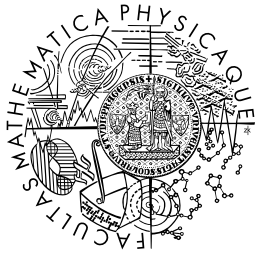


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Marcel Hruška

A Methodical Approach to the Evaluation of Appearance Computations

Department of Software and Computer Science Education

Supervisor of the master thesis: doc. Alexander Wilkie, Dr.

Study programme: Computer Science

Study branch: Computer Graphics and Game Development

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: A Methodical Approach to the Evaluation of Appearance Computations

Author: Bc. Marcel Hruška

Department: Department of Software and Computer Science Education

Supervisor: doc. Alexander Wilkie, Dr., Department of Software and Computer Science Education

Abstract: Abstract.

Keywords: key words

Contents

Introduction

Goals

Thesis organization

1. Rendering and Color Science

This chapter serves as an introduction to the computer graphics and the color science. We briefly overview basic aspects of these fields, mainly to familiarize the reader with some of the fundamental processes, their backgrounds and usages. We also establish the terminology, such as *rendering* or *RGB color space*, that will be used throughout the thesis frequently. A significant part of the following sections is based on Wyszecki and Stiles [30], Wilkie [28], Nimier-David et al. [20] and Pharr et al. [23].

First, we discuss the mechanisms behind the light and colors and then we look into the process of the physically based image reproduction.

1.1 Light

According to the definition by Barbrow [5], the light is "any radiation capable of causing a visual sensation directly". In other words, the visible light is an electromagnetic radiation that is perceivable by the human eye and allows us to see the objects around us.

As all electromagnetic radiations, the light also propagates in form of waves. The oscillation direction of these waves does not change the color of the light but it may interact differently with the reflective/refractive objects as it passes through them. A representation of such wave propagation is displayed in Figure 1.1.

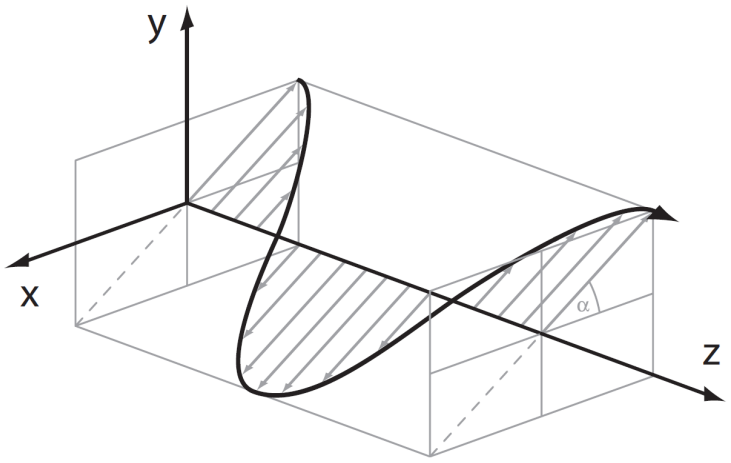


Figure 1.1: A propagation of wave [28]

Normally, by the term light we mean the *visible light* (also called the *white light*) which consists of multiple waves of unique frequencies (wavelengths). There are no exact boundaries to the visible spectrum as distinct human eyes might perceive light slightly differently but the lower boundary is estimated between 360 and 400nm and the upper boundary from 760 to 830nm [25]. Above this range there is the infrared light and below the ultraviolet. An explanatory image of the known electromagnetic wavelengths can be found in Figure 1.2.

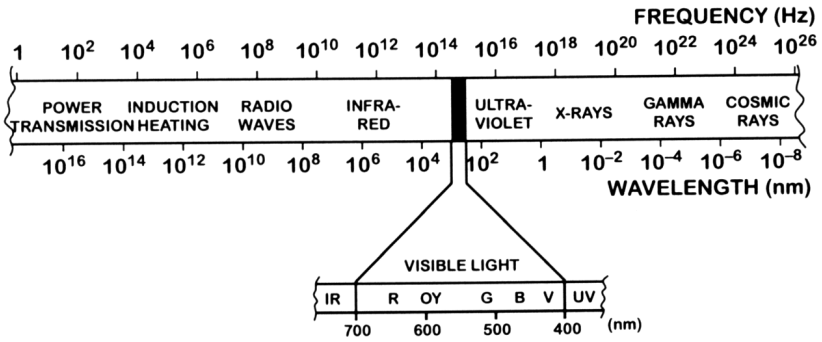


Figure 1.2: An image displaying various wavelengths [28]

1.1.1 Color

While we observe an illuminated object, three different signals are sent from the eye sensors (rods and cones) to the brain, each representing a red, a green or a blue wavelength. When put together inside the brain, they form a sensation of the final color.

To categorize the colors, we formed several reproducible representations of them called the *color spaces*. A natural decision was to create an *RGB color space* as it directly correlates with the signals sent from the human eye's rods and cones. Note that multiple variations of the RGB color space exist such as *sRGB* or *Adobe RGB*. An illustrative comparison between several color gamuts is shown in Figure 1.3.

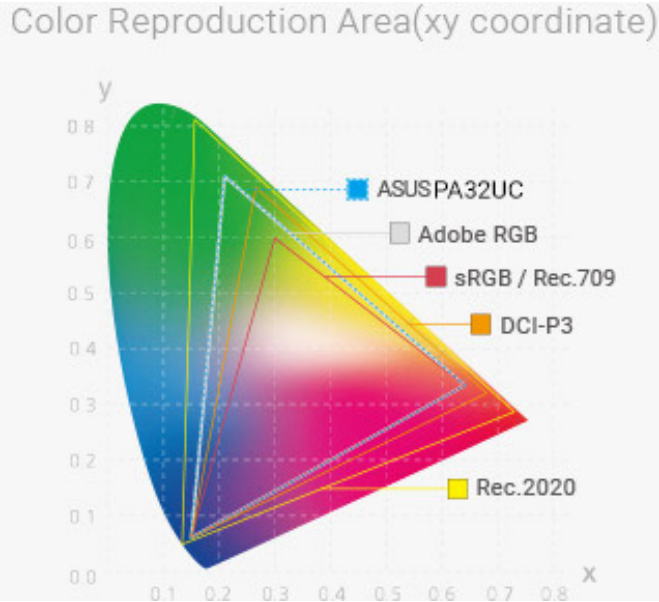


Figure 1.3: An illustrative comparison between five RGB gamuts by Asus [1]

In 1913, the Commission internationale de l'éclairage (International Commission on Illumination), shortly CIE, was formed as an authority that defines almost everything that concerns colors and their perception. In 1931, they conducted

[1] https://www.asus.com/Microsite/ProArtMonitor/experience-truecolor.html

the color matching experiments to obtain the three color matching functions that would convert the color stimuli perceived in our eyes to the *CIE RGB* color space. As these functions had a negative component, a new imaginary color space was created called *CIE XYZ*. These conversions are further described in subsection 1.1.2.

CIE also defined *CIE L*a*b** color space, standard illuminants D65 and D50 and many others.

1.1.2 Conversions to tristimulus color spaces

The conversion from the spectrum to a RGB color space:

1. Compute the tristumulus value XYZ using the CIE color matching functions (shown in Figure 1.4)

$$X = \int P(\lambda)\overline{x}(\lambda)d\lambda$$
$$Y = \int P(\lambda)\overline{y}(\lambda)d\lambda$$
$$Z = \int P(\lambda)\overline{z}(\lambda)d\lambda$$

   , where $P(\lambda)$ is the spectral power distribution and $\overline{x}(\lambda)$, $\overline{y}(\lambda)$ and $\overline{z}(\lambda)$ are the color matching functions.

2. Convert the XYZ to the desired RGB color space using a transformation matrix. Depending on the specific RGB color space, the matrix differs — an example of CIE XYZ to sRGB conversion is demonstrated in item 2.

$$r = 3.240X - 0.969Y + 0.55Z$$
$$g = -1.537X + 1.875Y - 0.204Z$$
$$b = -0.498 + 0.041Y + 1.057Z$$

3. (Optional) As the resulting r,g,b values may be negative, a gamut mapping might be necessary.

1.1.3 Photometry and Radiometry

Two different science fields were developed to quantify the light – the *photometry* and the *radiometry*. The radiometry recognizes the light as an electromagnetic radiation while the photometry focuses more on the human perception of the light. Despite the distinct purposes, their quantities are often easily convertible.

Radiometric Quantity	**Photometric Equivalent**
Spectral radiant energy $[J]$	Luminous energy $[Lumen - second]$
Radiant flux $[W]$	Luminous flux $[Lumen]$
Irradiance $[W.m^{-2}]$	Illuminance $[Lumen.m^{-2}]$
Radiant intensity $[W.sr^{-1}]$	Luminous intensity $[candela = Lumen.sr^{-1}]$
Radiance $[W.sr^{-1}.m^{-2}]$	Luminance $[candela.m^{-2}]$

[2] `https://scipython.com/blog/converting-a-spectrum-to-a-colour/`

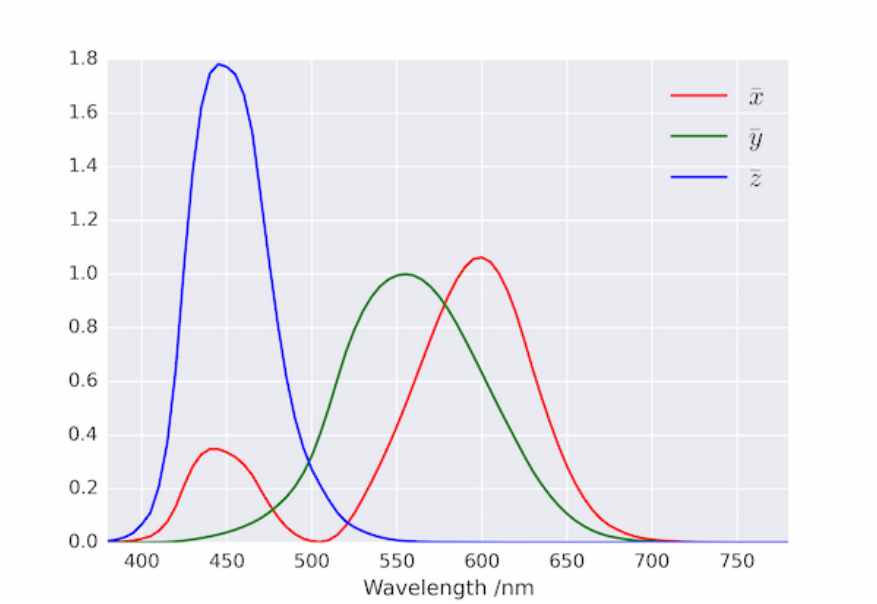


Figure 1.4: Color matching functions plotted in Python[2]

A brief description of each of them:

Spectral radiant energy Amount of the light energy at some wavelength

Radiant flux Amount of the light energy with respect to time

Irradiance Flux at a specific point (space)

Radiant intensity Flux in a direction (*sr* (steradian) is a unit of the solid angle — surface on a unit sphere, whole sphere has 4π steradians)

Radiance Spatial and directional flux

The relationship between these quantities is described by the *spectral efficiency function*. It states how efficiently a human eye reacts to different wavelengths, i.e. we can detect some wavelengths more easily than others. As we can see in Figure 1.5, the scotopic (night) perception peaks at around 507nm and the photopic (day) at 555nm.

1.2 Physically based rendering

One of the ultimate goals of the computer graphics is the ability to reproduce visually plausible and physically coherent images that should be indistinguishable from a photograph based on a description of a scene. Such process is called the *photo realistic rendering*. In this thesis, we abbreviate the term and call it simply the *rendering* as the non-photo realistic one does not concern us.

Depending on the implementation, the renderer simulates various phenomena commonly seen in nature such as light reflections, refractions, shadows, etc. Providing a powerful hardware, modern renderers adapt various physical models (or their approximations) of light transport or material properties to provide accurate photo realistic results. In reality, the renderers are so capable that the

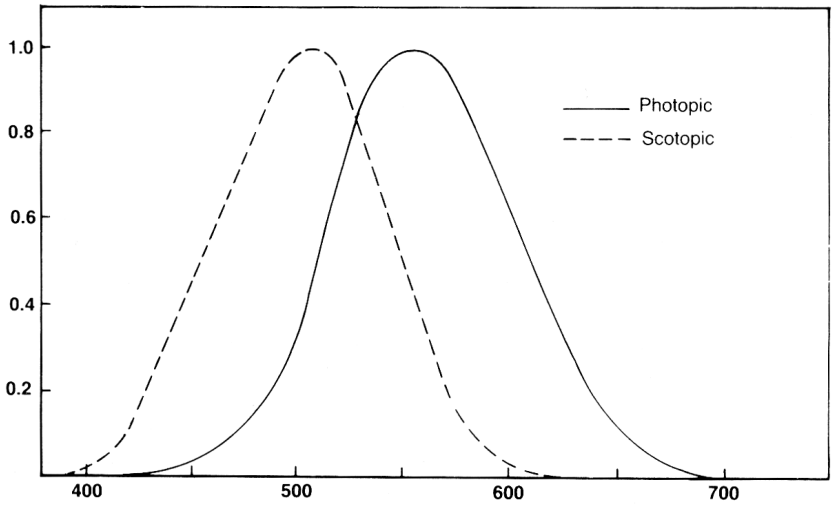


Figure 1.5: Relative luminous efficiency function [28]

rendered images are almost identical to the real life photos. An example can be seen in Figure 1.6.

Figure 1.6: An image generated with the Corona Renderer[3]

The main idea is similar for every renderer:

1. A 3D digital scene is described by the objects it contains
2. A light simulation algorithm runs for every visible pixel from the viewer
3. Upon object interaction, the shading of the intersected point is computed
4. As the algorithm terminates, a picture (”photograph”) of the scene called the *render* is created

[3] `https://corona-renderer.com/gallery`

Please see figure Figure 1.7 for a simple demonstration of this workflow.

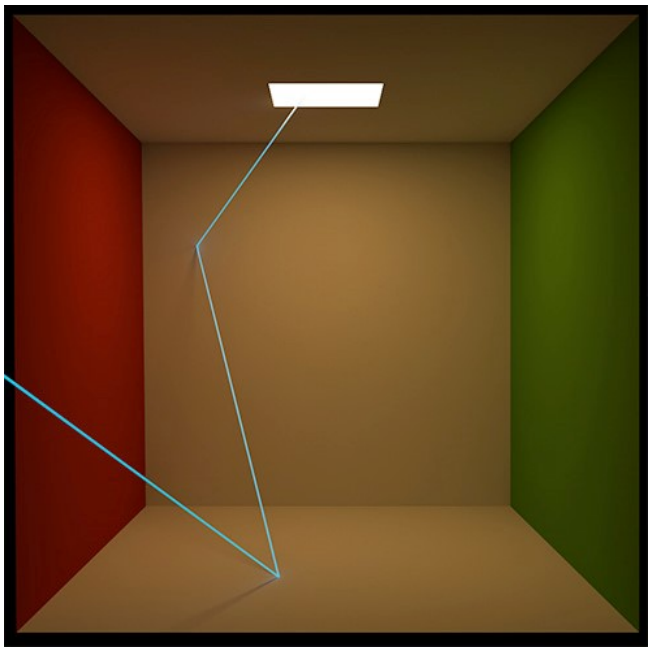


Figure 1.7: A visualization of a light transport algorithm (path tracer) [4]

1.2.1 Digital scene

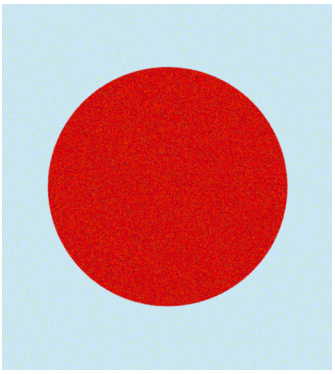
Basic elements of a digital scene are roughly the same for each renderer.

Camera A camera (or a sensor) in a digital scene works in the same manner as in real life — it records a picture. Generally, you may define the coordinate position and the viewing vectors but also the properties such as focal distance or the type of the film.

Light source The scene needs to be illuminated by one or multiple sources in order to be visible. The common kinds of lights are point light, area light, spot light or environment (constant) lighting.

Objects The actual visible contents of the scene are objects. Almost all rendering systems offer a choice to either use their precomputed basic geometry such as spheres or triangles or to include a mesh geometry described in an external file (usually created by an external modeling software). These objects must state their material properties so that the algorithm may correctly interact with them, e.g. diffuse vs. reflective material.

Unfortunately, as each renderer may have a very unique implementation details, the formats of the scenes are vastly different. For example, mitsuba uses XML but PBRT has its own specific format. An example of a simple scene for Mitsuba2 can be found in Figure 1.8.



```
<scene version="2.0.0">
 <!-- Light transport algorithm -->
 <integrator type="path"/>

 <!-- Camera looking at the sphere -->
 <sensor type="perspective">
  <transform name="to_world">
   <lookat origin="0,-6,0" target="0,0,0" up="0,0,1"/>
  </transform>
 </sensor>

 <!-- Red sphere in the middle -->
 <shape type="sphere">
  <bsdf type="diffuse">
   <rgb name="reflectance" value="1.0,0.0,0.0"/>
  </bsdf>
 </shape>

 <!-- Light blue light all around the scene-->
 <emitter type="constant">
  <rgb name="radiance" value="0.6,0.8,0.9"/>
 </emitter>
</scene>
```

Figure 1.8: A simple scene rendered with Mitsuba2 (left) along with its scene description (right)

1.2.2 BRDF

The fundamental part of the rendering process is its implementation of the light transport simulation. This and the following sections will describe the physics theory and the models behind the light transport. Then we take a look at the specific algorithms.

The materials are described by *Bidirection Distrubtion Reflectance Function*, shortly *BRDF* [19]. It looks as follows:

$$f_r(\omega_i, \omega_o) = \frac{dL_o(\omega_o)}{L_i(\omega_i) cos\theta_i d\omega_i} \tag{1.1}$$

This function is given the incoming vector ω_i, the outgoing vector ω_o and it states how much radiance is reflected from the direction ω_i ($L_i(\omega_i)$) to the direction ω_o ($L_o(\omega_o)$) for the specific material. An image interpretation of the function is in Figure 1.9. As it is a distribution function, we can also reformulate it's meaning as a probability density that a defined amount of light energy gets reflected from ω_i to ω_o.

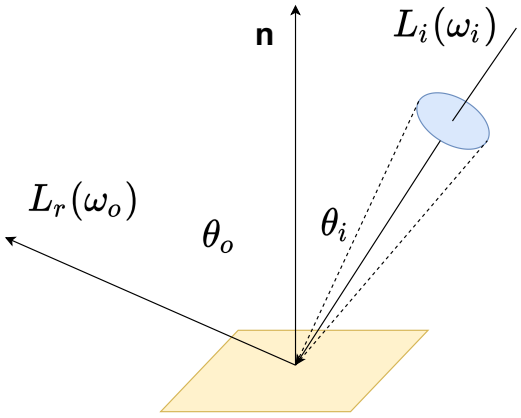


Figure 1.9: Bidirectional Distribution Reflectance Function

In the simplest cases, the BRDF states how reflective the surface of an object

is. The renders of diffuse, rough glossy and mirror materials are compared in Figure 1.10.

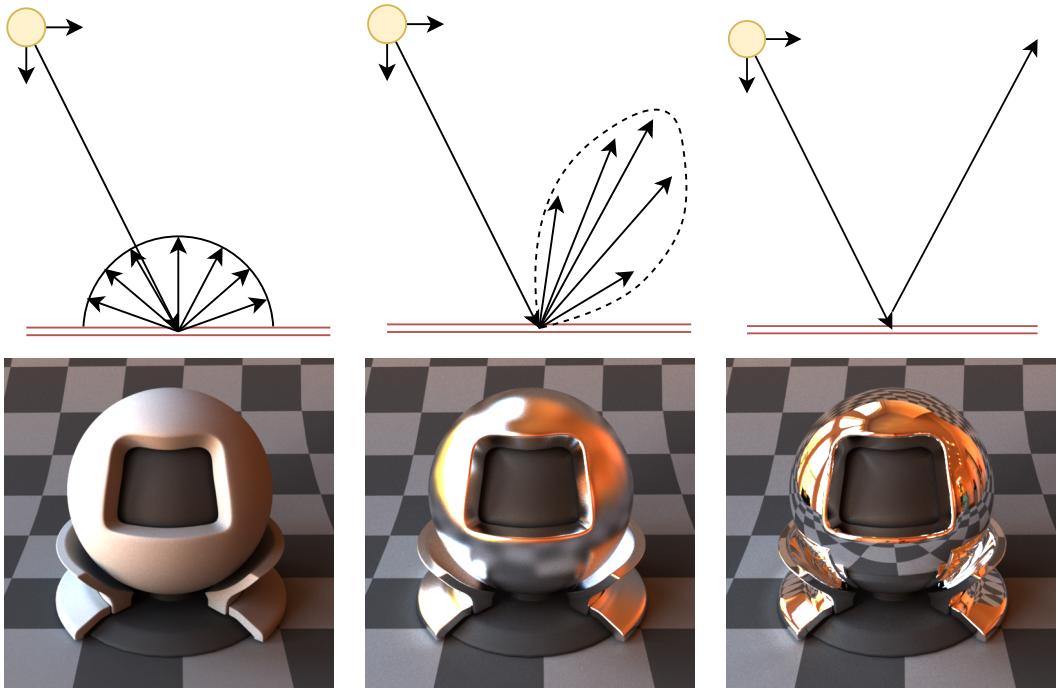


Figure 1.10: A preview of diffuse (left), glossy (middle) and mirror (right) materials rendered in Mitsuba2 along with their illustrative BRDF visualizations

Physically based BRDFs must fulfill several properties 11:

Heimholtz reciprocity The amount of reflected energy from the incoming direction to the outgoing direction is equal to the amount of energy in the reversed directions ($f_r(\omega_i, \omega_o) = f_r(\omega_o, \omega_i)$).

Energy conservation The amount of reflected energy cannot be larger than all received energy.

Positivity BRDF is always positive ($f_r(\omega_i, \omega_o) \geq 0$).

Note that BRDF concerns only opaque surfaces. There exist multiple distribution functions that describe behavior of other materials, for example:

BTDF Describes light transmission

BSDF Combination of BTDF and BRDF (e.g. glass, water)

BSSRDF Considers scattering of the light under the surface as well (skin)

1.2.3 Global Illumination

With the BRDF defined, we can now formulate an equation that evaluates the global illumination of a scene — illumination of each point from all light sources. It is generally called the *rendering equation* [16]:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + L_r(x, \omega_o) \tag{1.2}$$

Let's break it down first:

x is the currently computed point in the scene.

L_o is the outgoing radiance.

L_e is the emitted radiance of the point x as x can be on a light source.

L_r is also called the *reflectance equation* and it states the total amount of the reflected radiance for all contributions of the incident radiance. Hence, it is an integral over the upper hemisphere over x that look as follows:

$$L_r(x, \omega_0) = \int_\Omega f_r(x, \omega_o, \omega_i) L_i(x, \omega, i) cos\theta_i d\omega_i \tag{1.3}$$

, where

$f_r(x, \omega_o, \omega_i)$ is the BRDF of x as defined in Equation 1.1.

$L_i(x, \omega_i)$ is the incoming radiance from a light source.

An image interpretation of the reflectance equation can be seen in Figure 1.11.

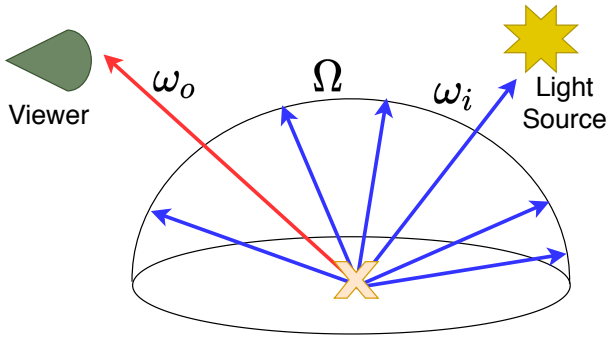


Figure 1.11: Reflectance Equation

As a matter of fact, each light transport algorithm tries to solve some of the formulations of the rendering equation.

Interestingly, the light transport is recursive in nature. As we can see from the rendering equation, to compute the outgoing radiance at a certain point x, we need to know all the contributed incoming radiances. These do not necessarily have to originate at a light source — the incoming radiance may come from another, non-emitting point y in the scene as a result of the rendering equation computed at the point y.

1.2.4 Monte Carlo integration

Before we proceed to the actual algorithms that evaluate the rendering equation, we briefly introduce a method that is used to approximate the definite integral part of the equation — *Monte Carlo integration* [9].

Formally, for a multidimensional definite integral

$$I = \int_\Omega g(x) dx \tag{1.4}$$

Monte Carlo (MC) estimates I as

$$\langle I \rangle = \frac{1}{N} \sum_{k=1}^{N} \frac{g(\xi_k)}{p(\xi_k)}; \xi_k \propto p(x) \tag{1.5}$$

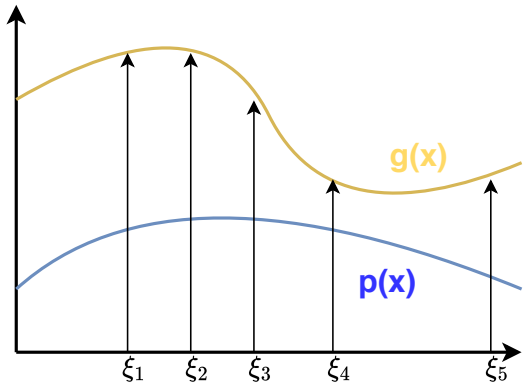


Figure 1.12: Monte Carlo method 2D visualization

In other words, Monte Carlo is a non-deterministic method that sums N randomly chosen samples ξ_k, computes their values $g(\xi_k)$ and averages them. To reduce variance, an importance sampling is introduced by drawing samples from a distribution $p(x)$ that is chosen for each specific problem to approximate the former $g(x)$ function. In reality, the importance sampling ensures that if we pick some samples twice as much, we decrease their weight to half.

There exist other methods that are used to approximate integrals such as deterministic quadrature or Markov Chain Monte Carlo (MCMC).

1.2.5 Light transport algorithms

Path tracing

Over the years, a large number of various light transport algorithms and their variations have been developed, where each has it's own benefits. The one that we will mention the most in this thesis is called the *path tracing*. It's core idea is simple:

1. For each pixel in the image plane, shoot a primary ray r from camera into the scene.
2. If r hits a non-emitting object at point x:
   - 2.1. Compute BRDF at x.
   - 2.2. Generate a new random direction ω. Ideally, the distribution of the generated direction should be proportional to the BRDF — e.g. diffuse BRDF would generate a direction uniformly over a hemisphere while glossy BRDF would prioritize samples from the reflectance lobe (look at Figure 1.10).
   - 2.3. Add the BRDF value to the final color of the pixel.

2.4. Check for a terminating condition — there exist several options, usually a combination of them is applied:

Maximum depth User specified maximum number of recursions.

Russian Roulette Randomly choose if the ray survives, with each consecutive ray the chance lowers.

BRDF-proportional Depending on the surface material, we decide whether the ray survives or not. For example, reflective or refractive surfaces need a lot more recursions as they propagate the light further to the scene than diffuse surfaces.

2.5. In case the termination was not successful, bounce – shoot a secondary ray r fromthe point x in the direction ω and continue from step 2.

3. If r hits an emitting object (light source), add it's emission L_e to the final color of the pixel

4. If no scene geometry is hit, terminate the algorithm and add the color of the surrounding light (if there is any).

The bouncing of the light in the scene nicely correlates with the recursive nature of the rendering equation. Even though the path tracing is a slow algorithm (e.g. not suitable for real-time rendering in games), it's variations can be extremely accurate, even indistinguishable from a real photograph.

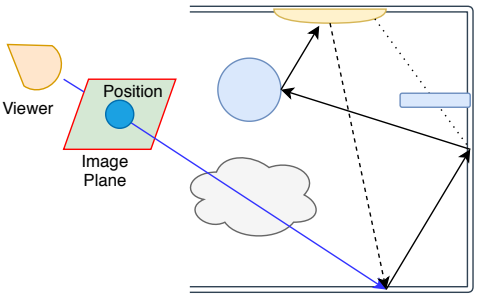
MIS In the algorithm described above, the direct illumination computation of each scene intersection is dependent only on the BRDF of the intersected surface and consequent walk to the light source. However, such scenario that the light source is hit at the end of every walk is greatly dependent on the number of samples and the maximum allowed depth of the recursion. Consequently, this creates variance which can be easily improved the integration of the *multiple importance sampling* (MIS). Generally, it involves a combination of multiple sampling techniques but in our case it is a combination of the BRDF proportional sampling and the light source sampling — in each step of the path tracing, every light source that is visible from the intersected point contributes to it's value. Both sampling methods are, of course, weighted to avoid an over-illumination.

Volumes Another aspect that needs to be accounted for in the rendering process are volumetric objects such as fogs or smokes. Normally, there are two ways that a volumetric object may effect the light passing through it. The volume is either attenuating the light by absorbing it or scattering to different directions. Or the volume can also strengthen it by emitting light (e.g. flame) or scattering light from different directions to the current one.

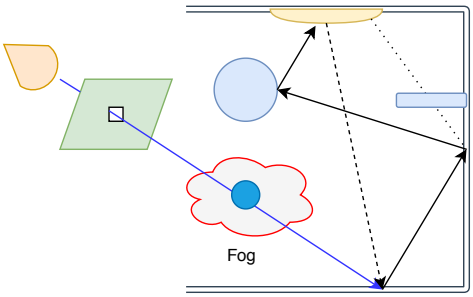
A single walk of a path tracer capable of volume tracing and MIS sampling is visualized in Figure 1.13.

Other methods

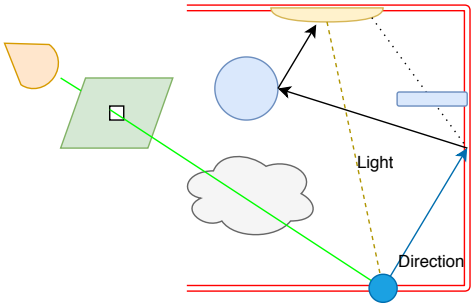
As the path tracing is of our main concern in this thesis due to its physically based and unbiased properties, some of the other global illumination techniques are described only briefly here:



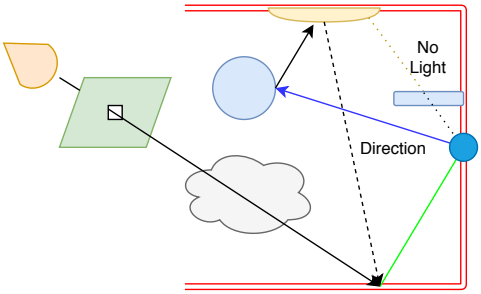
(1) Primary ray created



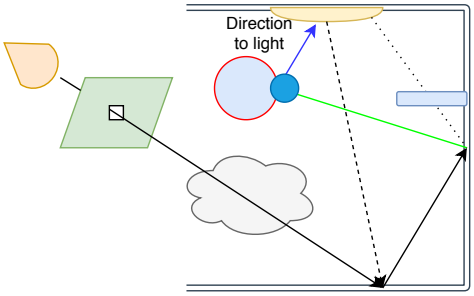
(2) Passing volume, attenuation



(3) Geometry hit, new direction, BRDF + light added



(4) Geometry hit, new direction, BRDF added (light obscured)



(5) Geometry hit, new direction points to light, add contribution and terminate

Figure 1.13: A visualization of a single walk in path tracer

Ray tracing [12] Similar to the path tracing but there are no bounces from the surfaces, simulates only reflections, refractions, scattering etc. Capable of realtime rendering these days.

Photon mapping [15] Two rays are traced independently — from the camera and from a light source until termination, then the radiance is computed based on their final positions. Faster at some scenarios but biased (does not have to converge to a correct solution).

Radiosity [24]Uses the finite element method instead of Monte Carlo. View independent, the light is traced from the source and bounced (possibly) to the viewer. Good for precomputations.

1.3 Spectral Rendering

So far, we've considered the colors to be internally represented by a tristimulus color space during the rendering process. For the explanation purposes, let's consider it to be RGB color space – the objects are defined in RGB, the path tracing step colors are RGB and the output color is RGB. In a large number of scenarios, this workflow is sufficient as we are capable of simulating a majority of the common aspects (e.g. optics) while keeping the rendering simple and robust. Unfortunately, the RGB color space is only a fraction of the visible gamut and does not contain any information about the light as an electromagnetic radiation. Consequently, we are loosing a significant amount of information, causing the colors to be at times inaccurate and some phenomena completely impossible to render.

Therefore, a new approach to the rendering has been introduced that internally represents the colors as a spectrum distribution function instead of a tristimulus color space — the *spectral rendering*. The core idea is to track and sample several wavelengths at once for each step of the path tracing and to perform the integration all over them. We also generalize the BSDF to account for the wavelengths: $f_r(\lambda, \omega_i, \omega_o)$.

Most of the phenomena evaluated in this thesis are a direct consequence of the light's nature as an electromagnetic radiation, hence the primary focus is placed on the spectral rendering. The following sections are largely based on Wilkie and Purgathofer [29].

1.3.1 Color representation

If the spectral rendering is desired, there is a need to store the spectral distribution function. While several techniques are feasible, it is often a matter of a simple trade-off between the precision and the performance. For example, sampling wavelengths uniformly each 10nm would yield significantly more accurate results than sampling only four wavelengths in total. However, such approach might become unbearable in terms of speed and memory. Moreover, the spectral functions are mostly quite smooth. Some examples of these functions can be seen in Figure 1.14.

A typical approach is to sample at larger ranges (10 or more nanometers) and combine it with the basis functions [22]. More approaches and their details are briefly explained in Wilkie and Purgathofer [29].

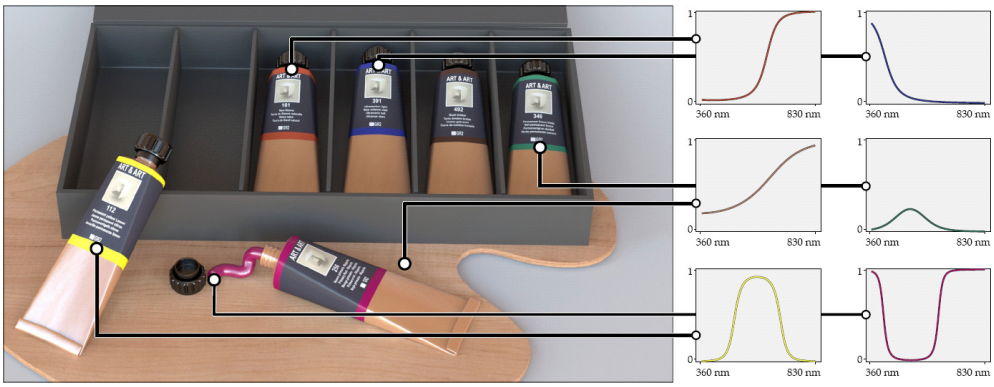


Figure 1.14: Spectral curves measured for different colors[14].

1.3.2 Advantages

The spectral rendering presents the ability to reproduce the colors in a more photorealistic way. We might not see it at first, but the colors produced by a conventional RGB renderer tend to be slightly over-saturated as they do not account for the spectral characteristics which might attenuate the final color. A comparison between the RGB and the spectral render of the same scene by Mitsuba2 is shown in Figure 1.15.

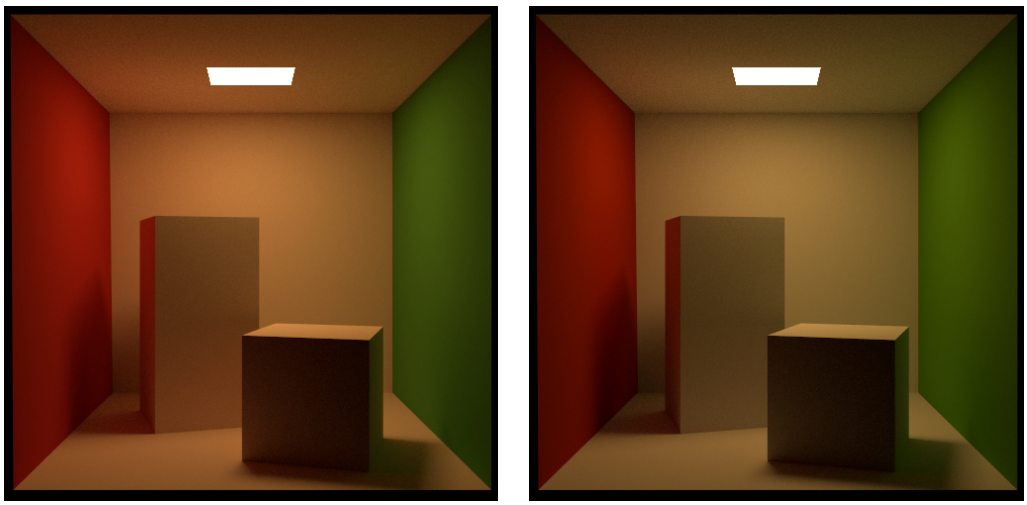


Figure 1.15: A comparison between the RGB render (left) and the Spectral render (right) of the same scene by Mitsuba2.

Still, the biggest improvement is the possibility to reproduce some of the natural phenomena for which the tracing of multiple wavelengths is an absolute necessity. Namely, those are:

Fluorescence Absorption and re-emission of a different color

Dispersion Splitting of the white light into its wavelength components via refraction

Polarisation Change of the oscillation direction of a light wave

Iridescence Thin layer constructive/destructive interference

They are explained in detail in chapter 2.

1.3.3 Disadvantages

On the other side, the need to numerically integrate over multiple wavelengths introduces a problem of the chromatic noise. Even though the performance is not a key factor, it obviously worsens as well.

Moreover, we are still unable to properly display spectral images on the current monitors. The only viable way is to store the distinct spectral bands as separate images. Clearly, it is quite inconvenient to have multiple results instead of one. Therefore, the rendering needs to have a well-done spectrum to final RGB conversion as the final image is still being displayed on an RGB monitor. Fortunately, the conversion is well-defined and fairly simple to implement.

Due to the fact that the RGB gamut is only a subset of the visible spectrum, the situation gets significantly more complicated for the reversed conversion. As there exist infinitely many spectra for one RGB value, several techniques were proposed to convert the tristimulus to the spectral domain — commonly called the *spectral upsampling*. For those who may be interested in the details of the current development to the spectral upsampling, refer to the article by Jakob and Hanika [14] which proposes a solution that is capable of converting full sRGB gamut with zero error.

There exist several reasons to integrate the spectral upsampling, mainly because the spectral values are a lot harder to obtain and to use. You need a specific device (spectrometer) that would measure the color values under a specific light and then use regularly distributed samples from it as an input. Because of the reproducibility of the RGB color space and its legacy usage (lots of existing textures are already defined in RGB), it is a lot more convenient to input the values of textures as an RGB value, convert them internally to the spectral domain and convert them to the desired color space for the output image.

2. Appearance Computations

So far, we've covered the fundamental basics of the light and the rendering process to be able to comprehend the more advanced techniques practiced in the computer graphics. As we've mentioned before, our primary goal is to evaluate the computational accuracy of several specific appearance sensations. Even though they are quite common in every day life, their integration to the modern renderers is, to this date, rare. In this chapter, we discuss these phenomena individually — their manifestations in the nature, the physics behind them and finally, their computations in the rendering process.

2.1 Reflectance

The reflective surfaces are a surprisingly common sighting. As the perfectly diffuse materials basically do not exist in the nature, a large set of the materials that surround us are considered glossy. In subsection 1.2.2, we explained the bidirectional reflectance distribution function that defines the reflective properties of a material.

2.1.1 Fresnel equations

It is necessary to know the basics of the geometry optics to be able to properly define a reflectance model. First of all, the *Snell's law* [23]

$$\eta_i sin\theta_i = \eta_t sin\theta_t \tag{2.1}$$

states that the incoming angle θ_i (angle between the surface normal and the incoming direction) times the *index of refraction* of the entering medium η_i must be equal to their transmitted counterparts. In other words, knowing the indices of refraction of the entering and the leaving media and the incoming direction, we can compute the transmitted direction.

The index of refraction (IOR) varies from material to material (e.g. IOR of glass is $\sim$1.5) and it essentially states how fast the light travels through the specific material.

However, this gives us only the direction of the refracted light. In most cases, we also need to know the ratio between the amount of reflected and refracted light. Depending on the polarization of the light (further explained in section 2.2), the *Fresnel equations* [23] take the two following forms:

$$r_s = \frac{\eta_t cos\theta_i - \eta_i cos\theta_t}{\eta_t cos\theta_i + \eta_i cos\theta_t}$$
$$r_p = \frac{\eta_i cos\theta_i - \eta_t cos\theta_t}{\eta_i cos\theta_i + \eta_t cos\theta_t}$$

From these, we can compute the *Fresnel reflectance* for an unpolarized light:

$$F_r = \frac{1}{2}(r_s^2 + r_p^2) \tag{2.2}$$

The transmitted energy is equal to $1 - F_R$ due to the energy conservation law.

Note that the previous computations describe only a subset of the materials called the *dielectrics*, which are the materials that do not conduct electricity and are capable of transmitting light, such as glass, water, diamond, etc. The second large group consists of the *conductors* which are basically all metals/materials with opaque surfaces. Actually, the light is also transmitted into the conductor, however, due to its physical properties, it is quickly absorbed. There exists a third group of the *semiconductors* which are very rarely considered in the physically based rendering and therefore we skip them. A comparison between a dielectric and a conductor is shown in Figure 2.1.

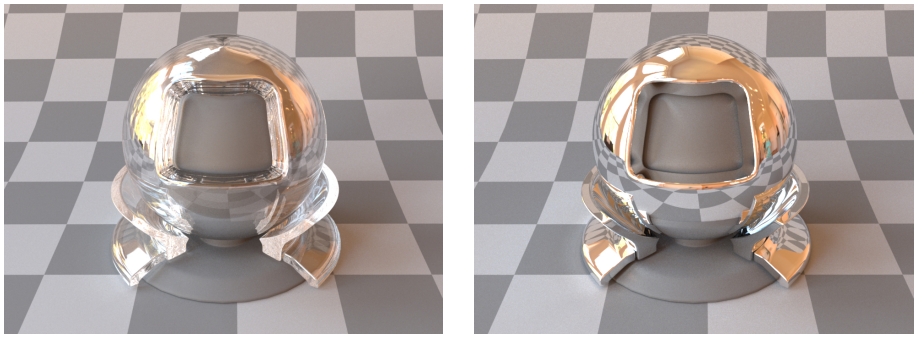


Figure 2.1: A preview of a dielectric (diamond, left) a conductor (aluminium, right) rendered in Mitsuba2 [4]

While this is a matter of simply computing the Fresnel equations, the practice is more complicated as most of the commonly seen materials are not perfectly smooth. They may be created rough on purpose or slight imperfections due to the manufacturing errors cause that the surfaces are generally rough.

2.1.2 Microfacet theory

With that in mind, the *microfacet theory* was developed by Cook and Torrance [10] to address this aspect and provide a theoretical representation of the rough surfaces.

The main idea is that a rough surface consists of the *microfacets* – a collection of very small surfaces distributed statistically throughout the whole underlying *macrosurface*. The aggregate behavior of the computed values for each of these microfacets determines the final scattering. An example of such distribution is shown in Figure 2.2.

As these microfacet computations are local, we need to consider the possibility that they might obscure each other. Three main aspects are accounted for:

Masking Microfacet is not visible from the viewer

Shadowing Microfacet is not reachable from the light source

Interrecflection Bounces between the microfacets

Many variations to the Cook-Torrance model have been developed, such as it's predecessor Torrance-Sparrow [26] or Oren-Nayar [21] for diffuse reflectance.

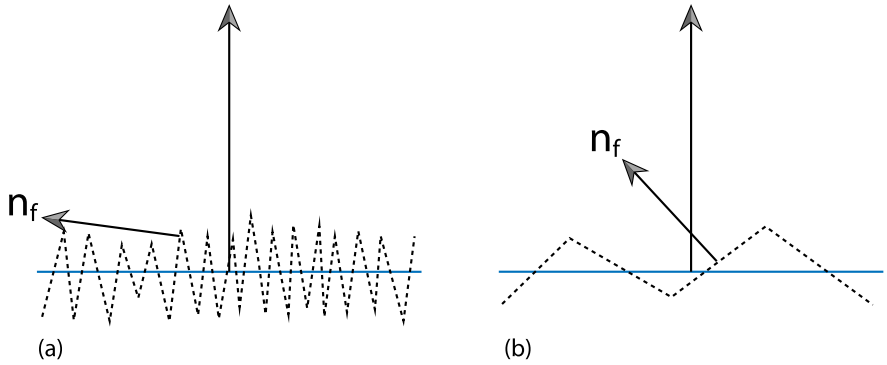


Figure 2.2: A demonstration of a very rough (left) and a relatively smooth (right) microfacet distribution [23]

In this thesis, we focus on the distribution functions of the microfacets as they are ultimately the deciding factor of the rough surface look. A nice comparison of the three commonly used microfacet distributions — *Phong*, *Beckmann* and *GGX* – along with their distribution functions, masking functions and sampling equations can be found in the Walter et al. [27]. As the exact formulations of those three methods are not a necessity for this thesis, we provide only a brief overview for each of them and an illustrative comparison between the GGX and the Beckmann of the same roughness in Figure 2.3.

Phong Even though the Phong distribution is purely empirical (not physically based), it is still quite popular choice for the microfacet distribution as it is simple to implement and provides sufficient results.

Beckmann The Beckmann distribution [6] is already physically based and for a long time has been considered the best solution to the rough surfaces as it is based on the Gaussian roughness. However, with the parameters set appropriately, it still provides the results very similar to the Phong distribution.

GGX The GGX distribution [27] was introduced as an improvement over the Beckmann's solution for some cases. It maintains stronger tails, thus better shadowing and is based on the measured data of the real rough materials.

2.2 Polarization

Similarly to all kinds of the electromagnetic radiation, the light also propagates through space as a wave. The oscillation direction of this wave neither defines nor modifies the color of the light but it makes the light behave differently upon an interaction with certain materials. Figure 2.4 explains the different directions of a wave.

In the light's natural state (sun, common light bulb), the directions of it's oscillation are arbitrary — such light is called *unpolarized*. The *polarized* light maintains a restricted direction of oscillation and it is a result of a *polarization* process. Note that the light is often only partially polarized as the restriction of the direction does not have to be perfect and allows some variations.

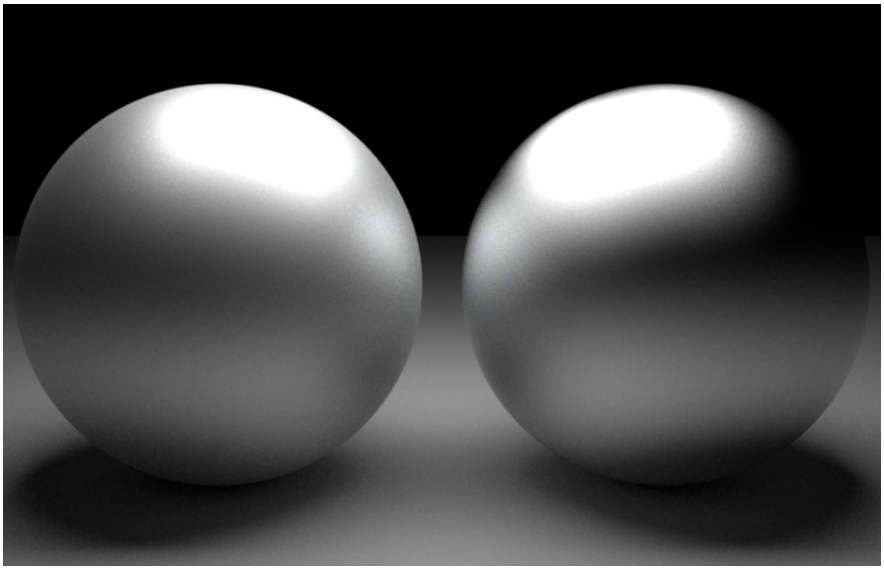


Figure 2.3: A rough aluminium sphere with the GGX distribution (left) compared to it's Beckmann equivalent (right) rendered in Mitsuba2

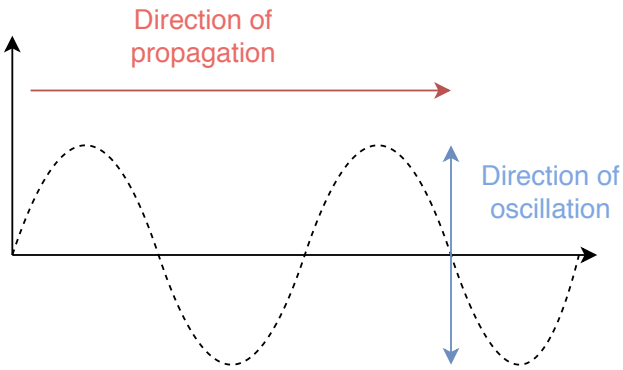


Figure 2.4: Demonstration of the direction of oscillation and the direction propagation

Depending on the shape of their electric fields, we distinguish three types of polarization which are demonstrated in Figure 2.5.

To create a linearly polarized light, one may put a dielectric object in the direction of propagation of an unpolarized light. Due to the optical properties of the dielectrics, the reflected and the transmitted light will be polarized proportionally to the angle of the incidence. The angle at which the reflected light is perfectly polarized is called the *Brewster's angle* [8]. It is computed by the following formula:

$$\theta = arctan(\frac{n_2}{n_1}) \tag{2.3}$$

, where n_1 is the IOR of the incident medium and n_2 of the transmitted medium.

In this case, we distinguish two types of the linearly polarized light depending on their relative orientation to the incident plane. The reflected light is called

[1] `http://hyperphysics.phy-astr.gsu.edu/hbase/phyopt/polclas.html`

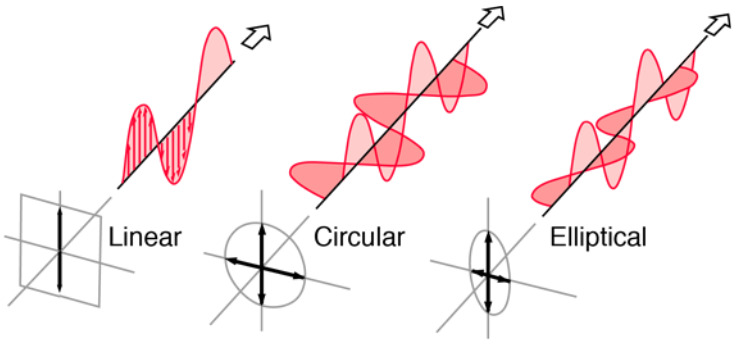


Figure 2.5: An illustrative demonstration of the different types of polarization [1]

p-polarized as its oscillation is parallel to the plane of incidence. The transmitted light is called *s-polarized* as its oscillation is perpendicular (from the German word) to the plane of incidence. Both are shown in Figure 2.6.

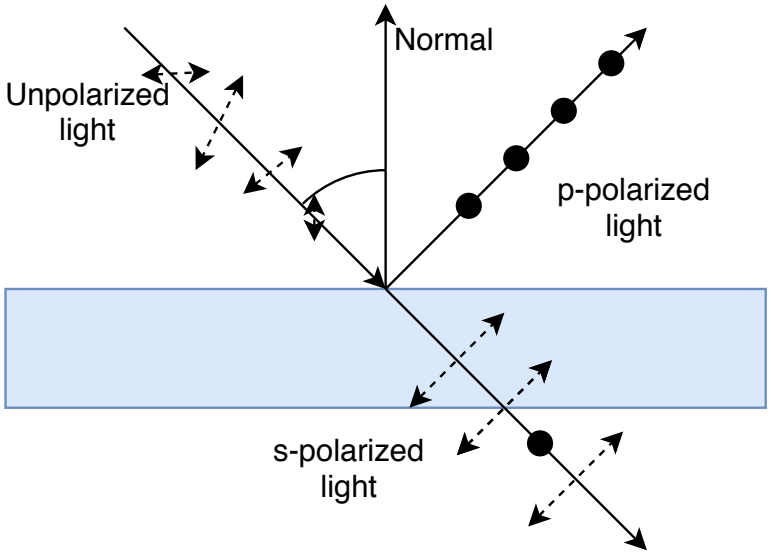


Figure 2.6: Prefectly p-polarized reflected and partially s-polarized light refracted from a dielectric interface at the Brewster's angle

The principle of Brewster's angle is used in a material called the *polarizer*. As the name suggests, it polarizes the light, restricting it's direction of oscillation accordingly to the specifics of the polarizer. If the light that passes through the polarizer is of the opposite (perpendicular) direction to the polarizer's transmission orientation, it won't let it through and no light is visible. Figure 2.7 illustrates the effects of a polarizer.

This property is frequently incorporated in the sunglasses or camera filters to reduce the glare of the sun reflected from a horizontal surface. The reflected p-polarized light goes through a polarizer with a perpendicularly oriented transmission axis which consequently eliminates the incoming light. The effect of a polarizing filter is shown in Figure 2.8.

[2] `https://www.apioptics.com/about-api/resources/visible-light-linear-polarizer/`

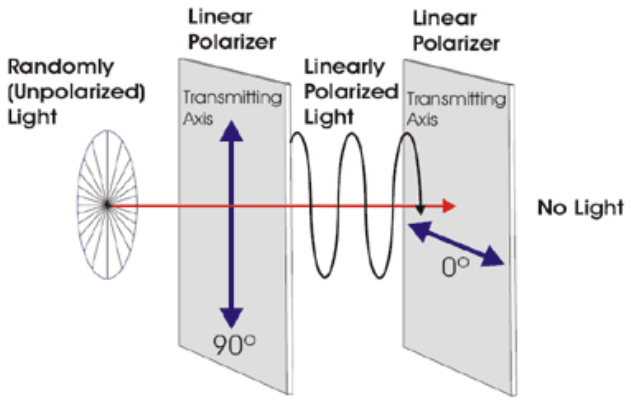


Figure 2.7: Unpolarized light passes through a vertical polarizer → linearly vertically polarized light passes through a horizontal polarizer → no light[2]

Figure 2.8: Polarizing filter by Nikon[3]

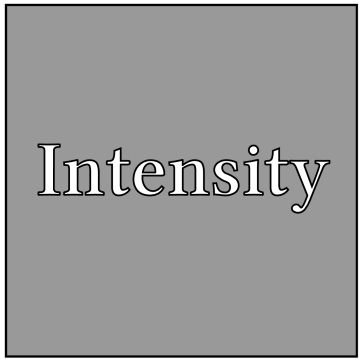
As the polarization is a vast topic, we do not need to go into further detail for the purposes of this thesis. If the reader wishes to learn more, please refer to a scientific literature, for example Kliger and Lewis [17]

2.2.1 Polarization in rendering

The integration of the polarization in the rendering process is quite rare as only a few scenarios display the effects of the polarization and one must implement quite complex behavior of the radiation waves to the spectral rendering. However, renderers such as Mitsuba2 or ART fully track the polarization state of light when needed. The implementation covered by this section is already in Mitsuba2 [4].

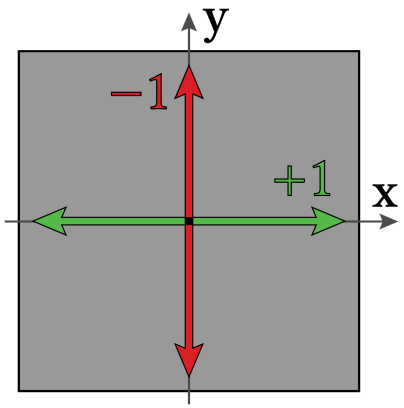
The polarization state is represented by *Stokes vector* — a 4-dimensional quantity that fully parameterizes the elliptical shape of the light's electric field for each wavelength separately. The information stored in the Stokes vectors is explained in Figure 2.9.

[3] `https://www.nikonusa.com/en/learn-and-explore/a/tips-and-techniques/polarizing-filters-add-pow-to-pictures.html`



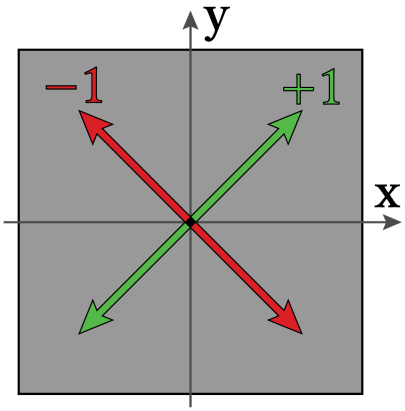
S_0

(1) Radiance - no polarization



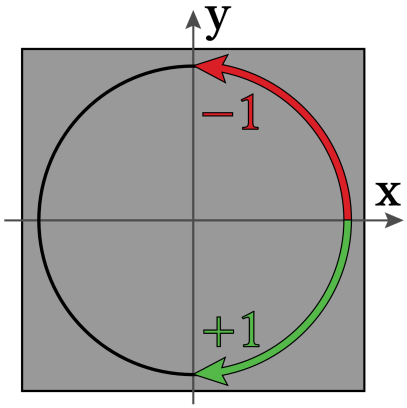
S_1

(2) Horizontal vs. vertical polarization



S_2

(3) Diagonal polarization



S_3

(4) Left vs. right circular polarization

Figure 2.9: Different information carried by the Stokes vector

As we have the polarization states properly represented and we can track them throughout the rendering process, the next step is to determine the affects to these states upon a surface interaction. A transformation between the incoming state and the outgoing state is represented by the *Mueller matrix* $M \in \mathbb{R}^{4x4}$. Due to the adjustments to all interactions in the light transport, we also generalize the BSDF $f_r(\lambda, \omega_i, \omega_o)$ to the polarized pBSDF $M(\lambda, \omega_i, \omega_o)$.

If the reader is curious about the complications this implementation brings and their solutions, he may want to look into the details of Mitsuba2 in Nimier-David et al. [20]. Nevertheless, they are not crucial for the purposes of this thesis and we purposely skip them.

2.3 Dispersion

Generally, the IOR of a dielectric at least slightly varies for different wavelengths (e.g. glass has IOR between 1.5 and 1.6). This causes that the polychromatic light is split into its spectral components upon an intersection with such materials. As each wavelength is slightly shifted, a rainbow effect can be perceived — this phenomenon is called the *dispersion*. In the nature, it can be frequently seen when light passes through liquids (sun through rain). For the scientific purposes, several variations of a device called the dispersive prism 2.10 have been created for this purpose.

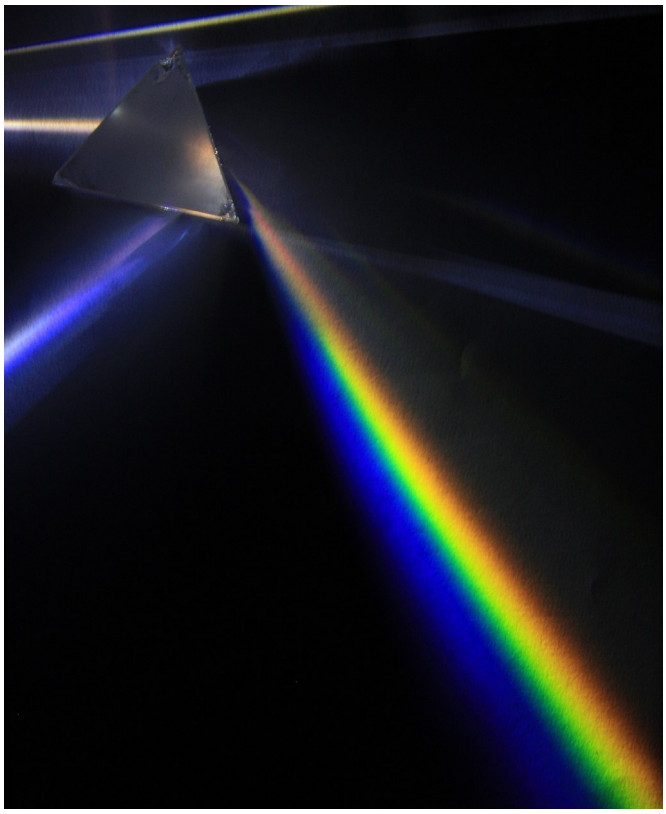


Figure 2.10: Photograph of a dispersive prism[4]

[4] `https://en.wikipedia.org/wiki/Dispersive_prism`

Figure 2.11: Structural iridescence of the peacock feathers (left) and the thin-film light interference in a soap bubble (right)[5]

In the computer graphics, even though it is possible to simulate the dispersion in the tristimulus rendering, it is insufficient and the obvious choice would be to use the spectral rendering as it already contains most of the information about the tracking of the wavelengths.

The missing part is the representation of the varying IOR — currently, the widely used method is called *Sellmeier approximation* [29]. Then, we have to add the ability to track the possibly dispersed monochromatic rays upon a surface interaction of a single polychromatic ray, i.e. create extra samples that were unnecessary before.

2.4 Iridescence

It is quite common that some objects in the nature exhibit an interesting behavior where the hue of their surface graudally changes with the viewing angle and the illumination angle, such as butterfly wings, soap bubbles, oil etc. This phenomenon is called *iridescence* or *goniochromism* and is caused by a large amount of interferences between the light waves and their consequent scattering depending on their wavelength which produces a rapid change in colors [7].

We distinguish two main types of the iridescence:

Microscopic structures Reflections from structures of a size similar to the wavelength (e.g. peacock feathers)

Thin film Light interaction with a thin film of a size similar to the wavelength (e.g. soap bubble)

An example of both can be seen in Figure 2.11.

In this thesis, we focus on the thin-film interference as it is nicely described as a physical process and it is already incorporated in Mitsuba [7]. From now on, by iridescence we mean the thin-film interference until told otherwise.

First, look at the light interactions inside the membrane of a soap bubble in Figure 2.12. The light strikes at the surface of the film, based on the angle it can be either reflected or transmitted. The transmitted light very quickly strikes the bottom boundary of the soap bubble (as it is very thin) and again can be reflected

[5] `https://en.wikipedia.org/wiki/Iridescence`

and/or refracted. As the film is a few hundreds of nanometers thick, this repeats with a great frequency and, as you can see, the light transmitted from the upper boundary can easily interfere with the light reflected from the lower boundary.

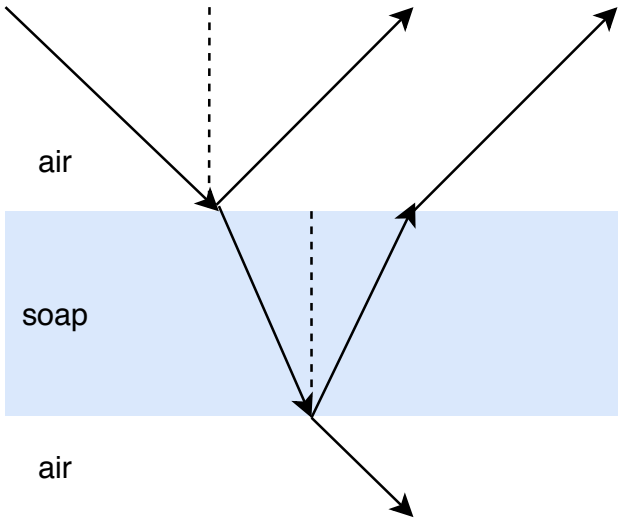


Figure 2.12: A cross-section of a light interaction with a soap bubble

An obvious observation is that the iridescence is also dependent on the thickness of the interacting layer - as the thickness increases, the transmission of the light takes longer time which consequently causes a lot less interferences. The difference between two variously thick films is displayed in Figure 2.13.

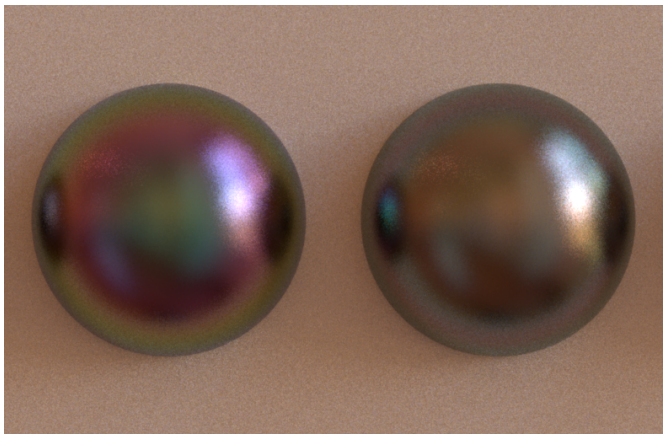


Figure 2.13: Two identical rough conductors with differently thick film layers on top of them: 550nm (left) vs. 1500nm (right) rendered in Mitsuba2

2.4.1 Iridescence in rendering

Based on the Belcour and Barla [7], we overview the computational process of the iridescence caused by a thin film on top of a rough material. We purposely avoid the exact formulations of the equations as these would be unnecessarily complicated to explain and it is sufficient to comprehend the basics in order to

evaluate the correctness of the computation. For more details, the interested reader is referred to the Belcour and Barla [7].

Essentially, this procedure computes an iridescence term of the thin film layer that is plugged into the BSDF of the rough base surface:

1. Compute the reflected and the transmitted values of the Fresnel equations for the IOR of the film and the IOR of the exterior
2. Compute the optical paths differences between the primary and the secondary light paths
3. Evaluate the Fresnel phase shift
4. Determine the term by using the Airy summation for the parallel and perpendicular polarization

2.5 Fluorescence

Curiously, certain materials or substances change their colors with no apparent respect to the illumination color. This behavior is called *fluorescence* and it can be quite commonly observed in the nature, for example in various minerals but also in the living organisms such as fish or arachnids.

The explanation behind this phenomenon is that the molecules of such substances absorb electromagnetic radiation of specific wavelengths and emit back different, usually larger wavelengths. The most eye-catching fluorescence is caused by the absorption of the ultraviolet light which is invisible to the human eye and therefore the fluorescent material might seem to change its color for no reason. An example of a fluorescent calcite is shown in Figure 2.14

Please note that there is a difference between fluorescence, luminescence and phosphorescence:

Luminescence Natural production of light caused by chemical reactions (no absorption)

Phosphorescence Similar to fluorescence but emits light even after the light source is gone

Fluorescence The emission stops almost instantaneously after the light source is gone

2.5.1 Fluorescence in rendering

As we are dealing with the wavelengths of a spectrum, the appropriate decision is to extend the spectral rendering to include the fluorescence. Once again, we refer the interested reader to Mojzík et al. [18] for the implementation details as we are only covering the fundamental ideas for the purposes of this thesis.

As we've mentioned before, we are in fact shifting the wavelengths of the absorbed spectrum to emit a new one — this is called *Stokes shift* (shown in Figure 2.15) and can be described by *fluorescence response* $\Phi(\lambda_i, \lambda_o)$.

[6] `https://www.naturesrainbows.com/single-post/2017/11/01/Fluorescent-Multi-Wave-Calcite-from-the-Elmwood-Mine`

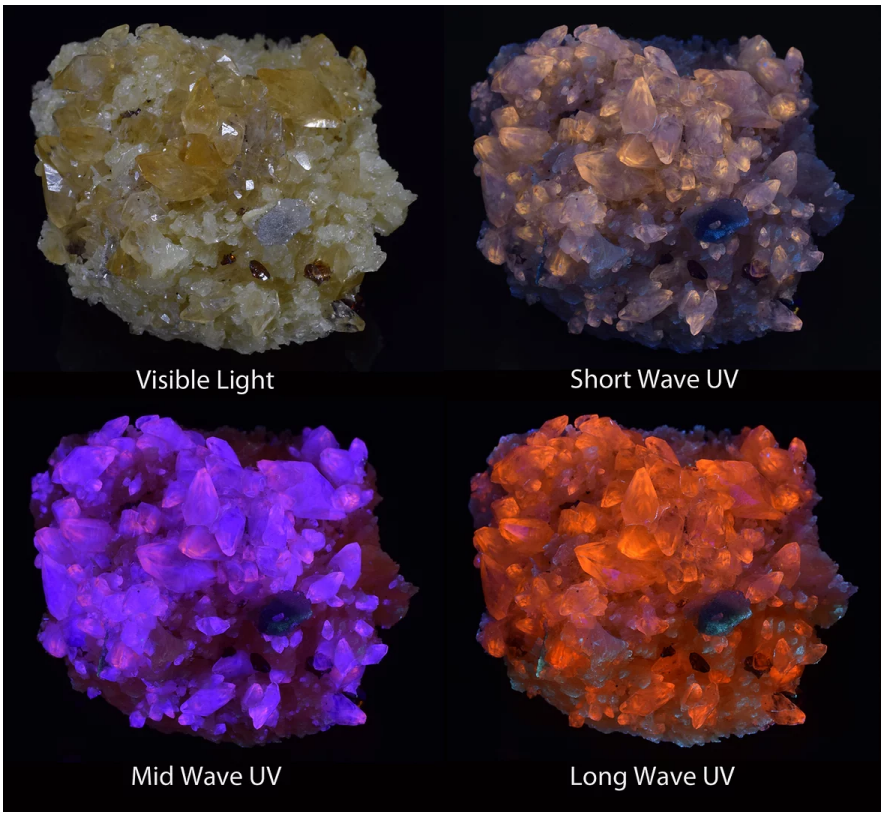


Figure 2.14: Calcite under different lights[6]

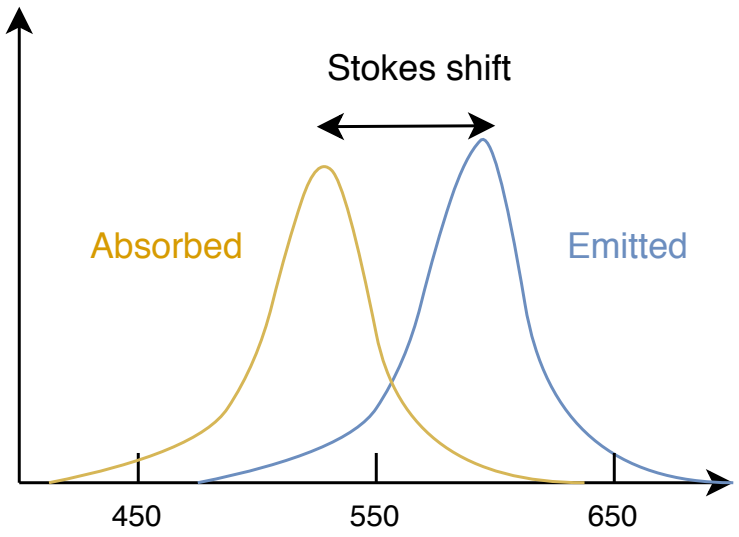


Figure 2.15: Illustration of the Stokes shift

The discrete form of a fluorescence response can be represented by *re-radiation matrix* which contains incoming wavelengths on its vertical axis and their corresponding outgoing wavelengths on its horizontal axis. The probability of the shift follows Kasha's rule — the spectrum distribution of the emitted light should not change, only the intensity of emission spectrum does.

A generalization of the BRDF that includes re-radiation was introduced by Hullin et al. [13] called *Bi-spectral Bidirectional Reflectance and Re-radiation distribution function (bi-spectral BRRDF)*:

$$f_r((\omega_i, \lambda_i), (\omega_o, \lambda_o)) = \frac{d^2 L(\omega_o, \lambda_o)}{L(\omega_i, \lambda_i) d\omega_i d\lambda_i} \tag{2.4}$$

and the corresponding *bi-spectral rendering equation*:

$$L(\omega_o, \lambda_o) = \int_\Lambda \int_\Omega L(\omega_i, \lambda_i) f_r((\omega_i, \lambda_i), (\omega_o, \lambda_o)) d\omega_i d\lambda_i \tag{2.5}$$

The topics of this chapter properly explained the appearance phenomena that we are investigating in this thesis. As we covered the basics of their computations as well, we may now proceed with the evaluation process to determine their accuracy.

3. Benchmark

The main aim of this thesis is to methodically examine the appearance phenomena that are frequently appearing in our day-to-day life but for some reason are still rarely implemented in the modern renderers. However, in this past few years, the need for the physically realistic renders has grown significantly and the implementations for these phenomena have been introduced. As they are still being consistently improved and integrated into modern mainstream renderers, it is absolutely necessary to have a testing suite which would properly evaluate their accuracy.

We propose a testing suite that contains a minimum number of test scenes which maximally exercise these implementations and an equivalent number of the reference images that we consider to be the ground truth, to our best knowledge. These are encapsulated in an automated workflow, which runs the tests with a single command and shows the results in form of a website. The suite also contains the data such as code snippets that can be easily integrated into any standard renderer.

The benchmark follows a few basic principles:

Easy to use The benchmark should provide a user-friendly environment that is comprehensible for an average developer or tester of the rendering features. Therefore, the whole suite is written in Python3 as it is currently one of the most popular scripting languages, it does not need to be compiled and is cross-platform. It also provides CLI command options that are invoke-able via python command.

Modularity Each part of the benchmark should be adjustable without the need to heavily modify the other parts. For example, if a new CLI option is to be added, you only need to change the `/src/arg_parser.py` file.

Extensibility It should be simple enough to extend the capabilities of the benchmark, such as adding new scenes, test case scenarios or even renderers. For example, you don't have to modify any code if you want to add a new scene — there are structures prepared for this scenario which simply need to be filled.

Simplicity The scenes are straightforward, containing only basic and portable geometry, light sources and cameras. This brings two large advantages — it is fairly easy to replicate them for different renderers and they are simple enough to understand the purpose of each element they contain. Along with the thorough comments, anyone with the basic knowledge acquired in the previous chapters of this thesis should comprehend their meaning.

Standalone The benchmark should contain all the data that the potential user would need to properly run or generally use the testing suite. For example, the geometry that is included in the scenes can be found in the `/data/common/` folder.

3.1 Framework

First of all, we take a look into the framework of the benchmark suite and its structure. The file organization is demonstrated in Figure 3.1 and the following sections describe each major subsection of it.

3.1.1 Code

The Python code here simplifies the benchmark process and ensures that the user is working with the benchmark correctly. It has only basic functionalities that generally process the inputs and run the renderer accordingly. We provide a quick overview of each of the Python files:

src/arg_parser.py Parses the CLI arguments and the `settings.json` file and fills its variables accordingly.

src/constants.py Simply contains the constants that are used in other scripts.

src/normalizer.py This script is called after the benchmark is done to normalize the resulting images, as each renderer might have unique naming conventions. It is used only in corner cases.

src/visualizer.py Runs a HTTP server which is required by jeri.io to upload EXR images and opens the website with the results.

benchmark.py The first script that is intended to be directly invoked by the user. Runs other helper scripts mentioned above, his purpose is to actually call the rendering executable for each of the scenes found in `/data/cases/` accordingly to their `configuration.json`.

visualize.py The second script that is intended to be directly invoked by the user. It serves as a wrapper around `src/visualizer.py`.

The choice for Python3 is therefore obvious — for these purposes, there is no need for any high performance or structural design solutions. Thanks to that, we don't have to force the user to compile anything and the benchmark is immediately after the download ready to use.

3.1.2 Cases

The folder `/data/cases/` contains the actual scene descriptions along with their configurations. They follow the structure:

/data/cases/<case name>/<renderer>/scenes+configuration

The benchmark script browses this sctructure, looking for the `configuration.json` file. The purpose of these configurations is that for different scenes we might want to provide different renderer parameters, such as spectral or polarized modes or various definitions.

Each scene is explained in great detail in section 3.3.

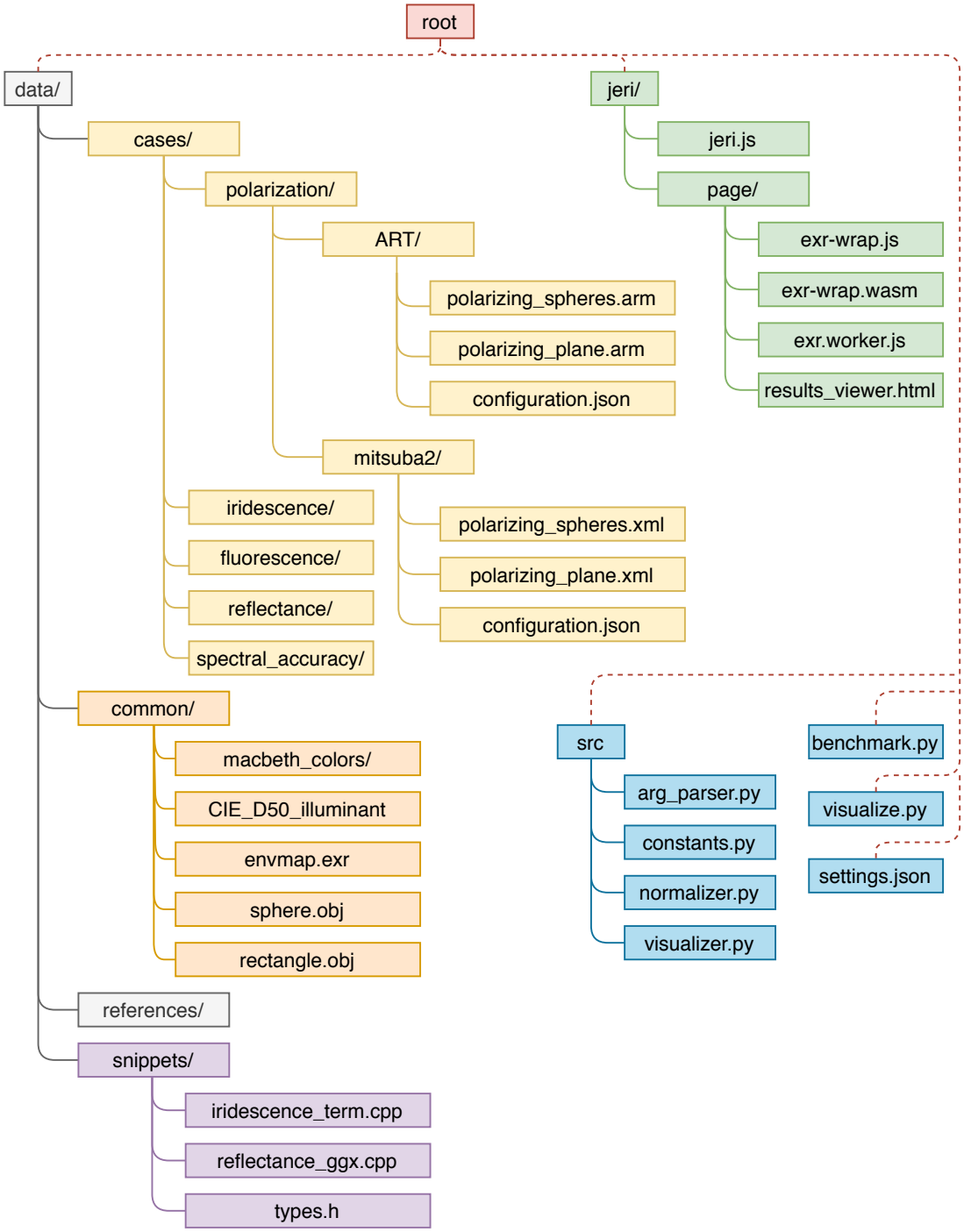


Figure 3.1: File organization of the benchmark

3.1.3 Common data

The folder `/data/common` contains the information and values that are used in the renderers. The built-in definitions of the geometry or the illumination might be unique for each renderer so it is convenient to have such information in a unified form. The folder contains:

macbeth_colors/ Spectral values for all Macbeth colors (24 patch version[1] as defined in ART

CIE_D50_illuminant Spectral values for CIE D50 illuminant [2] rescaled to be used in Mitsuba2

envmap.exr Museum environment map by Bernhard Vogl[2]

sphere.obj/rectangle.obj Unit sphere object and square object with the length of the side equal to 2

3.1.4 Code snippets

Some of the evaluated computations at least partially contribute to the material BSDF, hence it is possible to express them in a generalized form that is easily integrable into any conventional renderer. We decided to provide the code snippets written in C++ (stored in the folder `/data/snippets/`) so that any future user might utilize these to implement them into his own renderer. The folder contains:

iridescence_term.cpp Computation of the iridescence term along with the helper functions, inspired by the code created by Belcour and Barla [7]

reflectance_ggx.cpp Contains the methods for sampling, evaluating and masking according to the GGX reflectance definition [27]

types.h Structures used in the snippets mentioned above

3.1.5 JERI

For the user's convenience, we decided to integrate an EXR visualizer. As it is an extra addition, we decided to use an existing one instead of creating our own.

JERI (or Javascript Extended-Range Image) is an EXR viewer written in JavaScript developed by Disney Research [3]. It is simple to use and to integrate and provides many features over the images such as zoom, change of exposure and automatic error maps.

We use JERI to display the results of the whole benchmark to the user on a single website. The user may look at the results, the reference images and even at their differences (compared by L2 and MAPE error maps). A screenshot of the results website is shown in Figure 3.2.

[1] `https://xritephoto.com/ph_product_overview.aspx/?id=1192&catid=28`
[2] `http://dativ.at/lightprobes/`

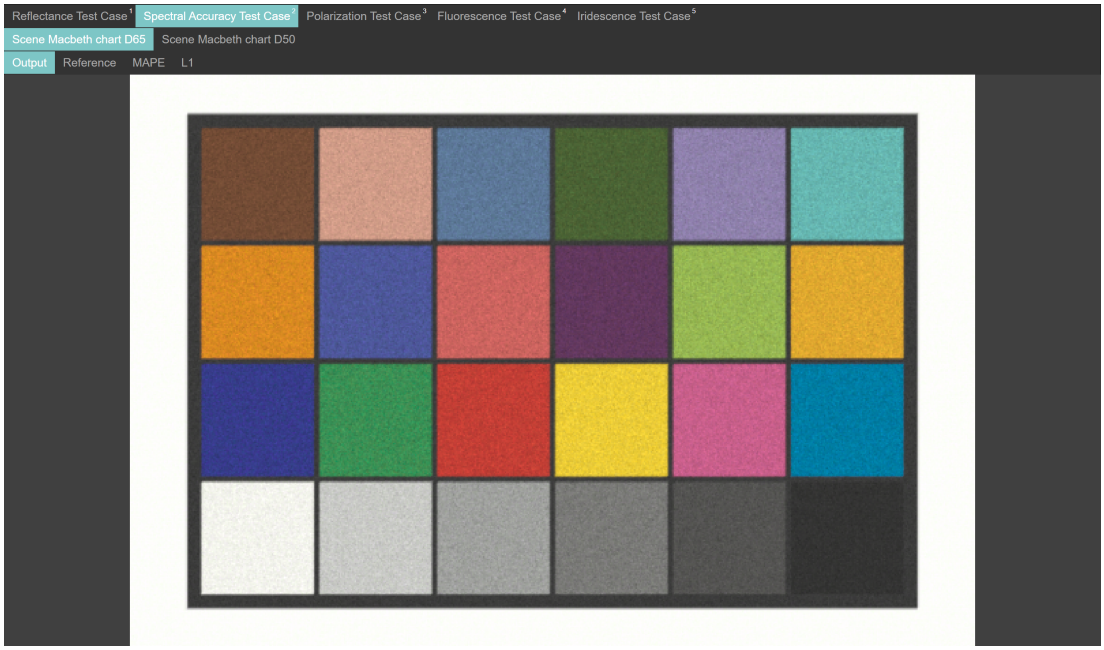


Figure 3.2: Results website

3.2 Supported Spectral Renderers

In the current state, the benchmark supports two renderers — *Mitsuba2* and *ART*. Both are physically-based, research-oriented, they are capable of representing the light in spectral domain and tracking the polarization states. These features make them suitable candidates for our purposes as we are evaluating the visual correctness of the physically-described appearance phenomena, including spectral accuracy and polarization. The following two sections contain an overview of these renderers.

3.2.1 Mitsuba2

Mitsuba2 has been released only recently (paper was published in November 2019) as a successor to a well-known, research oriented renderer Mitsuba 0.6. Rather than an upgrade, Mitsuba2 is a complete overhaul of its predecessor, incorporating the latest trend in programming. It is very well documented in mit [4] and Nimier-David et al. [20] and can be downloaded/cloned from `https://github.com/mitsuba-renderer/mitsuba2`.

It is written in C++17 and is designed to be modular — it contains a large number of various plugins where each adds a new functionality to the Mitsuba2 rendering process, such as:

Materials BSDFs for rough/smooth dielectrics, conductors, plastic, etc.

Light sources Uniform, spot, point, area, environment

Shapes Imported obj, ply but also built-in such as sphere

Integrators Direct illumination, path tracer, stokes integrator, etc.

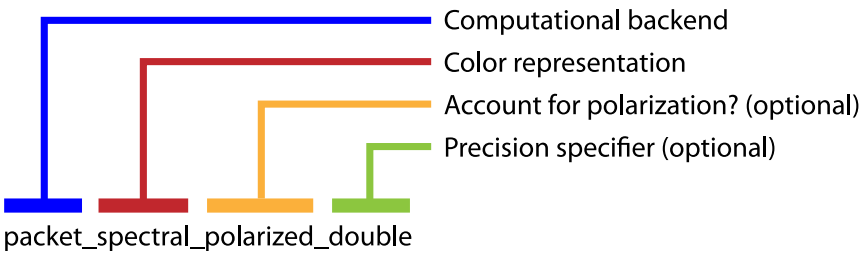


Figure 3.3: Different variants of Mitsuba2

and many more.

It can be compiled into several variants — from RGB CPU rendering to differentiable GPU spectral rendering that tracks polarization. These options are shown in Figure 3.3. The important part is that the renderer is retargatable which means that the user may choose the specific options setup and Mitsuba2 uses the appropriate internal representation of its data. For example, for RGB rendering, the color is obviously represented by an array of 3 floats. For spectral rendering, the array contains 4 floats which represent the spectral wavelengths and a stochastic approach is used to sample these wavelengths. This is possible thanks to the template metaprogramming that C++ offers.

Mitsuba2 also provides an extensive API for Python so that almost all functions may be used from within a code.

Anyone can contribute to the project via pull request as it is open source.

3.2.2 ART

Advanced Rendering Toolking, or shortly ART, is physically-based, research oriented rendering framework that has been developed by the Computer Graphics Group on Charles University in Prague, Faculty of Mathematics and Physics. In the past, there have been important contributions by people at the Institute of Computer Graphics in Vienna. However, as ART is currently at version 2.0.3, most of their work has not been ported from version 0.x/1.x. ART can be download/cloned from `git://cgg.mff.cuni.cz/ART.git` and this section is largely based on the it's documentation [1].

ART is written in Objective-C and therefore compilable on the majority of the modern operating systems. The scenes are written in a custom language that supports procedural modeling of the scene objects, such as loops and conditions. Also, the immediate results of the rendering process are custom spectral images which need to be tonemapped (included in the project) afterwards

On top of the standard features that most conventional renderers offer, such BSDFs, light sources, camera, path tracing, etc., ART implements several rare, or even unique ones, such as:

Sprectral rendering Uses Hero wavelengths spectral sampling

Fluorescence Supports fluorescent materials, volumes and even illuminants

Polarization Capable of tracking the full polarization state

The whole project includes multiple tools such as tonemapper which offers several tools for adjusting and converting the spectral images or polarization visualizer.

As ART is an open source project under GNU v3.0 license, anyone can download and use it.

3.3 Scenes

3.3.1 GGX Reflectance

3.3.2 Spectral accuracy

3.3.3 Polarization

3.3.4 Fluorescence

3.3.5 Iridescence

3.4 Usage

The user may clone/download the benchmark from the github repository of this thesis[3]. As the whole benchmark is written in Python3, there is no need to compile the project. Then, it is necessary to modify `settingsjson` file in the root directory of the benchmark suite — the user must specify the path to the renderer executable and the name of the renderer (currently, only `mitsuba2`) and `ART` are viable options).

That is all for the preparations and the benchmark can be run with simply running the `benchmark.py` python file in the CLI. Be aware that the script must be run from within the root directory of the benchmark suite! The following commands will get you to the root directory and run the benchmark.

```
cd /render_benchmark
python3 ./benchmark.py
```

As soon as the benchmark ends, the user may find the rendered EXR images in a folder named `outputs-yyymmdd-hhmmss` where the `yyymmdd-hhmmss` is substituted for the date and time when the command was issued.

Then, a user may use a second script `visualize.py` that opens the website with the results of the benchmark, the reference images and their difference images. This script must also be run from without the root directory of the benchmark suite.

The benchmark accepts several parameters:

--help (-h) Only outputs the help message

--scene (-s) <scene_name> Test only the scene called scene_name

--case (-c) <test_case_name> Test only the test case called test_case_name

--renderer (-r) [ART/mitsuba2] Specify the name of the renderer

[3] `https://github.com/marcel1hruska/render_benchmark`

--exec (-e) <path> Specify the path to the renderer executable

--log (-l) Write all outputs to the log file (in the outputs folder)instead of the console

--visualize (-v) Visualize the outputs immediately after the benchmark ends

For convenience, all of these options may be specified in the `settings.json` file but the ones defined in the CLI override them.

In the following sections, we demonstrate the basic uses cases of our testing suite using a fictional persona called Frodo.

3.4.1 Use case Regress test

Frodo is a mitsuba2 developer who recently changed the sampling strategies of the spectral rendering. However, he is not sure whether he broke some of the functionalities. He sets the benchmark for mitsuba2 and provides it his latest executable. He runs all test case scenarios and compares the results with the reference images.

3.4.2 Use case New feature

Frodo is an ART developer who would like to add the GGX microfacet distribution to ART as it is not currently supported in the latest version. He finds the code snippets that are attached to the benchmark suite and integrates them to ART.

Then, he looks up the scenes that test GGX reflectance created for mitsuba2 and, as these contain only a very simple a straightforward geometry, he duplicates them. He saves them in a folder `data/cases/reflectance/ART/` and creates `configuration.json` file in the same folder that only specificies the name of the resulting image, the filename of the test scene and the parameters that are to be passed to the rendered (examples can be seen in different ART scenarios).

He then provides the executable to the benchmark suite, sets it to ART and runs it. The benchmark automatically detects the new configuration for ART. Either from the reference images or from the difference images, he may determine some irregularities that are caused by his incorrect implementation of the masking function. As soon as he corrects it, he sees that the results are coherent.

3.4.3 Use case New scenario

Frodo is a mitsuba2 developer and he just added support for dispersion. As the dispersion has never been tested before, he simply adds a new folder to the benchmark suite `/data/cases/dispersion/mitsuba2` and, as in the previous case, he creates the `configuration.json` for it.

Now, he must add the keyword `dispersion` to the string array `TEXT_CASES` that can be found in the file `/src/constants.py`.

From now on, the benchmark contains the dispersion tests as well. If he wants to add the reference images as well, he may simply add them to the `/data/references` folder.

In case he also wants to add the new scenario to the visualizer, he needs to extend the file `/jeri/page/results_viewer.html`. But, as the HTML page contains a simple JSON structure for the elements that it should contain, it is fairly easy to do so.

3.4.4 Use case Improved feature

Frodo is an ART developer who reworked the fluorescent materials and would like to see if the new implementation improved them.

The test scenes may be considered as simple templates with whom we created the reference images. Frodo simply adjusts the test scenes in the fluorescence test case scenario. If he finds out that the results indeed improved, he may simply replace the existing reference images and keep the adjusted scenes.

3.4.5 Use case New renderer

Frodo has created his own spectral renderer the Ring that supports all the test case scenarios in the benchmark suite and he wants to evaluate the correctness of his implementations.

Therefore, he duplicates the template scenes from other renderers accordingly to his own scene format. He creates new folders in each test case scenarios in the `/data/cases/` folder, puts his scenes inside accordingly and create the `configuration.json` for each of them.

Then, he adds support for his renderer — in `src/constants.py`, he adds the `ring` keyword to the string array `RENDERERS`.

Now, he can set the benchmark to evaluate the new ring renderer and provides his executable. The benchmark pipeline as well as the visualization is done for it automatically.

3.5 Open source contributions

During our work on the benchmark, we have done several noteworthy contributions to three open source projects. Note that these extensions can be considered as byproducts and definitely not the main aim of this thesis — therefore, they are not yet in a form a pull request as this process requires a significant amount of time.

3.5.1 GGX for ART

The use case described in subsection 3.4.2 actually happened to us (not to Frodo). As a part of the work on the benchmark, we've decided to add the GGX distribution to the ART renderer and, coincidentally, it nicely correlated with the mentioned scenario. We had the scenes and the reference images prepared for mitsuba2, we simply replicated them for ART and implemented the GGX. Then, we simply iterated the benchmark and the adjustments in the code over and over until the results were satisfactory.

The implementation is attached to the thesis as GGX_ART.

3.5.2 Iridescence for Mitsuba2

Mitsuba does not have a native support for the iridescence but an external plugin has been developed to simulate the iridescent effects of a thin film dielectric layer on top of a rough conductor base. Unfortunately, it was created for Mitsuba version 0.6 and, as Mitsuba2 is fairly new, there has not been an effort to rework the plugin, thus we took the initiative and re-implemented it.

The whole work is based on the Belcour and Barla [7] and the supplementary code for Mitsuba 0.6 released along with it. There were some major changes to the spectral sampling strategy and the overall object structure done in Mitsuba2 that had to be adapted in the new, re-implemented version of the plugin.

The correctness of the rework has been confirmed in a similar manner to the normal benchmark workflow. We prepared the test case scenerio scenes for the iridescence, rendered them for both Mitsuba2 and Mitsuba0.6 and considered the latter version to be the ground truth. The implementation can be found on `https://github.com/marcel1hruska/mitsuba2`.

3.5.3 Multi-channel EXR support for jeri.io

While designing the polarization test scenes, we've encountered a compatibility issue between the Stokes vector output format of Mitsuba2 and ART. While ART stores the Stokes vector values into distinct EXR images, Mitsuba2 creates a single multi-channel EXR where each channel contains the Stokes vector information. As you can imagine, the visualization of the rendering results requires a custom solution — it consists of two parts. We've added a support for multi-channel EXR images to the jeri.io as there is currently no way to visualize them. It works as follows:

1. If the EXR image contains multiple channels, store them in a extra map `otherChannels` which connects channel name with its contents.

2. The user may specify the channel to display in the viewer data.

3. If the specified channel is in the map, display the wanted contents

The second part puts one more layer on top of that and highly custom for our purposes of resolving the incompatibility itself. We assume the format of ART — e.g. the file named `sphere.s0.exr` means that it is the output of the first channel of the Stokes vector. Our script in jeri.io tests whether this file really exists — if not, we assume from the name of the file that the user actually wants the channel `S0` of the file `sphere.exr` and therefore we switch to that.

This behavior is transparent to the user. If the file really does not exist, the jeri.io fails to find as any other file and displays nothing.

This addition is only a part of the compiled JavaScript code of the jeri.io. It can be found in the benchmark's file `/jeri/exr-worker.js` and `/jeri/jeri.js` between the lines commented as `multichannel custom support`.

3.6 Future Work

Conclusion

Bibliography

[1] Art documentation. `https://cgg.mff.cuni.cz/ART/assets/ART_Handbook.pdf`. Accessed: 2020-20-7.

[2] Cie data. `https://law.resource.org/pub/us/cfr/ibr/003/cie.15.2004.tables.xls`. Accessed: 2020-19-7.

[3] Jeri web. `https://jeri.io/index.html`. Accessed: 2020-19-7.

[4] Mitsuba2 docs. `https://mitsuba2.readthedocs.io/en/latest/`. Accessed: 2020-6-7.

[5] LE Barbrow. International lighting vocabulary. *Journal of the SMPTE*, 73 (4):331–332, 1964.

[6] Petr Beckmann and Andre Spizzichino. The scattering of electromagnetic waves from rough surfaces. *ah*, 1987.

[7] Laurent Belcour and Pascal Barla. A practical extension to microfacet theory for the modeling of varying iridescence. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[8] David Brewster. On the laws which regulate the polarisation of light by reflexion from transparent bodies. *Philosophical Transactions of the Royal Society of London*, 105:125–159, 1815.

[9] Russel E Caflisch et al. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 1998:1–49, 1998.

[10] Robert L Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics (ToG)*, 1(1):7–24, 1982.

[11] Bernardt Duvenhage, Kadi Bouatouch, and Derrick G Kourie. Numerical verification of bidirectional reflectance distribution functions for physical plausibility. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, pages 200–208, 2013.

[12] Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989.

[13] Matthias B Hullin, Johannes Hanika, Boris Ajdin, Hans-Peter Seidel, Jan Kautz, and Hendrik PA Lensch. Acquisition and analysis of bispectral bidirectional reflectance and reradiation distribution functions. In *ACM SIGGRAPH 2010 papers*, pages 1–7. 2010.

[14] Wenzel Jakob and Johannes Hanika. A low-dimensional function space for efficient spectral upsampling. In *Computer Graphics Forum*, volume 38, pages 147–155. Wiley Online Library, 2019.

[15] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. AK Peters/CRC Press, 2001.

[16] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.

[17] David S Kliger and James W Lewis. *Polarized light in optics and spectroscopy*. Elsevier, 2012.

[18] Michal Mojzík, Alban Fichet, and Alexander Wilkie. Handling fluorescence in a uni-directional spectral path tracer. In *Computer Graphics Forum*, volume 37, pages 77–94. Wiley Online Library, 2018.

[19] Fred E Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied optics*, 4(7):767–775, 1965.

[20] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6):1–17, 2019.

[21] Michael Oren and Shree K Nayar. Generalization of lambert's reflectance model. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 239–246, 1994.

[22] Mark S Peercy. Linear color representations for full speed spectral rendering. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 191–198, 1993.

[23] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[24] Francois X Sillion and Claude Peuch. Radiosity & global illumination. 1994.

[25] DH Sliney. What is light? the visible spectrum and beyond. *Eye*, 30(2): 222–229, 2016.

[26] Kenneth E. Torrance and Ephraim M. Sparrow. Theory for off-specular reflection from roughened surfaces. 1967.

[27] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007.

[28] Wilkie. Cgg course: Introduction to the color science. `https://cgg.mff.cuni.cz/~wilkie/Website/NPGR025.html`. Accessed: 2020-11-7.

[29] Kate Devlin1 Alan Chalmers1 Alexander Wilkie and Werner Purgathofer. Tone reproduction and physically based spectral rendering. Eurographics, 2002.

[30] Gunter Wyszecki and Walter Stanley Stiles. *Color science*, volume 8. Wiley New York, 1982.