_i, x_e)}{\sum_{j=0}^{n} idw(p_j, x_e)} r_i \tag{3.2}$$

where $\acute{r}$ is the novel light ray, r_i is the light ray sample and $idw(p_i, x_e)$ is the IDW function of $\acute{r}$. As defined the weight is in fact the distance measure of the individual sample in relation to the collective distance measures of all samples being considered. This distance measure can be based on any

property that applies to all samples.

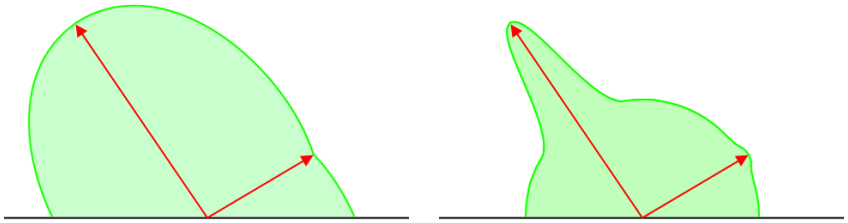
Remembering equation 2.4 and the theory on specular properties in most real world surfaces, we know that radiance from such surfaces vary depending on the viewing angle. In this case the location of our light probes. This leads to conclude that the angle from the reference point to the query origin is a more significant weight base for this application than the euclidian distance between the probe and the query origin (qo).

Utilizing angular distance with the IDW interpolation scheme leaves us with this function for a novel light ray;

$$\acute{r} = \sum_{i=0}^{n} \frac{ang(\vec{p}_i, \vec{x}_e)}{\sum_{j=0}^{n} ang(\vec{p}_j, \vec{x}_e)} r_i \tag{3.3}$$

where $ang(\vec{p}_i, \vec{x}_e)$ is the angular distance between the vector $\vec{p}_i$ from probe (p_i) to the environment intersection point (x_e) and the vector $\vec{x}_e$ from the QO to the intersection point.

Using this sort of interpolation a special case will have to be made only when there are no valid samples or when all valid samples are in the exact same direction as the QO.



(a) The interpolation result for equation 3.3

(b) Interpolation result for equation 3.4 using degrees and an exponent of `k=0.3`

Figure 3.3: The distinct difference between the overshooting classic IDW and the simplified exponent based IDW function.

An even more simple interpolation can be defined as

$$\acute{r} = \frac{1}{ang(\vec{p}_i, \vec{x}_e)^k} r_i \tag{3.4}$$

where k is the angular distance exponent. This interpolation is still considered an IDW interpolation, but offers a "handle" to adjust the bias of significance assigned to the participating light samples.

Using angular distance rather than euclidian distance embrace this specular reflection variation over a surface as it gives the most significance to samples that are angle wise closest to the QO.

The IDW interpolation presented in equation 3.3 always produce an interpolated value that is higher than the nearest neighbor and equation 3.4 can be presented with an exponent that displays this same property. However, none of these interpolation functions are practical in the case where an interpolated value should be based on more than two samples. In this case both functions will overshoot the intermediate samples by unsatisfactory high values. Therefore a simple elimination has been implemented to only take the two angle wise closest samples into account. See section **??** for further discussion of this choice.

3.2 Known limitations

The GDIBL method is proposed as a solution to spatial variance in complex and sampled lighting conditions without engaging in massive sampling. Even though this issue is accommodated by the method, there are some practical limitations to the utility of GDIBL.

3.2.1 Rendering specular reflections

As the GDIBL method is based on depth corrected IBL, this correction can be done at various levels of detail. The level suggested through this work does not provide sufficient detail to use the HDR lighting information for specular reflections. This is due to the fact that the reflected environment would be projected onto an approximated model, usually with far too little detail. Especially motion renderings will reveal such insufficient level of detail.

The GDIBL method thrives on referencing *simple* geometric environment models. In order to preserve the light weight idea of the method, possible solutions for this limitation could therefor lie in either storing additional depth information in parallax mapping or similar. While this may introduce a high detail depth reference through intersecting simple geometric models, it could leave the acquisition of real world depth data to technologies like Z-Cam or similar depth acquiring devices while depth maps can be rendered from any 3D production tool on the market.

Lastly it should be mentioned that in simple, synthetic cases where the geometric model is identical with the actual scene geometry, synthetic objects with specular surfaces should not trigger this issue.

3.2.2 Highly specular environment

While issues remain regarding the rendering of synthetic objects with specular reflection properties, specular reflections also present a challenge in

the environment scene. While light may fall through a window and be perceived as directional, we still have the option of modeling the window as a hole in a wall and leaving the sun in the distant environment outside. This utilize the advantage of the referential environment geometry and thus pose no problem. If, on the other hand a mirror reflects the sun's rays in any arbitrary direction through the room, this present a very limited beam from a perfectly plane surface.

This issue also include directional light emission from a modeled surface such as a screened lamp.

The issue with highly specular or directionally light emissive surfaces in the sampled environment is outside the scope of this project, but can be solved by denser sampling around that particular light beam. In this case an alternative interpolation method, to distinctively honor higher resolution in changing lighting conditions, must be considered.

3.2.3 Semi-transparent environment

A window can be modeled simply as a hole in the wall to accommodate the property of glass allowing the light to pass through it. In the case of a night time scene where it is dark out side, the glass is suddenly primarily a specularly reflecting surface and modeling the window as a hole will simply not suffice. While the night time window might reflect any light from the inside, it still allows the light of a distant street light to enter the indoor scene. This combination of transparent *and* reflective properties present a delicate issue that may only increase with even more demanding properties.

While steamy windows and translucent curtains are interesting materials to study, the issues of embracing such advanced properties currently lies outside the scope of the GDIBL method.

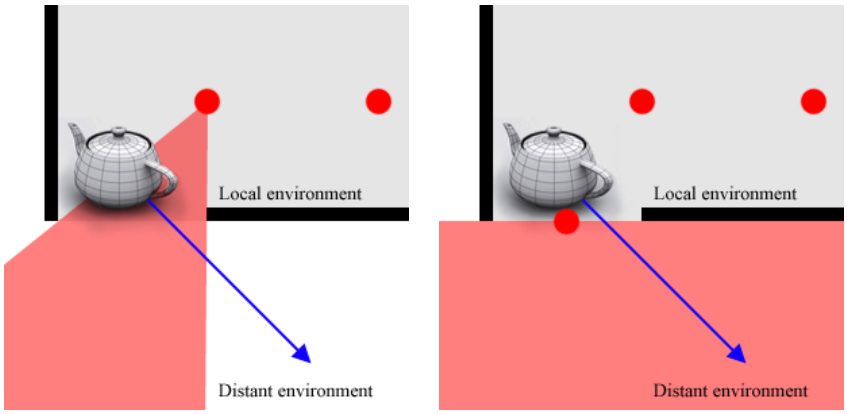
3.2.4 Distant environment

In addition to the previously known issues regarding modeling and rendering scenes lit by the GDIBL method, new issues have become apparent through the implementation work with this method.

The illuminating environment is divided into two categories in the GDIBL method; a local environment that is geometrically represented for reference and a distant environment that shares the infinite distance property of traditional IBL.

As light probes scattered around the local scene may be referenced by the local environment geometry, this geometry also define a limit for the information derivable from any one light probe. Such limitation can lead to environment shadows; areas for which no information is stored. In Figure 3.4 a scene is well provided for regarding internal light information. However, the gap in the local environment geometry provides a portal into the distant environment. While staying behind the outmost light probe, this distant environment is easily referenced simply by direction of the

closest light probe with a LOS in the given direction, but when a query is made into an unmapped area, no lighting information can be found, as seen in Figure 3.4(a). In such case, Algorithm 1 suggest to use the last or closest successfully queried direction, but a simple, more correct consideration can be made.



(a) Unmapped distant environment is referenced.

(b) Avoiding shadows by extreme placement of light probes.

Figure 3.4: Avoiding shadow areas in distant environment

As illustrated in Figure 3.4(b), this issue can be addressed by placing light samples in the extreme boundaries of or even outside the local scene. These samples, even if nothing else, provide information about the distant environment that is blocked from probes further inside the scene.

As carefully considering light probe placement will diminish shadowing in the distant environment, this issue does not affect the technical implementation of the method.

Chapter 4

Implementation

Now that the method has been presented, this chapter describes the actual implementation.

The intention of this implementation is to verify the effect of the GDIBL method in an environment that provides a basic, plain and in all other aspects simplified framework. The simplicity of the framework is sought in order to make the effects of the actual method as clear and evident as possible. The implementation presented through this report is an effort to demonstrate the validity and results of the GDIBL method thus far presented in pure theory.

4.0.5 Plug-in vs. stand-alone

Initially I had to decide whether to build an entire rendering system from scratch or write a plug-in for an existing renderer or other existing 3D tool. There were several reasons for choosing to build a rendering system myself.

With reference to the personal goals listed in section 1.2 building a self contained system would grant me insight in the entire process and the various considerations that can affect both process and outcome. Also given the task of proofing the validity of the GDIBL method, I had to have a way of following the rendering process and confirm that all values are correctly derived in respect to the GDIBL method.

In the following, the implementation is described in terms of relevance to the method and technical solutions to practical problems.

4.1 Basic raytracer

The ray tracer is a common term for a rendering system based on scattering rays into the scene and tracing their paths to find and shade the objects they intersect. A ray tracer provides a simple three dimensional coordinate system, a camera and various objects which can be rendered to an image and displayed on screen and/or saved to disk. This basis is described in the following section with respect to its supportive features and frame work properties.

As a basis for the GDIBL method, a ray tracer provides a more straight forward approach to IBL as the sampling of environment light contributions are intuitively done using rays. The 2002 IBL tutorial by Paul Debevec [Deb02] also illustrates the recursive use of rays to investigate the scene and surrounding environment.

In this section the foundational ray tracer is described to establish the framework for the GDIBL extension described in section 4.2.

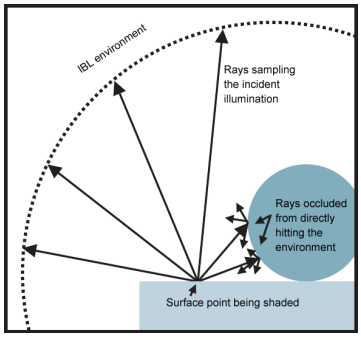


Figure 4.1: The recursive approach of the Debevec tutorial.

4.1.1 Coordinate system

A three dimensional coordinate system, consisting of an X-, a Y- and a Z-axis was defined and implemented through the `Vector` class as sets of three `double` values. Based on the reasons mentioned in section **??**, I chose to implement this class myself instead of utilizing e.g. the System.Direct3D.Vector class.

The world coordinate system (WCS) is used throughout the application, and local coordinate transformations has therefore not been required. The WCS is a right hand coordinate system, meaning that the internal orientation of the X-, Y- and Z-axis matches the schematic in Figure 4.2. In the abstract world of this implementation the Z-axis points up while the X- and Y-axis defines the horizontal plane. Up, down, right and left are of course only abstractions but does support description of scene configurations later on in this report .

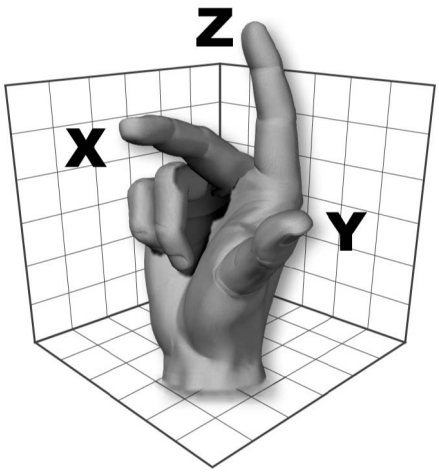


Figure 4.2: Right hand coordinate system

4.1.2 Camera

The orientation, position and ray casting of the camera are likewise based on the WCS. This means that whenever rays are generated from the camera, they are natively represented in world coordinates.

General camera class is inherited by `classicCam` and `angleCam`, see class diagram in Figure 4.7. The `getRay()` method of the camera-class provides each new query ray from the camera into the scene.

Figure 4.3 illustrates how a ray is fired from the camera position through an abstract image plane and thus queries the scene one pixel at the time.

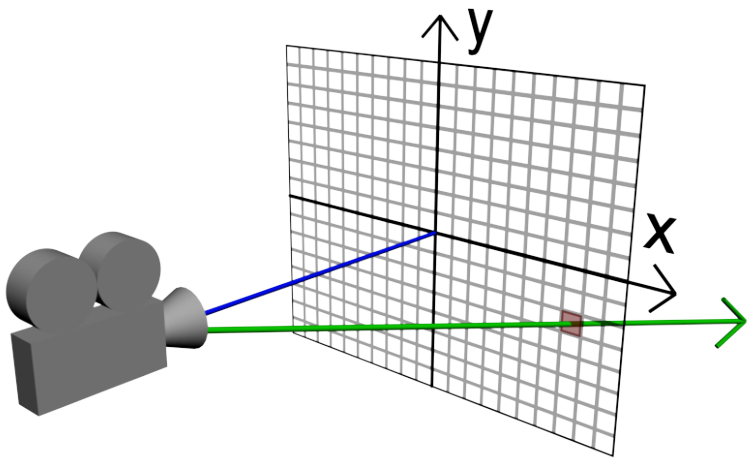


Figure 4.3: The implemented camera trace rays through the image plane.

```
1  public Ray GetRay(int x, int y){
2      // Distance to the relative right
3      Vector dRight = Right() * HorizontalUnit * x;
4      // Distance to the relative up
5      Vector dUp = Up * VerticalUnit * y;
6      // Distance in the relative direction
7      Vector dDir = FocalLength * Direction;
8      // Assemble to target direction
9      Vector dir = dRight + dUp + dDir;
10
11     return new Ray(this.Position, dir);
12 }
```

Listing 4.1: Generate a direction vector from (x,y) screen space coordinates.

In implementation, the `getRay()` constructs a new direction vector through multiplication of custom unit-vectors relative to the orientation of the camera. As seen in listing 4.1, this direction vector is combined with the position of the camera itself to produce the requested ray.

4.1.3 Ray tracing routine

The ray tracing algorithm has been split up into a basic loop including a ray generating method and a recursive methods retrieving pixel colors.

`getCol(Ray)` handles object and environment intersection as well as shading according to the material of the intersected object.

The class diagram for the basic ray tracer is depicted in Figure 4.4. Only the most basic features are realized at this point in relation to the introduction of the GDIBL extension presented in section 4.2.

The ray tracing routine along with most of the additional methods it needs are included in the `frmNT` class, which is specialization of a the `System.Windows.Forms.Form` and is, as a single file the heart of this im-

```
int imgH = (int)(scene.camera.renderResolution.Height / 2);
int imgW = (int)(scene.camera.renderResolution.Width / 2);

for (int imgY = -imgH; imgY < imgH; imgY++)
{
    for (int imgX = -imgW; imgX < imgW; imgX++)
    {
        int x = imgW + imgX;
        int y = imgH - imgY -1;
        FIRGBF color = Color.Black;

        // Fire ray and do intersection
        Ray ray = scene.camera.GetRay(imgX, imgY);
        color = getCol(ray, DEPTH);

        buffer[x, y] = color;
    }
}
bitmapImg = drawIt(buffer);
```

Listing 4.2: Traditional basic ray tracing algorithm

plementation.

4.2 GDIBL extension

The GI implementation in this particular application of the GDIBL method is based on recursive light sampling rather than photon mapping, irradiance mapping or any other multi pass algorithm. This again does not speak to the efficiency of the implementation but does apply GI on a practical, hands-on and uncomplicated basis.

In order to provide the final shader with the light information it needs, a few support features must be established beforehand.

4.2.1 HDR pipeline

Image based lighting is based on the high dynamic range of light information stored in HDR images. To implement a pipeline that embrace such an extended range of light values one need only to store color values in a 16 or 32 bit per channel floating point color format. However in order to read the HDR images from which the lighting is to be derived, the `FreeImage` library [Fre08] was introduced quite early in the process.

In addition to the traditional LDR formats, the `FreeImage` library provides read/write functionality for most HDR file formats, such as Debevec's .hdr, the OpenEXR format of Industrial Light & Magic and the 48-bpc TIFF format. It also provides suitable color formats capable of handling up to 48 bit per color channel as well as creation of custom channels for e.g. depth, object ID or what ever might be needed.

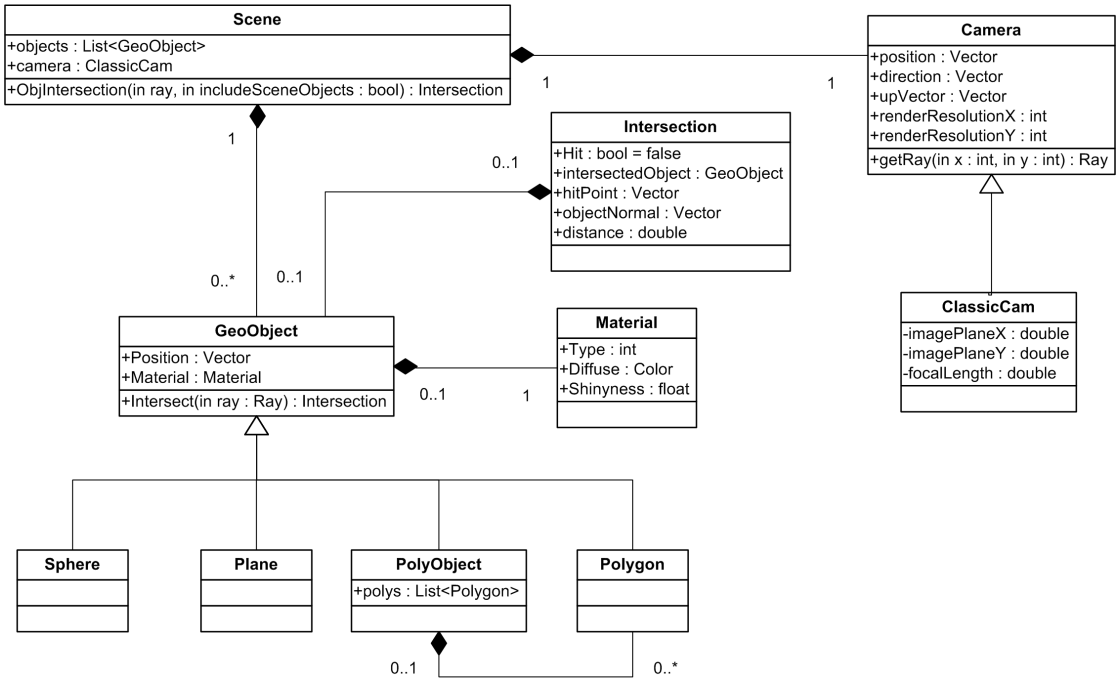


Figure 4.4: A class diagram for the initial ray tracer.

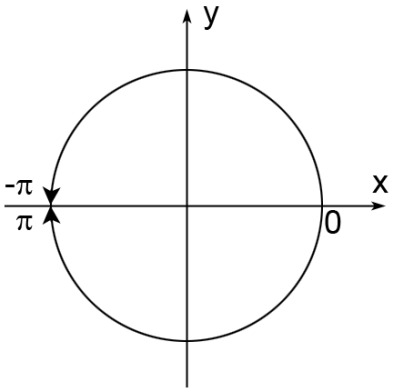
The FreeImage RGB floating point ($FIRGBF$) color type chosen for this implementation since it presents an RGB based floating point format with no additional channels and features. In other words; it was the most simple format that met the requirements.

Like all color types provided by the `FreeImage` library, the $FIRGBF$ type has implicit conversion from the System.Drawing.Color type and, according to the documentation, also from $FIRGBF$ back to System.Drawing.Color. This implicit conversion, along with it's explicit alternative, does not include tone mapping of any sort, and thus cannot handle color channel values that exceed the LDR roof of 1 (255 in 8-bit integer based formats). Even though explicit tone mapping is made available through the `FreeImage` library, this has not been utilized in the implementation of GDIBL method. Instead a simple clamping method makes sure that no values are either above 1 or below 0.

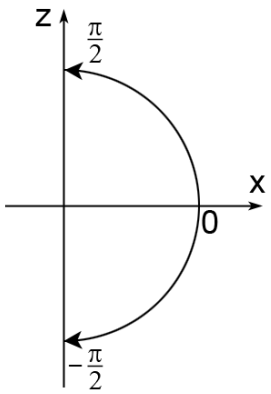
4.2.2 HDR environment map

The HDR file format chosen for this implementation is .hdr. In addition to choosing a file-format I chose to use latitude-longitude maps as the spherical coordinates could easily be converted to (x, y, z) based vectors and vice versa.

The coordinate mapping of spherical maps is illustrated in Figure 4.2.2. The spherical coordinates $(\rho, theta, \phi)$are derived from an (x, y, z) vector as follows;



(a) θ going from $-\pi$ to π is mapped to 0 and *image width* respectively.



(b) ϕ going from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ is mapped to 0 and *image height* respectively.

$$\rho = \sqrt{x^2 + y^2 + z^2} \tag{4.1}$$
$$\theta = \arctan \frac{y}{x} \tag{4.2}$$
$$\phi = \arctan \frac{\sqrt{x^2+y^2}}{z} \tag{4.3}$$

and (x, y, z) from spherical (ρ, θ, ϕ) like this;

$$x = \rho * \sin\theta * \sin\phi \tag{4.4}$$
$$y = \rho * \sin\theta * \cos\phi \tag{4.5}$$
$$z = \rho * \cos\theta \tag{4.6}$$

These convenient conversions along with the humanly readable visual representation of the spherical map, were the main reasons behind this choice.

4.2.3 Ray distribution

In order to sample the actual hemisphere over a surface point, we must integrate the incoming radiance over the hemisphere. The common approximation to integral functions in both physics and computer graphics are variations of the Monte Carlo method, equation **??**. This approximates the integral function by summing up random samples from the internal function and calculates a mean value. This obviously means that the more samples one calculate, the better the final approximation will be.

$$\int_{\bar{I}^s} f(u)du \approx \frac{1}{N} \sum_{i=0}^{N} f(x_i) \tag{4.7}$$

where $\bar{I}^s$ is the s-dimensional unit cube, $\bar{I}^s = [0,1] \times ... \times [0,1]$ and each xi is a vector of s elements. In this case three. The two common variations are the classic Monte Carlo method where the set $x1, ..., xN$ is a subsequence of pseudo-random numbers, and the quasi-Monte Carlo method where the set is a subsequence of a low-discrepancy sequence.

Using ray distribution means entering a game of chance in regard to sampling the significant light sources in the probe data. The chance for randomly distributed rays to hit all significant light sources in an image can be very small as massive light emission is rarely seen from large areas of the image plane.

Methods of calculating a diffuse map of hemisphere radiance have proved to be highly efficient for diffuse lighting as only the surface normal is needed to pinpoint the collective irradiance over the entire hemisphere. However, this method only works for convex objects as there are no native occlusion detection involved.
Under the general term *importance sampling*, numerous methods have been developed to sample, narrow down and target significant light sources in images for use in IBL. The purpose of such methods is to provide a more of the significant lighting information while using fewer sample rays. With reference to section **??** I chose to refrain from the introduction of such methods, but do recommend them for any future IBL implementation.

Using the Monte Carlo method with a low-discrepancy sequence of samples provide a uniform distribution over the hemisphere. This will produce a less noisy result than using random sample directions, but it will need the same high amount of sample rays to produce an acceptable result. The reason for this requirement is that a uniform distribution also means using the *same* distribution over a plane surface. This means that if the distributed rays miss a significant light, this light will be missing from all sample sets based on that hemisphere direction, i.e. a plane surface will be incorrectly rendered in it's entirety.

Random distribution

Alternatively random distribution sends out rays randomly scattered over the hemisphere. These rays are likewise used to sample the irradiation from the given direction and are conclusively aggregated to derive the collective irradiation in the emission point. But as these rays does not abide to any restrictive rules of distribution other than to stay within the given hemisphere, the dice deciding whether they hit a significant light is rerolled at every sample point. This renders the surface more vibrant due to noise in the random sample results, but it also heighten the probability that the general impression of the surface is coherent with the environment from which the irradiance is derived. And since this particular implementation was written in C#, it was therefor reasonable to utilize the random number generator native to this language, and thus base the sampling on the

pseudo-random Monte Carlo method.

As stated in [Jen01]; Using the relationship in equation 2.9 the diffuse radiance, rho_d can be found to be

$$\rho_d = \frac{d\Phi_r(x)}{d\Phi_i(x)} = \frac{L_r(x)dA\int_\Omega d\vec{\omega}}{E_i(x)dA} = \pi f_{r,d}(x) \tag{4.8}$$

for a Lambertian surface. This is true since $\int_\Omega d\vec{\omega}_d = \pi$.

The reflected direction is, as mentioned in section 2.1.3, perfectly random for a Lambertian surface. Thus given two uniformly distributed random numbers $\xi_1 \in [0,1]$ and $\xi_2 \in [0,1]$ we find that this randomly reflected direction, $\vec{\omega}_d$ is distributed as

$$\vec{\omega}_d = (\theta, \phi) = (\cos^{-1}(\sqrt{\xi_1}), 2\pi\xi_2) \tag{4.9}$$

here presented in spherical coordinates (θ, ϕ) for the direction; θ is the angle from the surface normal and ϕ is the rotation around this normal from the x-axis.

This method has been used for creating a set of random directions which, when coupled with the given point on the surface, presents a set of rays randomly scattered over the hemisphere, as illustrated in Figure 4.5.

The $\sqrt{}$ factor in equation 4.9 bias the θ values to produce angles closer to the normal, see Figures 4.5(a) and 4.5(c). In order to evenly distribute the ray directions over the entire hemisphere, the $\sqrt{}$ factor has been removed from the construction to produce the results seen in Figures 4.5(b) and 4.5(d).

```
1  private void populate()
2  {
3      Dirs = new List<Vector>();
4      for (int i = 0; i < amount; i++)
5      {
6          double rnd1 = rnd.NextDouble();
7          double rnd2 = rnd.NextDouble();
8          double theta = Math.Acos(rnd1);
9          // for normal biased (even 2D) distribution, use this
10         //double theta = Math.Acos(Math.Sqrt(rnd1));
11         double phi = 2 * Math.PI * rnd2;
12         Vector v = Vector.FromAngles(theta, phi);
13         Dirs.Add(v);
14     }
15 }
```

Listing 4.3: The actual implementation for populating a list of random direction vectors.

Listing 4.3 is the actual C# implementation of the random direction generation in the GDIBL implementation.

In the `getCol(Ray ray, int step)` method that handles the actual object intersection and shading, the getRaysBy(Vector pos, Ray normal)

returns a set of rays based on the position and a re-populated set of random direction vectors. These rays are then tested for intersection with all objects in the scene or only environment geometry if the rays are distributed from an environment surface, see section 3.1.2.

4.2.4 Shading

For the actual shading of the queried pixel I chose to multiply the diffuse color of the intersected object with the derived irradiation per color channel. This provides a simple and transparent shading algorithm that makes it practicable to deduce the incident radiation by simple division. This also supports the theory regarding the light absorbtion and transport, since the reflected light can never become more powerful than the incident.

```
1  private FIRGBF getCol(Ray ray, int steps){
2      Intersection intersInfo = scene.ObjIntersection(ray, true);
3      FIRGBF color = Color.Black;
4      if (! intersInfo .Hit || intersInfo .Env)
5      {
6          color = blend(ray, intersInfo , BLEND_MODE_INVERSE_DISTANCE);
7      }else{
8          if ( intersInfo .Hit && !intersInfo.ObjectNormal.IsBackFace(ray.dir) &&
             steps > 0)
9          {
10             Material mat = intersInfo.IntersectedObject.Material;
11             int mode = mat.Type;
12
13             switch (mode)
14             {
15
16                 ...
17
18                 case Material.MODE_IBL;
19
20                   if (!mat.shininess.Equals(1F))
21                   {
22                       // rayDist = new RayDistribution(IBL_COUNT);
23                       Ray[] dist = rayDist.GetRaysBy(new Ray(intersInfo.HitPoint,
                           intersInfo.ObjectNormal));
24                       FIRGBF[] buffer = new FIRGBF[dist.Length];
25
26                       int rayCount = 0;
27                       foreach (Ray dRay in dist)
28                       {
29                           Intersection dRayHit = scene.ObjIntersection(dRay,true);
30                           if (dRayHit.Hit)
31                               buffer [rayCount++] = new FIRGBF(Color.Black);
32                           else
33                               buffer [rayCount++] = getCol(dRay, steps);
34                       }
35                       color = colMultiply(intersInfo .IntersectedObject.Material.
                           diffuse , middle(buffer));
36                   }
```

```
37                  if (mat.shininess > 0)
38                  {
39                      Vector Refl = ray.dir + 2 * intersInfo.ObjectNormal * -
                          intersInfo.ObjectNormal.Dot(ray.dir);
40
41                      // Take it .1 normal out from the surface, so it won't hit it
                          again :)
42                      Vector point = intersInfo.HitPoint + (intersInfo.
                          ObjectNormal * .1);
43                      FIRGBF refCol = getCol(new Ray(point, Refl), --steps);
44
45                      color.red += refCol.red * mat.shininess;
46                      color.green += refCol.green * mat.shininess;
47                      color.blue += refCol.blue * mat.shininess;
48                  }
49                  break;
50
51                  ...
52
53              }
54          }
55      }
56      return color;
57  }
```

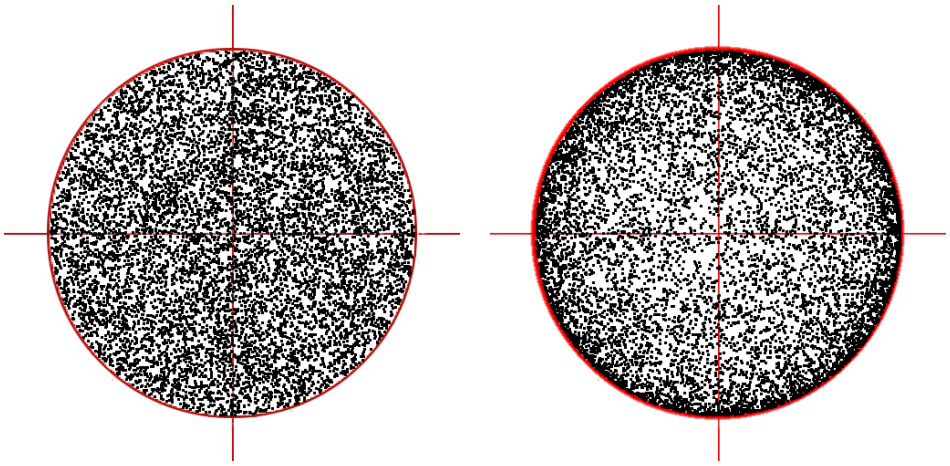
Listing 4.4: `getCol` returns the color of the shaded point based on it's material type and chosen light probe blending mode.

In implementation the `intersInfo` object is an Intersection instance based on the ray for which shading is required. That is, if the ray is an initial one it shoots from the camera position into the scene, as described in section 4.1.2. The `intersInfo` object of class Intersection holds information about whether a scene or environment geometric object has been intersected, and in such case the normal and material of the intersected surface and at which distance from the ray origin the intersection occurred.

As indicated in the introduction of section 4.1, the rendering system is based on recursive sequencing of ray distribution and ray tracing.

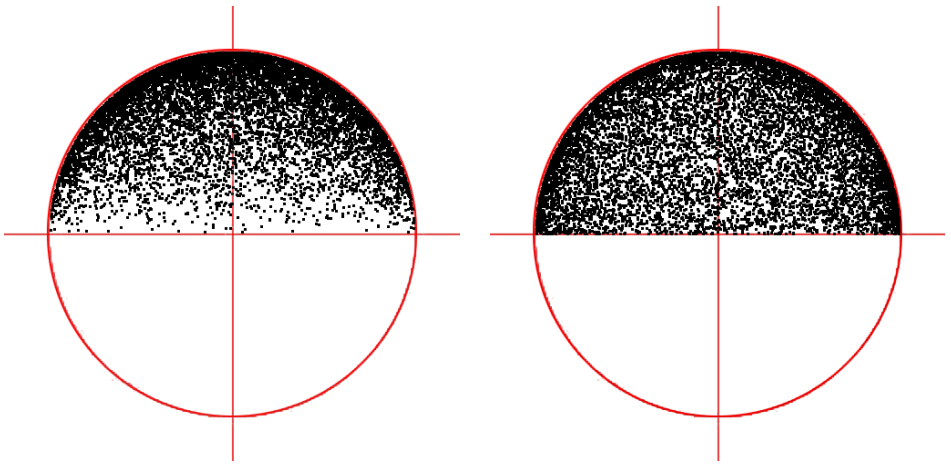
Figure 4.7 shows the most significant additional classes and methods needed for applying the GDIBL method to the basic ray tracer.

Adding the HDR pipeline, the HDR environment maps, recursive ray distribution and IBL shading, the system was now ready to take on the task of rendering synthetic objects with high complexity lighting conditions derived from a sparse number of light probes and a simple geometric model of the scene.



(a) Top view of a normal biased distribution.

(b) Top view of an unbiased distribution.



(c) Side view of a normal biased distribution.

(d) Side view of an unbiased distribution.

Figure 4.5: 2D views of a biased and an unbiased random ray distribution over a hemisphere. All distributions are based on 10000 rays.

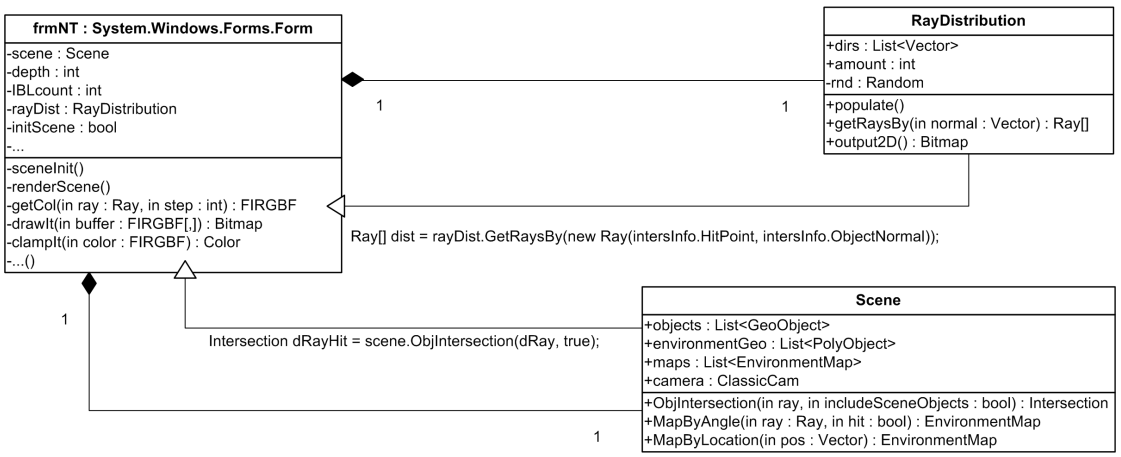


Figure 4.6: The `RayDistribution` object is queries for a new set of random rays for a given hemisphere. These are then used to test for intersection with any scene objects.

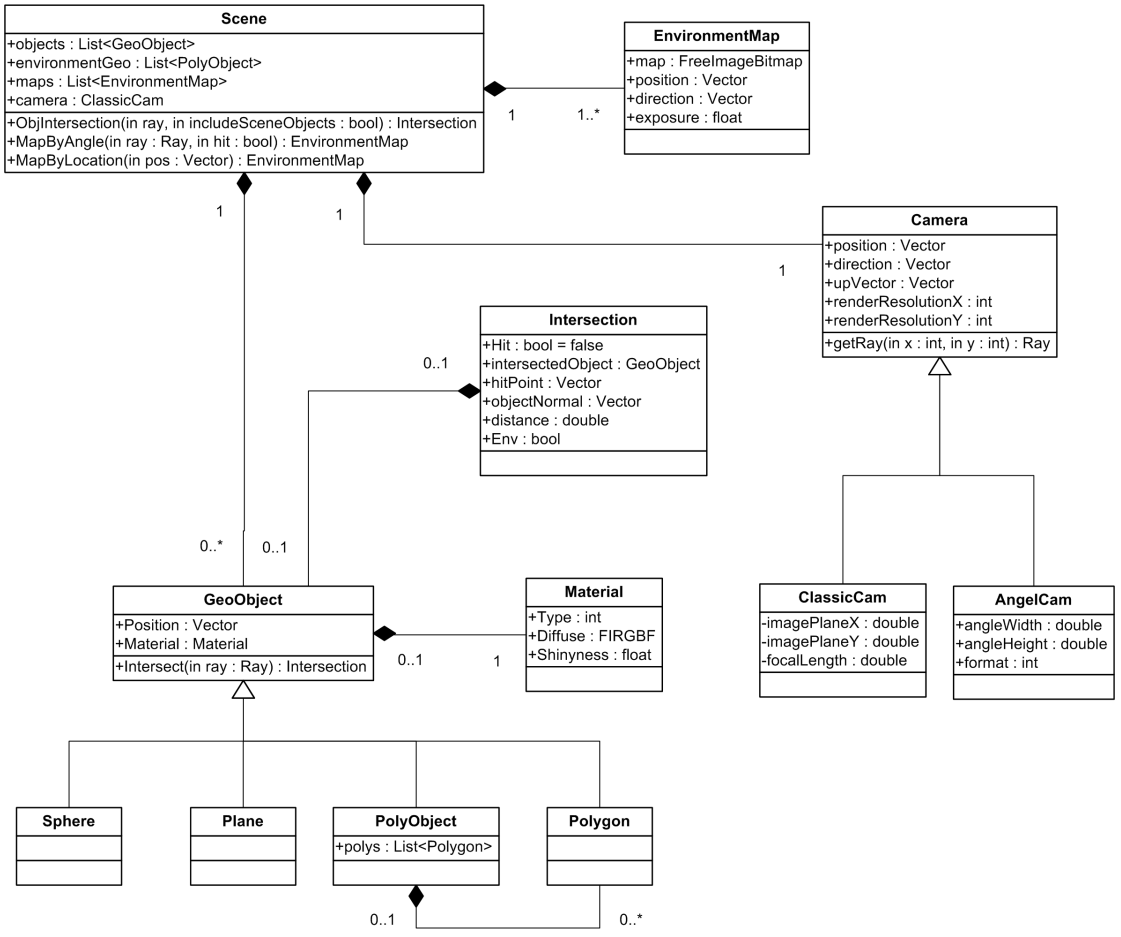


Figure 4.7: A class diagram including the GDIBL extension.

Chapter 5

Results

Testing has been a common task through the work associated with this project. Various features were added to support feature testing, such as the graphical output of the ray distribution, the `AngleCam` for testing environment mapping and multiple logging routines to provide feedback and tracking rays, colors and intersection tests.

This section focus on the validation test of the GDIBL method and the results of my implementation.

Initially a suitable test scenario is established, illustrated and described in terms of the possible and the expected test result. In order to validate the method, the results must meed the theoretical expectations.

For this the second section of this chapter describe and analyze the produced results to evaluate the success of the implementation and the validity of the method.

5.1 Test scenarios

In it's current state, the GDIBL should be able to take on real world scenes. Such a task do however require a highly controlled acquisition and calibration process, as described by Jeremy Pronk in [Pro07]. Instead I have produced a synthetic scene and rendered three light probes which are perfectly aligned to the geometry of the scene. In this way, the only thing being tested is the actual method itself.

5.1.1 The setup

The scene environment consists of a box with a hole in each side, which leaves two opposing corners, as seen in Figure 5.1. The opposing corners have been colored red and blue respectively to traditionally demonstrate effective and variant color bleeding. The whole scene has been illuminated by the Uffizi light probe of the Debevec Light Probe Image Gallery[1]. The

[1] http://www.debevec.org/probes/

fact that the gallery buildings shield the sky light on two sides produce the directed inflow of light from the two portals in the box.

The light probes were rendered in HDR to angular maps and transformed to spherical latitude-longitude maps in *HDR Shop*[2] as presented in Figure 5.1.1. The center of each light probes is pointing in the direction of the Y-axis, which means that the left and right edges are -Y while $\frac{1}{4}$ from the left edge is -X, $\frac{1}{4}$ from the right edge is X, the top edge is Z and the lower edge is -Z.

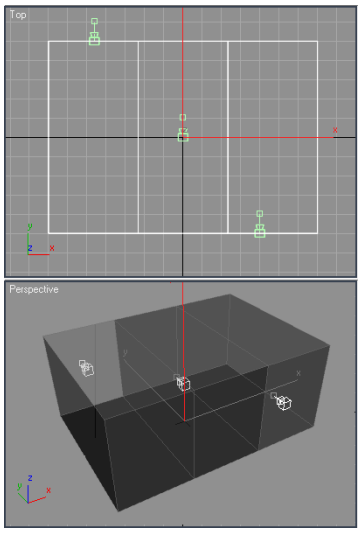


Figure 5.1: Wire frame box, see section 3.2.4.

The local environment geometry is defined by the box model depicted in Figure 5.1. This defines the *local environment* geometry of the test scenario. As the rendered light probes store information about both the inside walls of the box as well as all of the *distant environment* that can be seen from inside the box, see section 3.2.4. The light probes are located exactly in the position from which they were rendered. In this way all calibration and alignment issues are avoided.

For this test three white spheres are placed along the center of the box, from one end to the other. This is done to illustrate the effect of the significantly different lighting conditions they are located in.

5.1.2 Expected result

As it can be seen in the light probe renderings of the Uffizi box scene, the outside illumination is scattered in through both holes, reflected on the colored walls and spread throughout the box. With the GDIBL method we should be able to avoid repeating the heavy GI calculations while at the same time benefit from the incident effect of the pre-rendered scenery. The expected result of the GDIBL method would be to provide the white spheres with location specific lighting from the environment geometry based on the three light probes.

Due to issues regarding tone mapping and color spaces that lie outside the scope of this project, the areas outside the scene and environment geometry are rendered white, which should not influence the visual perception of the rendered objects. As mentioned in section 4.2.1, the resulting rendering is clamped to an LDR as a legacy gesture to the initial ray tracer implementation. However, the un-clamped, full HDR versions are located on the enclosed disk and are also the basis for the following evaluation.

[2]http://projects.ict.usc.edu/graphics/HDRShop/

Shading deviations		
A	**B**	**C**
0.267	0.266	0.437
0.216	0.185	0.241
0.285	0.226	0.261
0.150	0.198	0.582
0.121	0.156	0.477
0.226	0.230	0.527
0.183	0.197	0.251
0.147	0.142	0.157
0.229	0.196	0.186

5.2 Test results

The results presented in this section both support the effect of the GDIBL method as well as the relative correctness[3] of the implementation.

Figures 5.2(a) and 5.2(b) are rendered from the colored, diagonal corners of the scene and sufficiently illustrate both the variation of light throughout the scene but also the methods' native color bleeding phenomenon.

The resulting renderings in Figure 5.2 were produced with a maximum depth of recursion of 3 and with 50 rays per sample distribution. Denser and more concave scenes might need substantially more rays as well as additional depth in order to indirectly illuminate all parts of the scene.

That the three spheres deviate internally in terms of shading is due to the fact that they are presented with various lighting conditions all determined by their location in the scene. This is the

In the setup of Figure 5.2(c) the outer spheres are moved slightly further towards the open portals to further clarify the effect of the spatial light variation.

Remember also, that the scene surrounding the spheres is considered to be and fully handles as environment geometry and is simply having the blended values of in scene light probes projected onto it, while only the spheres are treated with the GDIBL method. This embrace both the comfort of avoiding additional GI calculations including the local environment as well as the de facto production methods using separately acquired high resolution footage of the scene as a backdrop rather than reconstructed scenery from light probes.

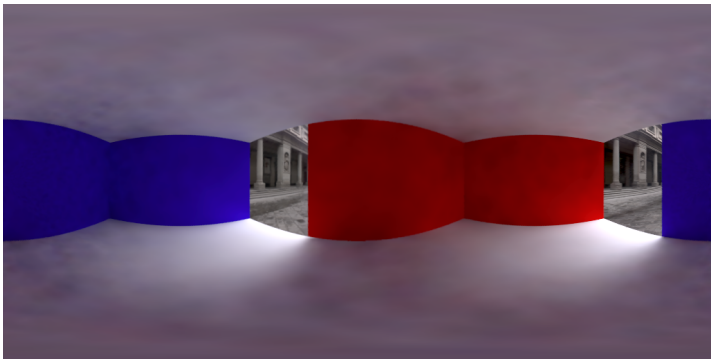
The diversity in shading presented in 5.2 shows us a distinct variation in both color balance and light intensity through the scene and adds to

[3]By relative correctness, is meant an implementation that does the job and correctly implements the mainstay theoretical concepts and mathematical models, but is not necessarily utilizing the ideal data types for storing and treating key values through the process.

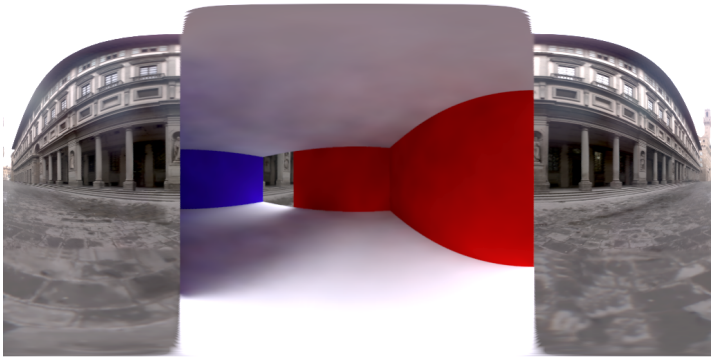
support the validation of the GDIBL method. This table of sampled shading diversity is based on 5 point samples in each geometric extreme of the spheres. Sphere **A** is the one located in the blue corner, **B** is the middle one and **C** is the one located in the red corner.

Through this testing scenario and the results hereof, it has been made clear that the GDIBL method does deliver the promised lighting framework. An effective, image based lighting system that provides spatial variation in lighting conditions through referencing multiple light probes through simple geometric intersection.

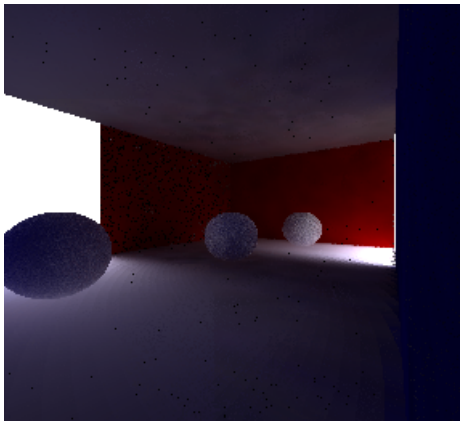
(a) Probe 1 - from the opening at the right.



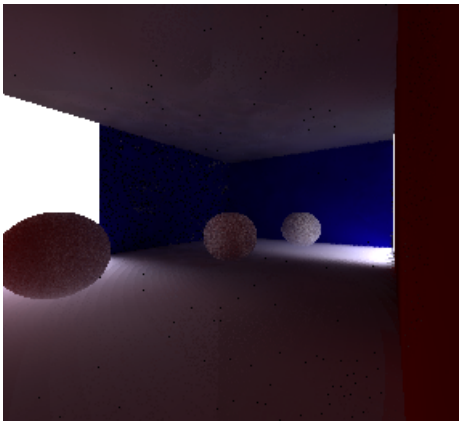
(b) Probe 2 - the center of the box.



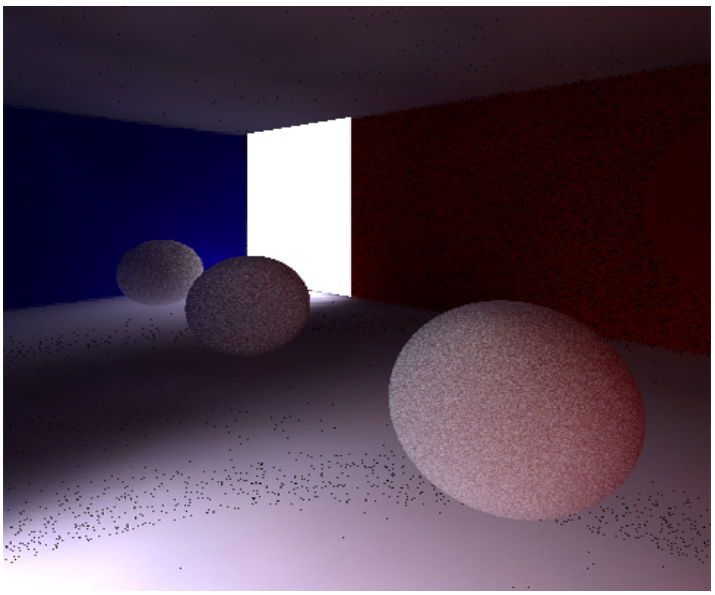
(c) Probe 3 - from the opening at the left.



(a) A view from the blue corner.



(b) A view from the red corner.



(c) A diagonal view from the portal by the red corner.

Figure 5.2: Rendered results of the Uffizi box scenario.

Chapter 6

Conclusion

Ultimately this chapter follows the findings, challenges and results through to a final evaluation of the validity of the proposed method as well touch on the implementation process itself.

6.1 Production evaluation

The process of developing a complete rendering system while practically applying a so far purely theoretical method has been both insightful and challenging. In retrospect the process might have been eased if a more structured picture of the actual application had been produced in advance.

Letting go of some of the control and allow other software packages do what they do best (and fastest) might like wise have been desired, and finding the work of Jeremy Pronk [Pro07] at a late point in the process only supported that notion.

My personal goal of achieving a better understanding of the practical application of computer graphics theory has been fulfilled, and have shown me both distressful and joyous aspects of software production in the process.

6.2 Meeting the expectations

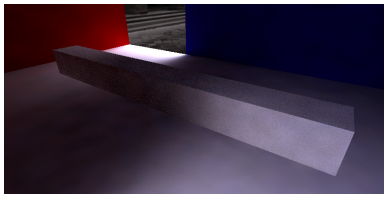
Given the results presented in section 5.2, it has been demonstrated that the proposed method is effective in regard to it's theoretical intention. With a positive result on both spatial lighting variation based on IBL the method stands strong as nothing in the method has interfered with the practical benefits thereof, as cheep environment color bleeding phenomena, etc. It is in other words a more powerful image based lighting method.

6.3 Remaining issues

Regarding the use of the GDIBL method in a production pipe lines, this work provides no guidelines for acquisition, calibration or alignment. It does however provide documentation for a successful implementation and an effective lighting method.

What remains to be addressed are the issues regarding specular surfaces in both the local scene and in the local environment. Semi-transparent materials and other special cases are already out of the scope for this project, but an issue that might require some attention larger scale production, is the interpolation of information from multiple light sources.

When that issue is solved in a fast, seamless manner, this lighting method will be ready for production.



Chapter 7

References and glossary

References

[BKM07] Jens Rosenkjær Andersen Bjarne Kondrup Mortensen. Modeling view-dependent surface point radiance with parameterized maps. Technical report, Aalborg University, 2007.

[CPB06] Luc Claustres, Mathias Paulin, and Yannick Boucher. A wavelet-based framework for acquired radiometric quantity representation and accurate physical rendering. *The Visual Computer*, 22(4):221–237, April 2006.

[Deb98] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 189–198, New York, NY, USA, 1998. ACM.

[Deb02] Paul Debevec. Tutorial; image-based lighting, 2002.

[DM97] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 369–378, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Don06] Weiming Dong. *Advances in Computer Graphics*, volume 4026 of *Lecture Notes in Computer Science*, chapter Rendering Optical Effects Based on Spectra Representation in Complex Scenes, pages 719–726. Springer Berlin / Heidelberg, 2006.

[Fre08] The freeimage project. online resource, August 2008.

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

[HSK+05] Vlastimil Havran, Miloslaw Smyk, Gfzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Importance Sampling for Video Environment Maps. In Kavita Bala and Philip Dutré, editors, *Eurographics Symposium on Rendering 2005*, pages 31–42,311, Konstanz, Germany, 2005. ACM SIGGRAPH.

[HSL01] Ziyad S. Hakura, John M. Snyder, and Jerome E. Lengyel. Parameterized environment maps. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 203–208, New York, NY, USA, 2001. ACM.

[Jen01] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A.K. Peters, July 2001. 2nd printing with corrections March 2005.

[KB04] Juho Kannala and Sami Brandt. A generic camera calibration method for fish-eye lenses. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1*, pages 10–13, 2004.

[ML03] Alexandre Meyer and Céline Loscos. Real-time reflection on moving vehicles in urban environments. In *VRST '03: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 32–40, New York, NY, USA, 2003. ACM.

[Pro07] Jeremy Pronk. Spatial image based lighting. In *SIGGRAPH '07: ACM SIGGRAPH 2007 posters*, page 168, New York, NY, USA, 2007. ACM.

[SFDC01] Yinlong Sun, F. David Fracchia, Mark S. Drew, and Thomas W. Calvert. A spectrally based framework for realistic image synthesis. *The Visual Computer*, 17(7):429–444, 2001.

[UGY06] Jonas Unger, Stefan Gustavson, and Anders Ynnerman. Densely sampled light probe sequences for spatially variant image based lighting. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 341–347, New York, NY, USA, 2006. ACM.

[Ves08] Kasper Vestergaard. Geometrically derived image based lighting. preliminary research, Aalborg University, April 2008.

[Wu08] Ying Wu. Radiometry, brdf and photometric stereo. online lecture notes, September 2008.

Glossary

FIRGBF
: A color format providing a 32-bit floating point value for each color channel in the RGB set.. 33

BRDF
: The Bi-directional Reflectance Distribution Function is a function that is used to describe the light reflective qualities of a surface. 10, 11

BSSRDF
: The full version of BRDF including sub-Surface Scattering properties for simulating the light scattering behavior that takes place beneath the immediate surface of an object. 10

CGI
: Computer Generated Imagery. Images, whether still or motion, that are produced entirely on a computer. CG images might be composited into real footage as also strongly manipulated real footage is sometimes called CGI as well as CG is also more general acronym for *computer graphics*.. 5

distant environment
: Scenery that is assumed to be infinitely far away and thereby out of interest for light interaction. Usually not modeled and textured like the local scene. 3, 14

environment geometry
: The rough geometry used for depth reference in the GDIBL method. 20

GDIBL
: Geometrically Derived Image Based Lighting; the proposed method of this report deriving novel lighting conditions by combining light information from multiple light probes. iii, 3, 11, 19, 20, 22–24, 29, 33, 36

GI
Global Illumination is a common name for lighting algorithms that include both direct and indirect lighting. 3, 12, 33

HDR
High Dynamic Range; a wide range of light intensities and wave lengths. Given the dynamic asset the representation only needs to include from the darkest dark to the brightest bright in that particular case. 12, 13, 17, 22, 23, 33

HDRI
High Dynamic Range Imaging. Images using 16, 32 or 48 bit per channel to represent a high dynamic range of light. 13

IBL
Image Based Lighting; a common name deriving illumination of synthetic objects in computer graphics from HDR images of real world lighting. 3, 4, 11–14, 17, 19, 23, 24, 29, 34

IDW
Inverse distance weighted interpolation renders samples further away less influential than samples closer to the interpolated point.. 22, 23

Lambertian
Also called *perfect diffuse*; a Lambertian surface reflects incoming light uniformly in all directions. 11, 35

LDR
Low Dynamic Range; a common term used about a more narrow range of light than medium or high dynamic range. This term has also, by the introduction of HDRI been applied to the traditional 8-bpc images. 13, 33

LOS
A line of sight is a straight line between two points that is not blocked by any objects. 20, 21, 24

QO
A phrase used about the geometric surface point in the scene currently querying the scene for irradiation information. 22, 23

WCS
The World Coordinate System is a three dimensional right hand coordinate system, in which all locations, orientations and directions are defined for the implementation described in this report. 30

Listings