**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

I would like to express my sincere gratitude to my supervisor doc. Alexander Wilkie, Dr., for all the patience, help and advice he has given me.

I want to thank my girlfriend, my family and my friends for their constant support, especially during the last half year.

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: Various rendering techniques often use different approaches to the same aspects of the image synthesis process, mainly due to their complexity and constant development. Excluding global illumination algorithms, appearance descriptions are key distinguishing factors between the rendering systems. These descriptions might include BRDF models, support for spectral color representation, and even integration of advanced phenomena, such as fluorescence. Unfortunately, as there are no standardized implementations of these features, their computations might not be completely accurate, which may result in their incorrect representation.

This thesis describes an evaluation suite that methodically tests rendering algorithms based on their appearance reproduction capabilities. The core of the suite is a set of scenes that test five specific appearance phenomena — polarization, GGX reflectance, fluorescence, iridescence and the overall spectral accuracy. Each test case scenario contains as few scenes as possible while maximizing the number of covered aspects of the tested feature. For the user's convenience, we wrap the scenes inside an automatic workflow that runs the specified test case scenarios and displays the results. As a correctness metric, we provide manually verified reference images that are considered to be the ground truth. The user may compare the benchmark results with the reference images by, for example, observing provided error maps.

Keywords: appearance evaluation,rendering evaluation,spectral rendering

Contents

Introduction

Studies of nature have always been an important part of human history. Common phenomena such as rainbows or light reflections have been one of the primary topics discussed in the scientific circles which, once they were properly explained, led to the formulations of their descriptions.

In the modern era of computers, many of these phenomena are well-defined and can be accurately represented by physical models. One of the ultimate interests of the computer graphics is to replicate these sensations, creating realistic images that would be indistinguishable from a photograph.

Although the publications discussing the rendering process were released mostly in the second half of the 19th century [15][20], it is only now that we are capable of computing certain specific aspects of the natural light. Mostly thanks to more powerful hardware, correctly simulating the light transport while evaluating non-trivial appearance models can be done in a matter of minutes.

In the past, the physical realism of image synthesis was mostly present in research-oriented systems only. The commercial world used to approximate the computations, as they were aiming for visually appealing images instead of realistic ones. Recently, the interest in physical realism has also been rising in the entertainment industries and more advanced light transport simulations are becoming a standard, opening possibilities to reproduce appearances more accurately.

However, as many rendering techniques are constantly being improved, various rendering systems contain custom, sometimes even distinct, implementations. In addition to light transport techniques, there are many other distinguishing key features, such as material models or the internal light representation. Even though large parts of the rendering systems present obvious similarities, there is no standardized implementation of any of these features and so their computations may vary in terms of accuracy.

Unfortunately, due to these dissimilarities, there is no unified evaluation process that would assess the correctness of a specific technique or the whole rendering system. In fact, whenever an improved or a completely new technique is introduced, the creators present their results on their own set of scenes. Even involuntarily, these scenes might not properly exercise the techniques, possibly leaving some inaccuracies unexposed. Therefore, there is a need for a standardized way to test the rendering systems and their features. We provide a solution that methodically evaluates the computations of several appearance phenomena.

Goals

The main goal of this thesis is to introduce a set of scenes that test various rendering systems based on their appearance reproduction capabilities. These scenes are specifically designed to expose potential differences between the computation of the specific phenomenon and its defined behavior in nature. While it is possible to test the light transport simulations, we focus on the specific visual sensations that are, to this date, being developed and discussed, such as fluorescence, dispersion, and polarization. As we desire to simulate these evaluated phenomena according to their physical descriptions and without any lossy conversions from the RGB domain, the spectral internal representation of light is an absolute necessity.

For the user's convenience, we wrap the test scenes in an automated workflow and we provide the reference images that we consider to be the ground truth.

Thesis organization

This thesis is structured as follows:

Chapter 1 introduces the reader to color science and computer graphics. It explains the fundamental basics of the rendering process to comprehend more advanced computational models that are discussed later. It also defines the terminology that is widely used throughout the thesis.

Chapter 2 continues in discussing the specific appearance phenomena that are the main interest of this thesis, providing in-depth explanations as well as exact implementations for each of them.

Chapter 3 shows the actual output of the thesis — the evaluation suite. We describe the framework and justify its design by demonstrating possible use case scenarios. We also explain individual test scenes and the methodology that we used to add them.

Chapter 4 directly compares two different renderers based on their computation implementations using the results from the benchmark.

At the end of the thesis, we provide a user guide which clarifies how to run and use the benchmark.

1. Rendering and Color Science

This chapter serves as an introduction to computer graphics and 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 the publications by Wyszecki and Stiles [37], Wilkie [35], Nimier-David et al. [21] and Pharr et al. [27].

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

1.1 Light

According to the definition by Barbrow [1], the light is "any radiation capable of causing a visual sensation directly". In other words, a visible light is electromagnetic radiation that is perceivable by human eye.

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

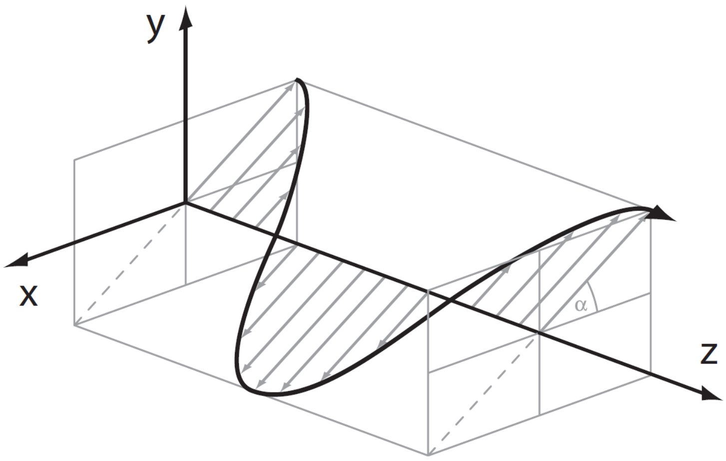


Figure 1.1: A propagation of wave [35]

Usually, by the term light, we mean the visible 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. The lower boundary is estimated between 360 and 400nm and the upper boundary from 760 to 830nm [30]. The light above this range is called infrared light and below ultraviolet. An explanatory image of the known electromagnetic wavelengths can be found in Figure 1.2.

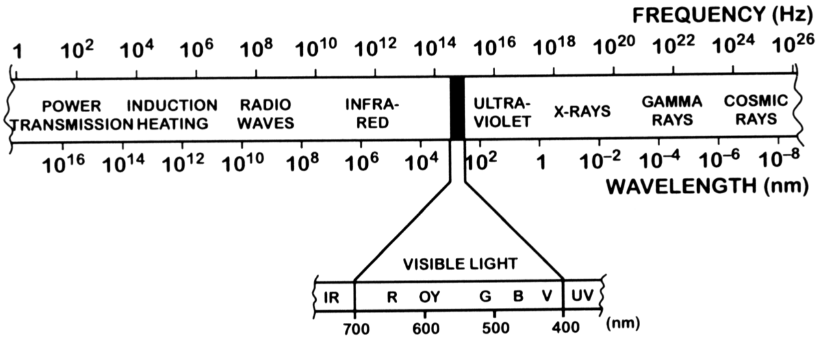


Figure 1.2: An image displaying various wavelengths [35]

1.1.1 Color

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

To categorize the colors, several reproducible representations were formed, 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 eyes' 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.

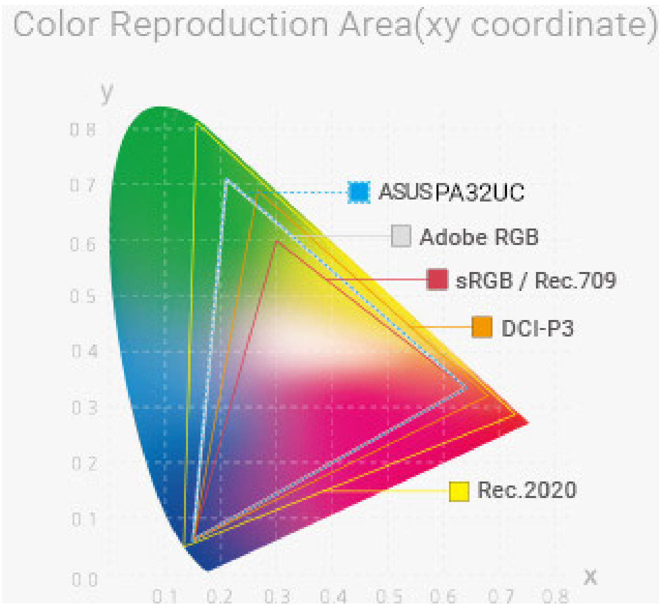


Figure 1.3: An illustrative comparison between five RGB gamuts by Asus [1]

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`

color matching experiments to obtain 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 a spectrum to an RGB color space looks as follows:

1. Compute the tristimulus 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. The matrix differs depending on the specific RGB color space — an example of CIE XYZ to sRGB conversion:

$$\begin{aligned} r &= 3.240X - 0.969Y + 0.55Z \\ g &= -1.537X + 1.875Y - 0.204Z \\ b &= -0.498 + 0.041Y + 1.057Z \end{aligned}$$

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 sets of measurement units were developed to quantify the light — *photometry* and *radiometry*. Radiometry recognizes the light as an electromagnetic radiation while photometry focuses on the human perception of the light. Despite the distinct purposes, their quantities are often easily convertible. The following table shows some of their basic quantities:

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 $[candela.m^{-2}]$

Each of these quantities is briefly mentioned in the following description:

[2] The (X,Y,Z) data are taken from `https://www.waveformlighting.com/tech/color-matching-function-x-y-z-values-by-wavelength-csv-excel-format`

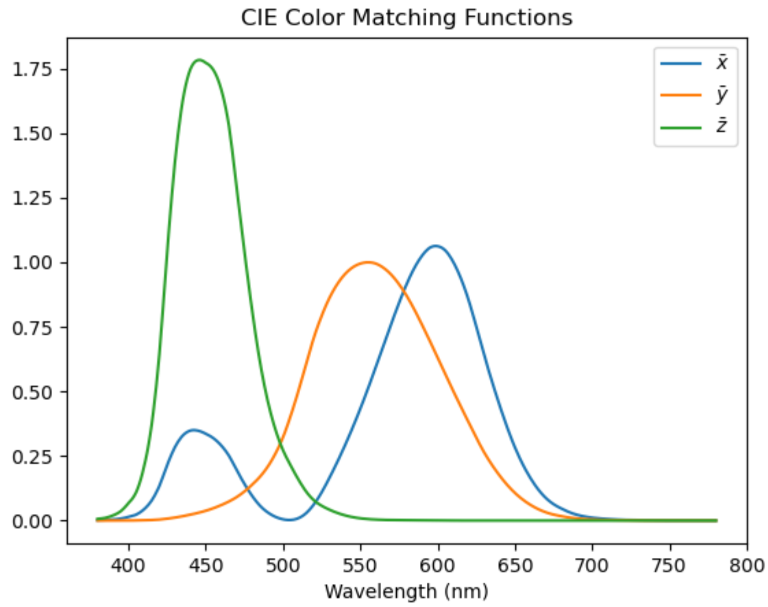


Figure 1.4: Color matching functions plotted in Python[2]

Spectral radiant energy Amount of light energy at a specific wavelength

Radiant flux Amount of light energy with respect to time

Irradiance Flux at a specific point (space)

Radiant intensity Flux in a direction (steradian, shortly *sr*, is a unit of the solid angle — a surface on a unit sphere, the 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, implying that some wavelengths are detected more easily. As we can see in Figure 1.5, the scotopic (night) perception peaks at around 507nm and the photopic (day) perception at 555nm.

1.2 Physically based rendering

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

The main job of a rendering system (*renderer*) is to simulate various phenomena commonly seen in nature, such as light reflections, refractions, shadows, etc, accurately to their physical models. Nowadays, modern renderers are capable of recreating the scenes so authentically that the rendered images are almost identical to real-life photos. An example can be seen in Figure 1.6.

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

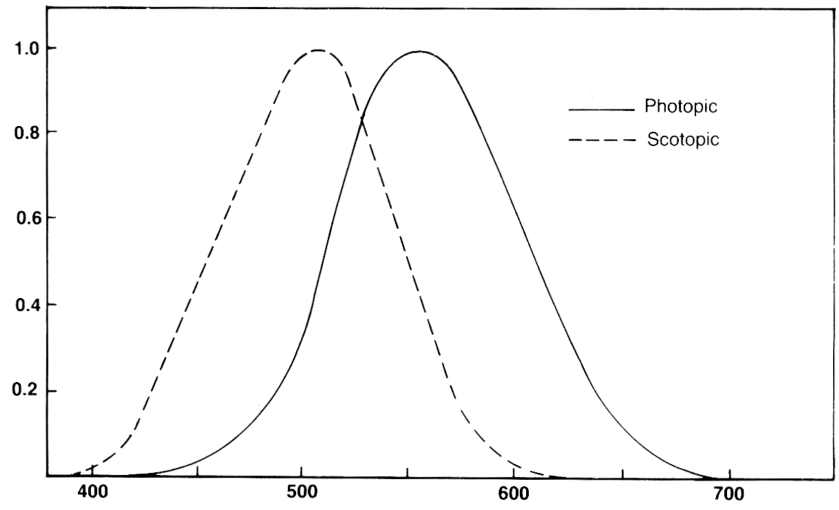


Figure 1.5: Relative luminous efficiency function [35]

The rendering workflow 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, an image ("photograph") of the scene called the *render* is created

This process is further explained in subsection 1.2.5.

1.2.1 Digital scene

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

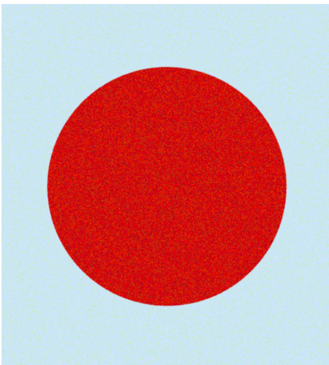
Camera A camera (or a sensor) in a digital scene works in the same manner as in real life — it records a picture. Generally, it is possible to define the coordinate position and the viewing vectors but also the properties such as focal distance or the type of the film.

Light source The scene needs to be illuminated by at least on light source to be visible. The common types of lights are point light, area light, spotlight, and environment (constant) light.

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

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

Unfortunately, as each renderer may have unique implementation details, the formats of the scenes are vastly different. For example, Mitsuba uses XML, but PBRT has its specific format. An example of a simple scene for Mitsuba2 can be found in Figure 1.7.



```
<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 1.7: A simple scene rendered in Mitsuba2 along with its description

Once the scene is described, the renderer runs a light transport simulation inside it to determine the colors of the final image. As this is a fundamental part of the rendering process, the following sections describe the physics theory and the models behind the light transport and the materials.

1.2.2 BRDF

The reflective properties of a material are described by *Bidirectional Distribution Reflectance Function*, shortly *BRDF* [20]. Its equation looks as follows:

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

Essentially, it states how much radiance is reflected from the incoming direction ω_i ($L_i(\omega_i)$) to the outgoing direction ω_o ($L_o(\omega_o)$) for a specific material. An image interpretation of the function is in Figure 1.8. As it is a distribution function, we can also reformulate its meaning as a probability density that a defined amount of light energy gets reflected from ω_i to ω_o.

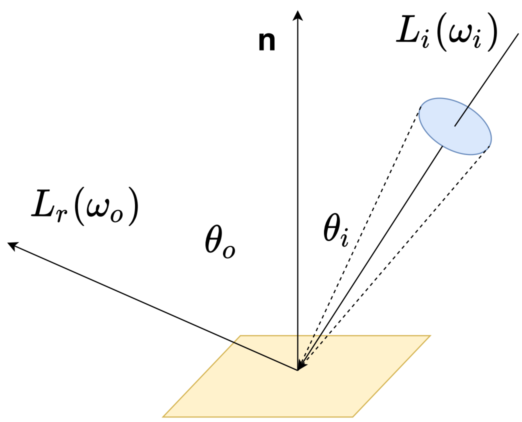


Figure 1.8: Bidirectional Distribution Reflectance Function

Renders of a diffuse, a glossy, and a mirror material are compared in Figure 1.9.

Physically-based BRDFs must fulfill the following properties 9:

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

Energy conservation The amount of reflected energy cannot be greater 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 the 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 (e.g. skin)

In this thesis, most of the materials that we talk about are described by BSDFs.

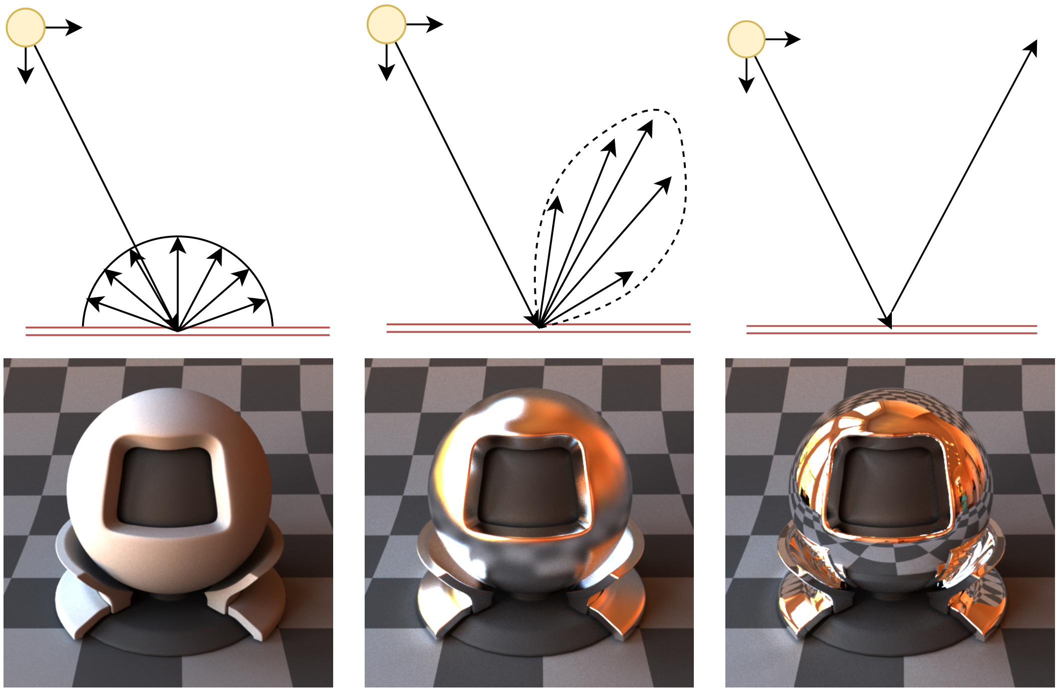


Figure 1.9: A preview of a diffuse (left), a glossy (middle) and a mirror (right) material rendered in Mitsuba2 along with their illustrative BRDF visualizations

1.2.3 Global Illumination

With the BRDF defined, we can now formulate an equation that evaluates the global illumination of a scene, which is the illumination of each point from all light sources. It is generally called the *rendering equation* [15] and it can be computed as follows:

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

, where:

x is the currently computed point in the scene.

L_o is the outgoing radiance.

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

L_r is also called the *reflectance equation* and it computes 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 is denoted 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.

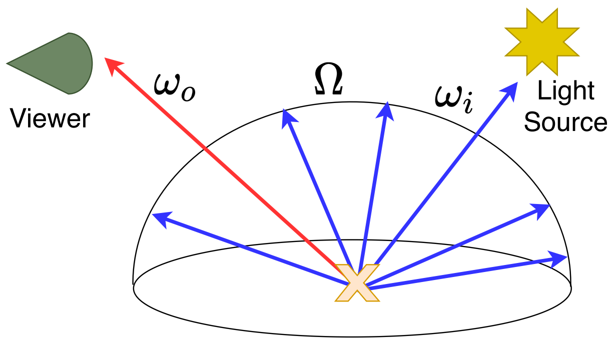


Figure 1.10: Reflectance Equation

An image interpretation of the reflectance equation can be seen in Figure 1.10.

Every 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 contributing 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 — *Monte Carlo integration* [5].

Formally, for a multidimensional definite integral

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

Monte Carlo (MC) estimates I as

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

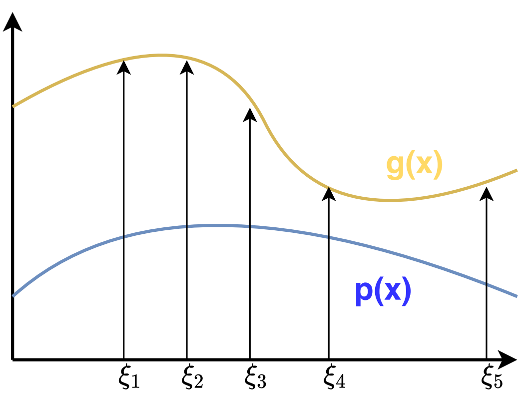


Figure 1.11: Monte Carlo method 2D visualization

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, 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, importance sampling ensures that if some samples are generated twice as much, their weight is decreased by 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 its own benefits. The one that we mention the most in this thesis is called *path tracing*. Its core idea is simple:

1. For each pixel in the image plane, shoot a primary ray r from the 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 generates a direction uniformly over a hemisphere while glossy BRDF prioritizes samples from the reflectance lobe (shown in Figure 1.9).

   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 A user specified maximum number of recursions.

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

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

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

3. If r hits an emitting object (light source), add its 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 correlates with the recursive nature of the rendering equation. Even though the path tracing is a slow algorithm (i.e. not suitable for real-time rendering in games), its variations can be extremely accurate. Providing equally accurate surface models used in the scene, the resulting images might even be indistinguishable from real photographs.

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 the consequent walk to the light source. Ideally, each walk should end by hitting a light source, which depends on the number of samples and the maximum allowed depth of the recursion. Consequently, this creates variance which can be easily improved by the integration of the *multiple importance sampling* (MIS). Generally, it involves a weighted combination of multiple sampling techniques. Typically, it combines 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 point's value. Both sampling methods are, obviously, weighted to avoid over-illumination.

Volumes Another aspect that needs to be accounted for in the rendering process are volumetric objects, such as fogs or smokes. Typically, there are two ways that a volumetric object may affect the light passing through it. Firstly, the volume can attenuate the light by absorbing it or scattering it to different directions. Secondly, the volume can also strengthen the light by its own emission (e.g. flame) or scattering light from different directions to the sampled one.

A single walk of a path tracer capable of volume tracing and MIS sampling is visualized in Figure 1.12.

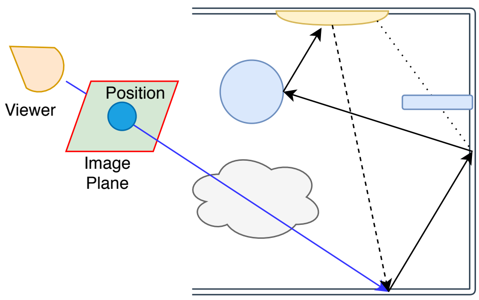
Other methods

Path tracing is our main concern in this thesis due to its physical and unbiased properties. Some of the other global illumination techniques are mentioned only briefly in the following description:

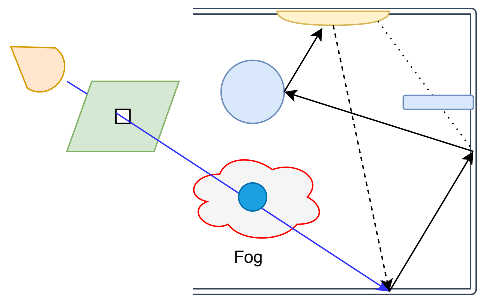
Ray tracing [10] Similar to path tracing, but there are no bounces from the surfaces in general, it simulates only reflections, refractions, scattering, etc. Nowadays, real-time rendering is capable of ray tracing.

Photon mapping [14] Two rays traced independently — one from the camera, the other from the light source — until termination occurs. The resulting radiance is computed based on their final positions. Faster in some scenarios but biased (does not have to converge to a correct solution).

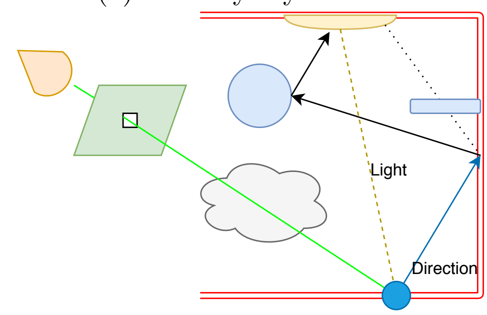
Radiosity [29] Uses the finite element method instead of Monte Carlo. View independent, the light is traced from the source and (possibly) bounced to the viewer. Good for precomputations.



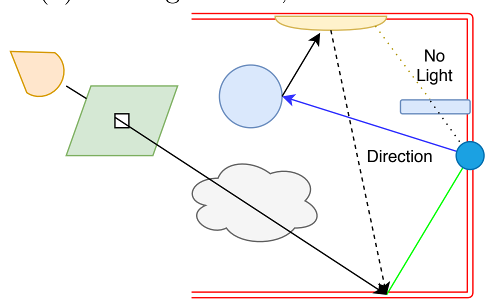
(1) Primary ray created



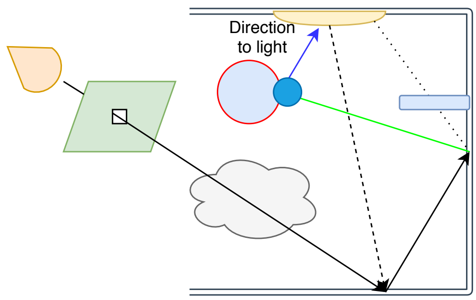
(2) Passing volume, attenuation



(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 1.12: A visualization of a single walk in the path tracer

1.3 Spectral Rendering

So far, we have considered the colors during the rendering process to be internally represented by a tristimulus color space. For the explanation purposes, let us assume RGB color space — colors of all objects in the scene are defined in RGB, the path tracing step colors are in RGB and the output color is in RGB. In many 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 has no notion of the light as electromagnetic radiation. Consequently, we are losing a significant amount of information, causing the colors to be inaccurate at times, and some phenomena 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 — *spectral rendering*. The core idea is to track and sample several wavelengths at once for each step of path tracing and to perform the integration over all of 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 electromagnetic radiation, hence the primary focus is placed on spectral rendering. The following sections are mainly based on a publication by Devlin et al. [7].

1.3.1 Color representation

If spectral rendering is desired, a representation of the spectral distribution functions needs to be implemented. While several techniques are feasible, it is often a matter of a simple trade-off between precision and performance. For example, sampling wavelengths uniformly each 5nm would yield significantly more accurate results than sampling only four wavelengths in total. However, such an approach might become unbearable in terms of speed and memory. Moreover, the spectral functions are usually quite smooth, therefore such dense sampling is mostly unnecessary. A few examples of these functions can be seen in Figure 1.13.

A typical approach is to sample at larger ranges (more than 10 nanometers) and use basis functions for the representation [26]. More approaches and their details are briefly explained by Devlin et al. [7].

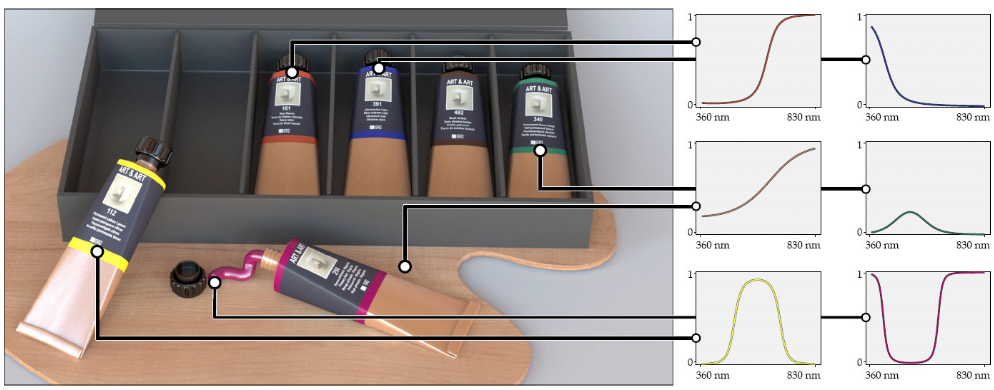


Figure 1.13: Spectral curves measured for different colors[13].

1.3.2 Advantages

Spectral rendering has 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 of light. A comparison between an RGB render and a spectral render of the same scene by Mitsuba2 is shown in Figure 1.14.

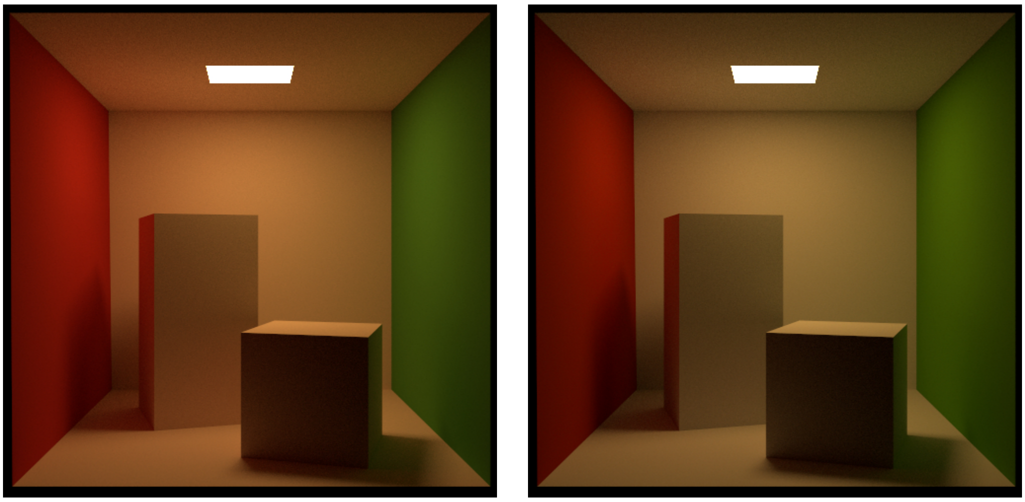


Figure 1.14: A comparison between a RGB render (left) and a spectral render (right) of the same scene by Mitsuba2 [18]

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

Fluorescence Absorption and re-emission of a different color

Dispersion Splitting of polychromatic light into its wavelength components via refraction

Polarisation Change of the oscillation direction of a light wave

Iridescence Thin layer constructive/destructive light 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 chromatic noise. Fortunately, this has been effectively eliminated by the introduction of *Hero Wavelength Spectral sampling* [36]. Also, even though the performance is not a key factor, it worsens as well.

Moreover, we assume that spectral images will never be properly displayable on screens. Even the most modern monitors use a color space that is still smaller than the gamut of human-perceived colors. The only viable way is to store the distinct spectral bands as separate images. However, it is quite inconvenient to have multiple results instead of one as it does not correlate with the images that we capture in reality. Therefore, as the final image is still being displayed on an RGB monitor, the rendering needs to have a correct spectrum to final RGB conversion. Fortunately, the conversion is well-defined and fairly simple to implement.

Since 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 *spectral upsampling*. For those who may be interested in the details of the current development of spectral upsampling, refer to the article by Jakob and Hanika [13], which proposes a solution that is capable of converting full sRGB gamut with zero error.

It is also possible to simply map RGB values to their measured data. However, this requires a lot of manual work as the same ”color” (reflective spectrum) might have to be measured for various lighting conditions and mapped accordingly.

There exist several reasons to integrate spectral upsampling, mainly because the spectral values are a lot harder to obtain (using spectrometer) and to use. Because of the reproducibility of the RGB color space and its legacy usage (lots of textures have always been defined in RGB), it is convenient to input the values of textures as RGB values, convert them internally to the spectral domain, and convert them back to the desired color space for the output image.

2. Appearance Computations

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

2.1 Reflectance

Reflective surfaces are a surprisingly common sighting. As the perfectly diffuse materials practically do not exist in nature, a large set of materials that surround us is 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 geometry optics to be able to properly define a reflectance model. First of all, *Snell's law* [27]

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

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 ~1.5) and it essentially describes the ratio between the speed of light in the vacuum and the speed of light in the current medium:

$$n = \frac{c}{v} \tag{2.2}$$

, where n is the IOR, c is the speed of light in the vacuum and v is the phase velocity of light in the current medium.

However, this gives us only the direction of the refracted light. In most cases, it is also necessary to know the ratio between the amount of reflected and the amount of refracted light. Depending on the polarization of the light (further explained in section 2.2), the *Fresnel equations* [27] 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:

$$F_r = \frac{1}{2}(r_s^2 + r_p^2) \tag{2.3}$$

According to the energy conservation law, the transmitted energy is equal to $1 - F_r$.

Note that the previous computations describe only *dielectrics* — materials that do not conduct electricity and are capable of transmitting light, such as glass, water, diamond, etc. The second large group, *conductors*, consists of all materials with opaque surfaces, such as metals. Conductors also transmit light, however, due to their physical properties, it is quickly absorbed. There also exists a third group called *semiconductors* which are 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.

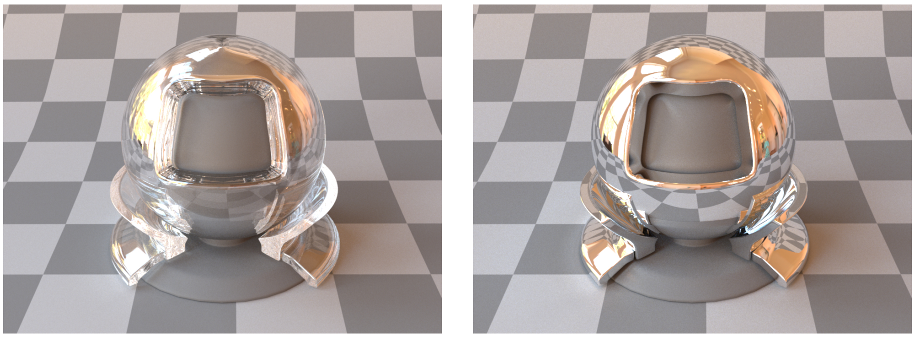


Figure 2.1: A preview of a dielectric (diamond, left) a conductor (aluminum, right) rendered in Mitsuba2 [18]

While this is a simple matter of computing the Fresnel equations, the practice is usually more complicated. Most of the commonly seen materials are at least slightly rough, either on purpose or due to the manufacturing errors.

2.1.2 Microfacet theory

To describe rough surfaces, the *microfacet theory* was introduced by Cook and Torrance [6].

The main idea is that a rough surface consists of *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 the 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

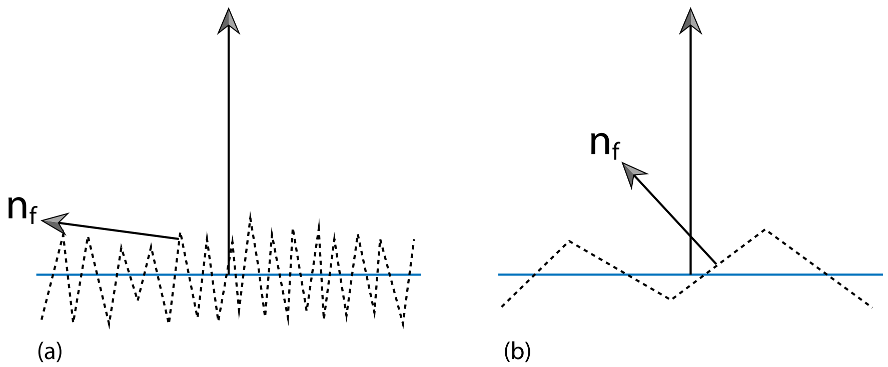


Figure 2.2: A demonstration of a very rough (left) and a relatively smooth (right) microfacet distribution [27]

Many variations to the Cook-Torrance model have been developed, such as its predecessor Torrance-Sparrow [31] model or Oren-Nayar [25] model for diffuse reflectance.

In this thesis, we focus on the microfacet distribution functions, 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 article by Walter et al. [33]. As the exact formulations of these methods are not a necessity for this thesis, we provide only a brief overview for each of them. We also provide a comparison between the GGX model and the Beckmann model in Figure 2.3.

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

Beckmann The Beckmann distribution [2] 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 results very similar to the Phong distribution.

GGX The GGX distribution [33] was introduced as an improvement of the Beckmann's solution for some cases. It maintains stronger tails, thus has better shadowing, and it is based on the measured data of the real rough materials.

2.2 Polarization

Similarly to all kinds of 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. However, it makes the light behave differently upon 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 its oscillation are arbitrary — such light is called *unpolarized*. The *polarized* light

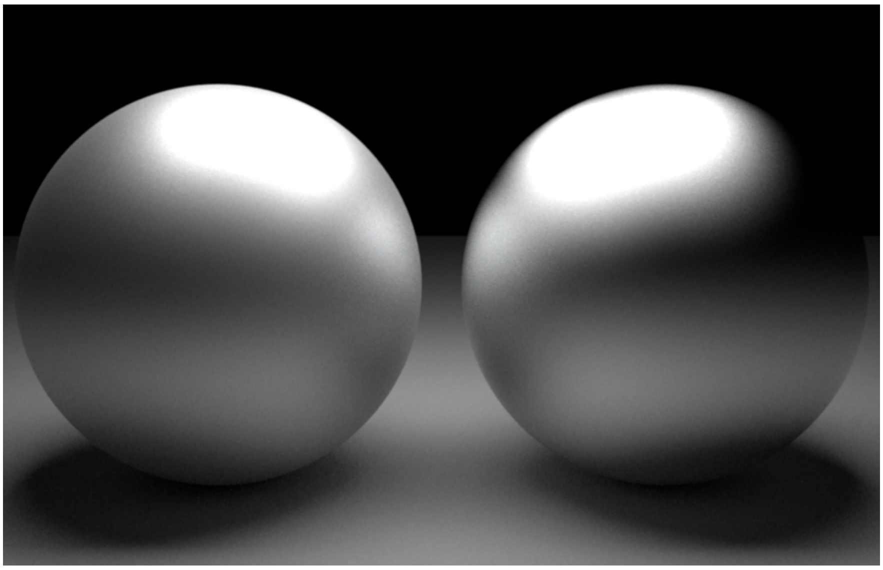


Figure 2.3: A rough aluminum sphere with the GGX distribution (left) compared to its Beckmann equivalent (right) rendered in Mitsuba2

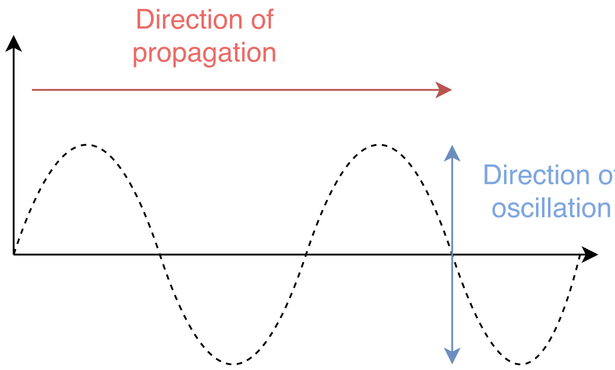


Figure 2.4: Demonstration of the direction of oscillation and the direction of propagation

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.

In reality, each photon is polarized — by default, it keeps the same restricted direction of oscillation until surface interaction. The photons of polarized light are all polarized in the same manner while the photons of unpolarized light are polarized randomly.

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

To create polarized light, a dielectric object may be placed in the direction of propagation of 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

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

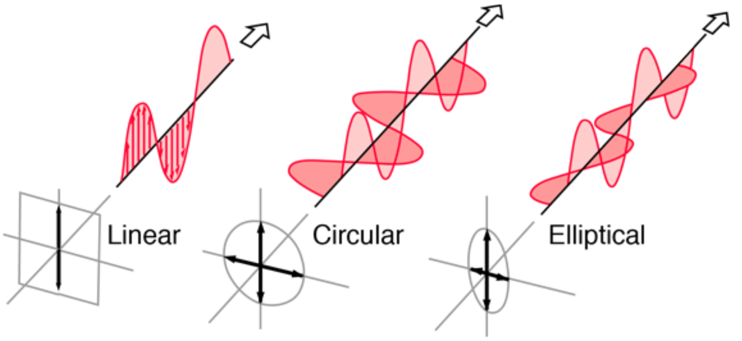


Figure 2.5: An illustrative demonstration of the different types of polarization [1]

is called *Brewster's angle* [4]. It is computed by the following formula:

$$\theta = arctan(\frac{n_2}{n_1}) \tag{2.4}$$

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

In the simplest case of a reflection from a dielectric plane, we distinguish two types of linearly polarized light depending on the relative orientation of their polarization to the incident plane. The reflected light is called *p-polarized*, as its oscillation is parallel to the plane of incidence. The transmitted light is called *s-polarized*, as its oscillation is perpendicular to the plane of incidence. Both are shown in Figure 2.6.

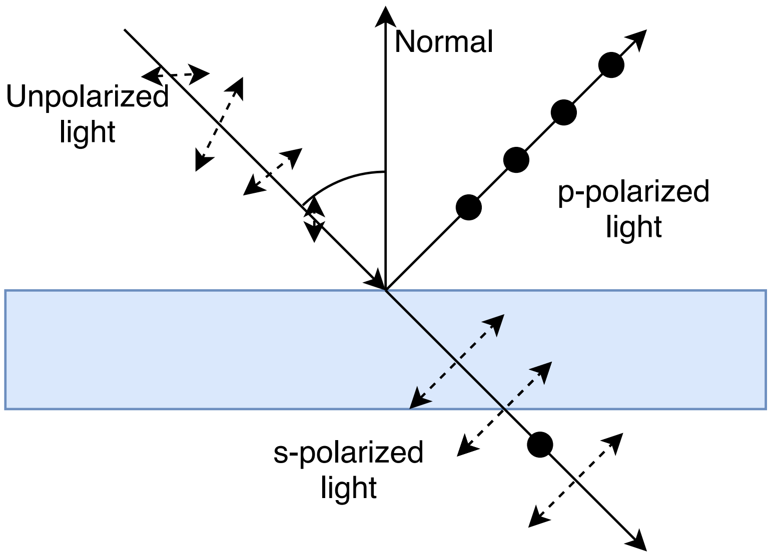


Figure 2.6: Perfectly p-polarized reflected light and partially s-polarized refracted light due to an interaction with a dielectric interface under Brewster's angle

The principle of Brewster's angle is commonly used in a material called the *linear polarizer*. As the name suggests, it polarizes the light, restricting its direction of oscillation according to the properties of the polarizer. If the light that passes through the polarizer is of the opposite (perpendicular) direction to the polarizer's transmission orientation, it will not let it through and no light will be visible. Figure 2.7 illustrates the effects of a polarizer.

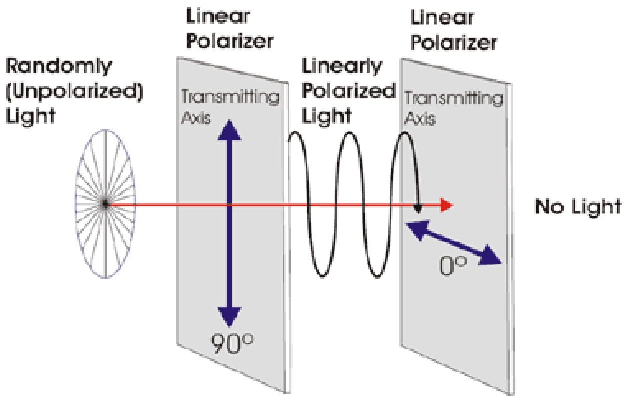


Figure 2.7: Unpolarized light passes through a vertical polarizer → linearly vertically polarized light passes through a horizontal polarizer → no light[2]

This property is frequently incorporated in 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.

Figure 2.8: Polarizing filter by Nikon[3]

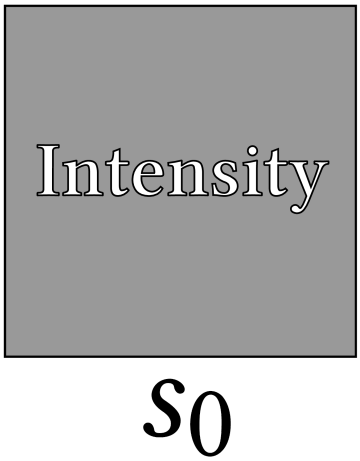
The information about the polarization we cover in this section is sufficient for the purposes of this thesis so we do not need to go into further detail. If the reader wishes to learn more, please refer to scientific literature, for example, the *Polarized light in optics and spectroscopy* [16]

2.2.1 Polarization in rendering

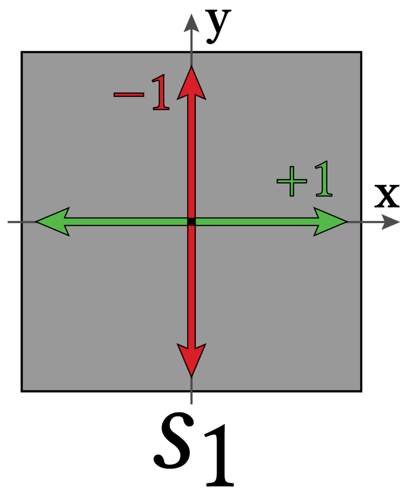
The integration of polarization in the rendering process is quite rare. Only a few scenarios display the effects of polarization and one must implement quite complex

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

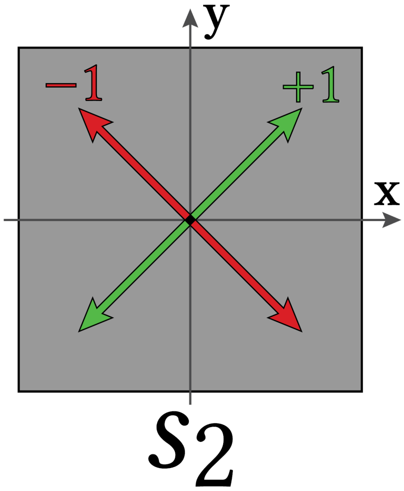
[3] https://www.nikonusa.com/en/learn-and-explore/a/tips-and-techniques/polarizing-filters-add-pow-to-pictures.html



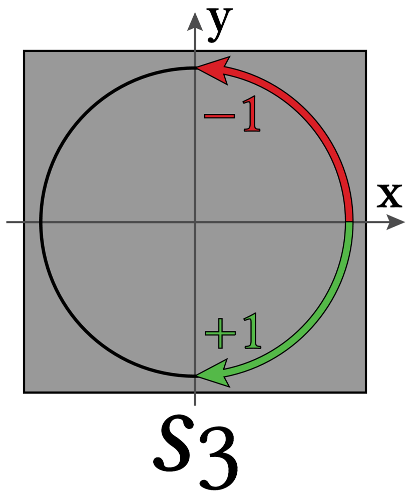
(1) Radiance - no polarization



(2) Horizontal vs. vertical polarization



(3) Diagonal polarization



(4) Left vs. right circular polarization

Figure 2.9: Different information carried by the Stokes vector

behavior of the radiation waves to spectral rendering. However, Mitsuba2 and ART fully track the polarization state of light when needed. The implementation covered by this section is already in Mitsuba2 [18].

The polarization state is represented by the *Stokes vector* — a 4-dimensional quantity that 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.

Also, it is necessary to represent the changes 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 documentation of Mitsuba2 by Nimier-David et al. [21]. 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 the polychromatic light to 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 *dispersion*. In nature, it can be frequently seen when light passes through liquids (e.g. sun through the rain). To artificially reproduce spectral dispersion, an object having a shape of a triangular prism, commonly called dispersive prism 2.10, can be used.

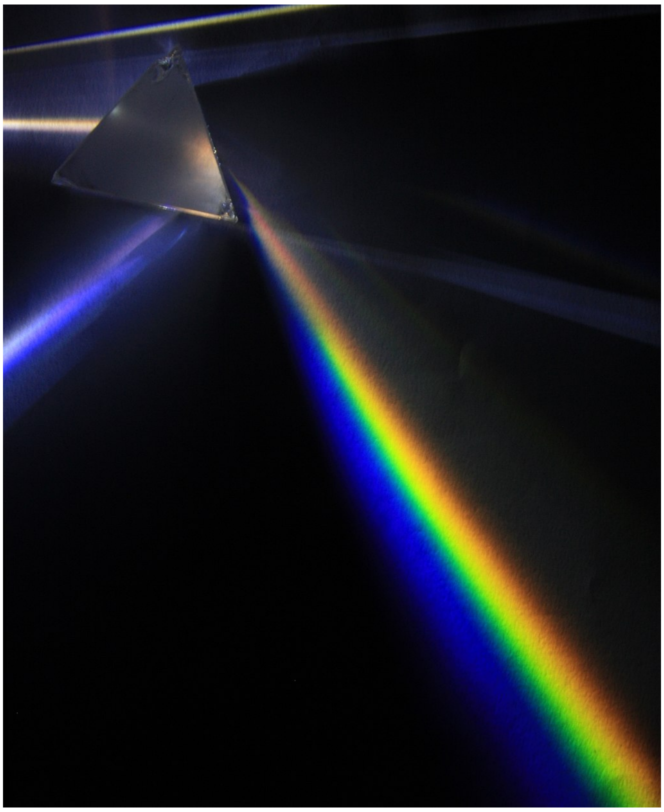


Figure 2.10: Photograph of a dispersive prism[4]

2.3.1 Dispersion in rendering

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

It is necessary to properly represent the varying and not always monotonic IOR — for these reasons, *Sellmeier approximation* [7] is widely used. Then, the renderer must be capable of tracking the possibly dispersed monochromatic rays

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

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

upon a surface interaction of a single polychromatic ray, i.e. create additional samples that were unnecessary before.

2.4 Iridescence

It is quite common that some objects in nature exhibit an interesting behavior where the hue of their surface gradually changes with the viewing angle and the illumination angle. This phenomenon is called *iridescence* or *goniochromism* and it can be observed on very thin materials, such as butterfly wings, soap bubbles, oil, etc. It 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 [3].

We distinguish two main types of iridescence:

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

Thin-film Light interaction with a thin film of a size similar to a light 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 properly described as a physical process and it is already incorporated in Mitsuba [3]. From now on, by iridescence, we mean the thin-film interference.

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 and, based on the incident 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 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.

An obvious observation is that iridescence is also dependent on the thickness of the interacting layer - as the thickness increases, the transmission of the light

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

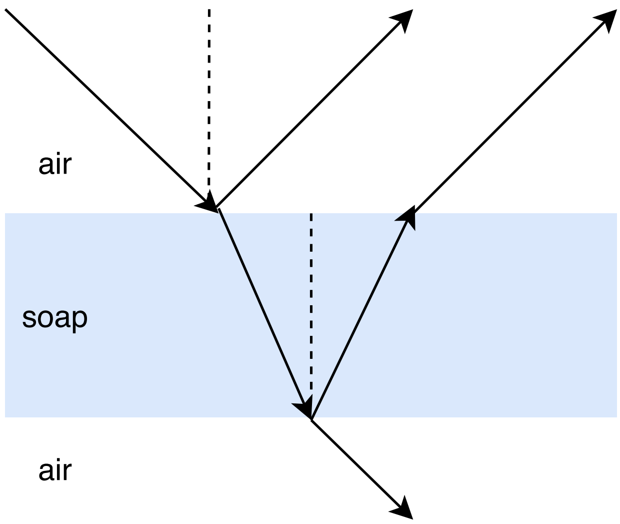


Figure 2.12: A cross-section of light interactions with a soap bubble

takes longer time which consequently causes fewer interferences. The difference between two variously thick films is displayed in Figure 2.13.

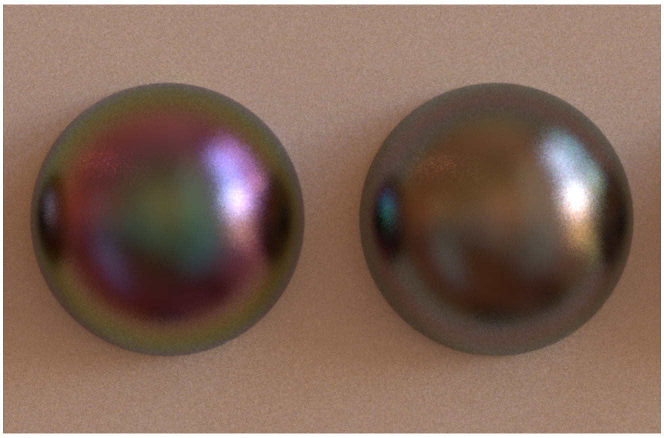


Figure 2.13: Two identical rough conductors with differently thick film layers on top of them: 550nm (left) vs. 1500nm (right) rendered in Mitsuba2

2.4.1 Iridescence in rendering

Based on the publication by Belcour and Barla [3], we overview the computational process of 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 publication by Belcour and Barla [3].

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

1. Compute the reflected and the transmitted values of the Fresnel equations for the IOR of the film and the IOR of the exterior

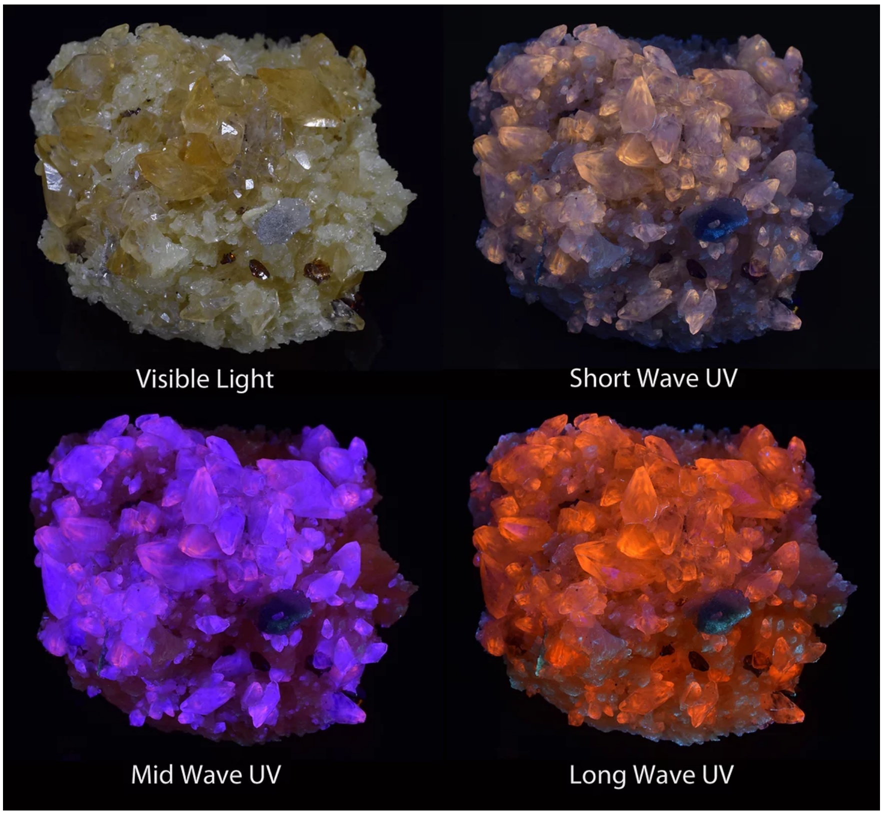


Figure 2.14: Calcite under different lights[6]

2. Compute the optical path 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 of all consecutive reflections and refractions

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 nature, e.g. in various minerals but also 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. An example of fluorescent calcite is shown in Figure 2.14

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

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

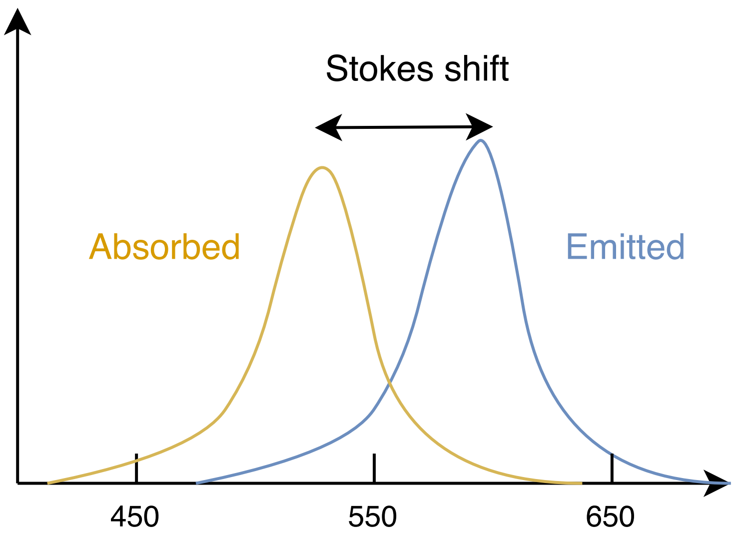


Figure 2.15: Illustration of Stokes shift

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

Phosphorescence Absorbs light and emits different one, even for some time after the light source is gone

Fluorescence Absorbs light and emits different one, 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 light spectrum, the appropriate decision is to extend spectral rendering to include fluorescence. Once again, we refer the interested reader to the article by Mojzík et al. [19] for the implementation details as, for the purposes of this thesis, we are only covering the fundamental ideas.

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

The discrete form of a fluorescence response can be represented by a *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 spectral distribution of the emitted light should not change, only the intensity of the emission spectrum should.

A generalization of the BRDF that includes re-radiation, called *Bi-spectral Bidirectional Reflectance and Re-radiation distribution function (bi-spectral BRRDF)*, was introduced by Hullin et al. [12]. It can be denoted by the following equation:

$$f_r((\omega_i, \lambda_i), (\omega_o, \lambda_o)) = \frac{d^2 L(\omega_o, \lambda_o)}{L(\omega_i, \lambda_i) d\omega_i d\lambda_i} \tag{2.5}$$

The corresponding *bi-spectral rendering equation* can be formulated as follows:

$$L(\omega_o, \lambda_o) = \int_\Lambda \int_\Omega L(\omega_i, \lambda_i) f_r((\omega_i, \lambda_i), (\omega_o, \lambda_o)) d\omega_i d\lambda_i \tag{2.6}$$

Along with polarization, dispersion, iridescence, and reflectance, fluorescence is the last appearance phenomenon that we investigate in this thesis. With all of them covered and properly explained, we may now look into the evaluation process that we propose 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 are, for some reason, still rarely implemented in the modern renderers. However, in the past decade, the interest in 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 the conventional rendering systems, it is necessary to have a testing suite that 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, to our best knowledge, we consider to be the ground truth. These are encapsulated in an automated workflow, which runs the tests with a single command and shows the results in a form of a website. The suite also contains data, such as code snippets, that should simplify the replication process of these implementations so that they 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 a 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 a 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 do not 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 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 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 at the framework of the benchmark suite and its structure. The file organization is demonstrated in Figure 3.1 and the following sections describe its each major subsection.

3.1.1 Code

An automated workflow simplifies the evaluation process and ensures that the user is correctly working with the benchmark. The suite consists of several files to accommodate the principle of modularity:

/data/configuration.json Contains information about all scenes, test case scenarios, and renderers in a single JSON structure. In case the user wants to add a new scene, a renderer, or a test case scenario, he fills this structure instead of writing new code. A part of the file along with explanatory comments is shown in Figure 3.2.

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

/src/configurator.py Parses the `/data/configuration.json` file and prepares structures for both the benchmark script and the results viewing website. In case the `configuration.json` file is incorrect, it stops the benchmark or simply warns the user to fix it.

/src/normalizer.py Normalizes the names of the resulting images after the benchmark ends, as each renderer might have unique naming conventions. It is only used in corner cases.

/src/visualizer.py Runs an HTTP server, which is required by JERI (discussed in subsection 3.1.6) to upload EXR images, and opens the website with the results.

benchmark.py A script that is intended to be directly invoked by the user. Runs other helper scripts mentioned above. His purpose is to call the rendering executable for each of the scenes found in `/data/cases/` according to their configurations.

visualize.py A 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 — it is a modern, fast, well-known scripting language that is perfectly suitable for our purposes, as there is no need for high performance or structurally complicated solutions. Therefore, we do not have to force the user to compile the project and the benchmark is immediately ready to use.

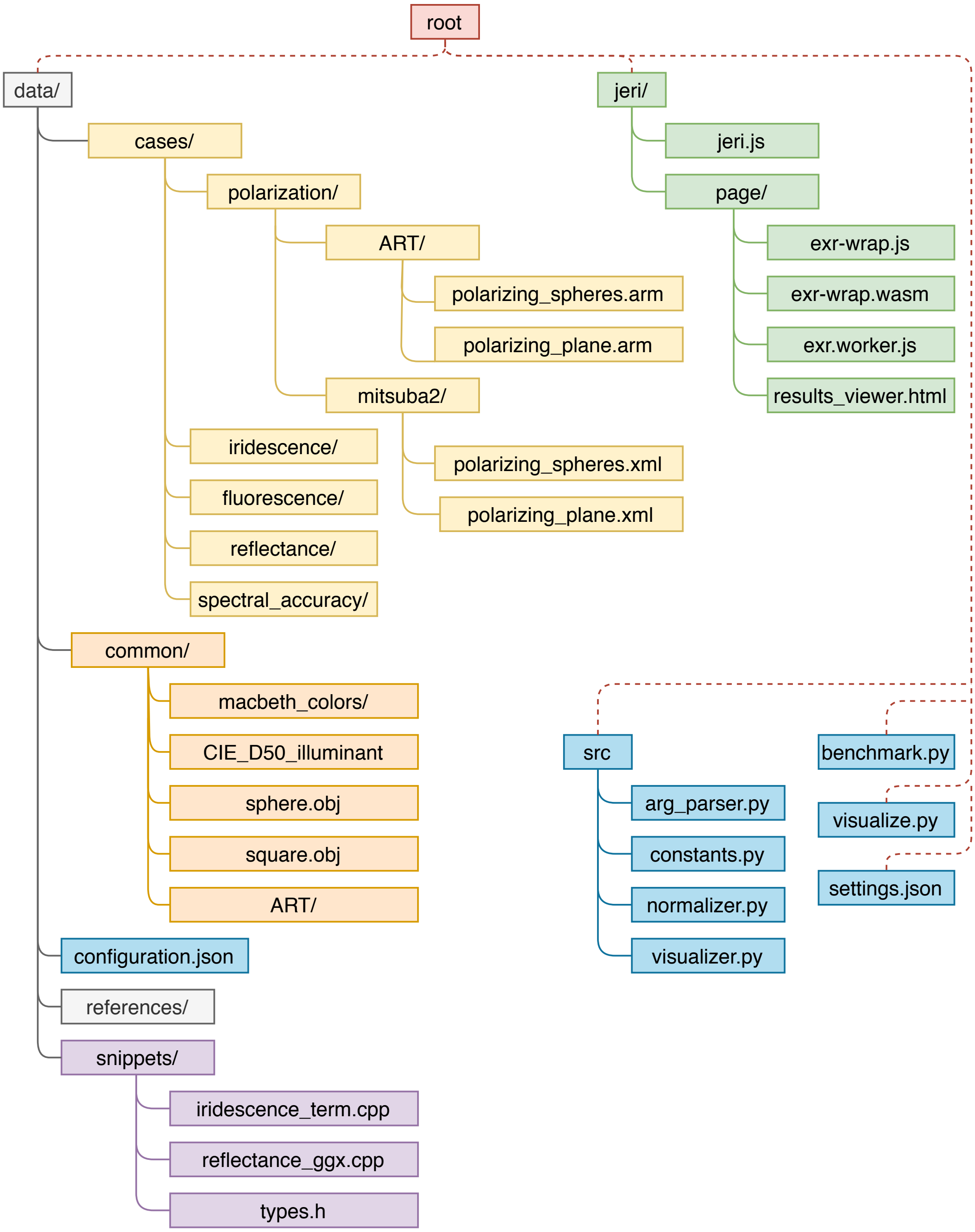


Figure 3.1: File organization of the benchmark

```
{
    "renderers" : [     // supported renderers
        {
            "name" : "mitsuba2",    // name, used in settings
            "format" : "xml",       // scene description format
            "global_params" : ["-t","8"]    // parameters for all scenes
        },
        {
            "name" : "ART",
            "format" : "arm",
            "global_params" : ["-j","8"]
        }
    ],
    "root" : [  // root of all scenes
        {
            "case" : "reflectance", // name of test case scenario
            "scenes": [
                {
                    "name" : "ggx_copper_50", // name of the scene and the output file
                    "file" : "ggx_copper",    // file with scene description
                    "renderers" : [           // renderers that support this scene
                        {
                            "name" : "mitsuba2", // name of the supported renderer
                            "params" : ["-m", "scalar_spectral", "-Drotate=50"]
                                // specific parameters for this scene
                        },
                        {
                            "name" : "ART",
                            "params" : ["-DROTATE=-50","-wp","d65"]
```

Figure 3.2: Part of the configuration file

3.1.2 Cases

The folder /data/cases/ contains the scene descriptions in the following folder structure:

```
/data/cases/<case_name>/<renderer>/scene
```

The benchmark script browses this structure to find every scene as specified in configuration.json. The scenes are explained in great detail in section 3.4.

3.1.3 Common data

The folder /data/common/ contains information and values that are used during the rendering process. 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 [23] rescaled for Mitsuba2

square.obj Square mesh with the length of side equal to 2

sphere.obj Unit sphere mesh

ART/square.arm ART specific file, contains a method that creates a square mesh compatible with the sphere.obj

3.1.4 Reference images

The folder /data/references/ provides the reference images for each tested scene. Most of these are rendered in Mitsuba2, except for the fluorescent ones, which are rendered in ART, as the fluorescence is not yet supported by Mitsuba2.

We decided to use the EXR format for the reference images, mostly because we wanted to grant the user as much information as possible (e.g. exposure). In this case, typical image formats, such as PNG, are insufficient. Also, EXR can be considered a standard for HDR image viewing.

The file polarizing_spheres.exr is somewhat unique, as it contains four different images in four channels. Our visualizer is adjusted to support multi-channel EXRs, but in case the user wants to check them manually, he needs a viewer that supports such format.

3.1.5 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 implement them into his own renderer. The folder contains:

[1] https://xritephoto.com/ph_product_overview.aspx/?id=1192&catid=28

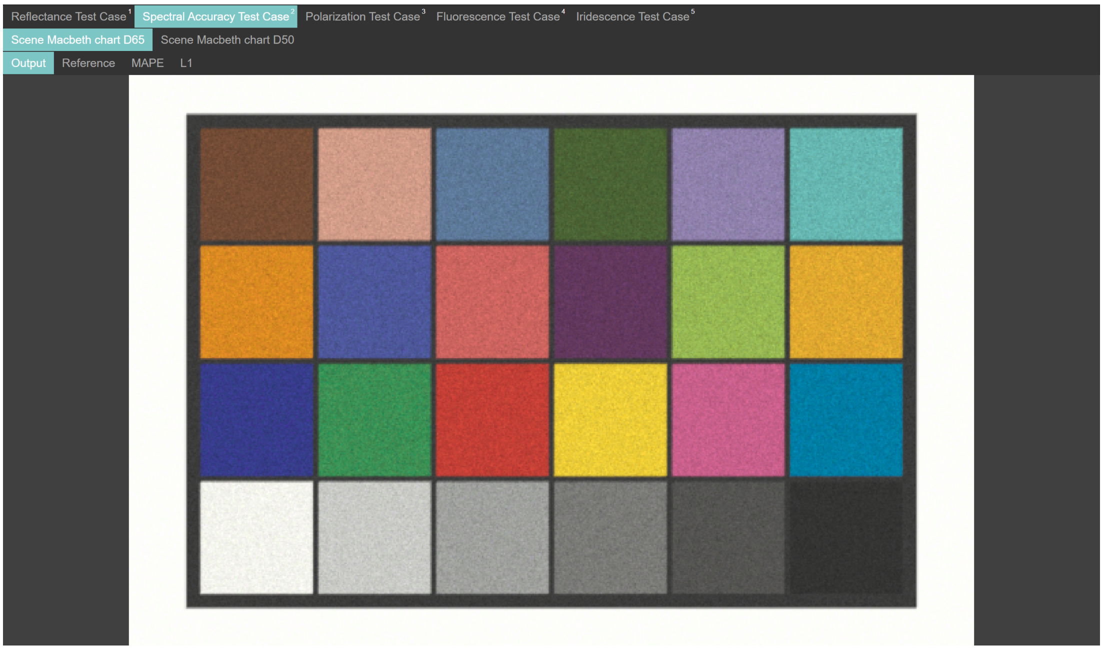


Figure 3.3: Results website

iridescence_term.cpp Computation of the iridescence term along with the helper functions, inspired by the code created by Belcour and Barla [3]

reflectance_ggx.cpp Methods for sampling, evaluation and masking according to the GGX reflectance definition [33]

types.h Structures used in the snippets mentioned above

3.1.6 JERI

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

JERI (Javascript Extended-Range Image) is a website EXR viewer written in
JavaScript and developed by Disney Research [8]. It is simple to use and to integrate and provides many features over images, such as zoom, change of exposure, and automatic error maps.

We use JERI to display the results of the whole benchmark on a single website. The user may look at the results, the reference images, and even their differences (compared by L1 and SSIM error maps). A screenshot of the results website is shown in Figure 3.3.

Please note that the difference images are supposed to be a helper tool for the user rather than an absolute metric of the correctness. The user is encouraged to use his own difference images or to assess their inconsistencies differently.

3.2 Supported Spectral Renderers

In the current state, the benchmark supports two renderers — *Mitsuba2* and *ART*. Both are physically-based, research-oriented rendering systems, capable of

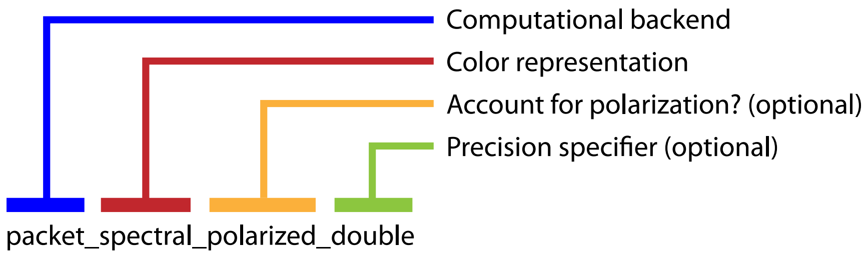


Figure 3.4: Different variants of Mitsuba2

representing the light in the 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 two following sections contain overviews of these renderers.

3.2.1 Mitsuba2

Mitsuba2 has been released only recently (paper was published in November 2019 [21]) 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 trends in programming. Its specifics can be found in the documentation by Mitsuba [18] and Nimier-David et al. [21] and the project can be downloaded/cloned from `https://github.com/mitsuba-renderer/mitsuba2`.

It is written in C++17 and designed to be modular — it contains a large number of various plugins with each adding 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 geometry such as spheres

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

and many more.

Mitsuba2 is capable of running in several modes, from RGB CPU rendering to differentiable GPU spectral rendering that tracks polarization. Multiple options can be arbitrarily combined, which is illustratively shown in Figure 3.4. The important part is that the renderer is retargetable, which means that the user may specify the rendering mode without recompiling the project. Thanks to the template metaprogramming that C++ offers, Mitsuba2 chooses the appropriate internal representation of its data. For example, in RGB rendering, the color is described by an array of 3 floats, representing red, green, and blue color respectively. In spectral rendering, the array contains 4 floats, where each represents the spectral power of a specific wavelength and a stochastic approach is used to sample these wavelengths.

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

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

3.2.2 ART

Advanced Rendering Toolking, or shortly ART, is a 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 the current version of ART is 2.0.3, most of their work has not been ported from versions 0.x/1.x. ART can be download/cloned from `git://cgg.mff.cuni.cz/ART.git` and this section is largely based on its documentation [22].

ART is written in Objective-C and therefore compilable on the majority of 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, ART generates custom spectral images as the immediate results of the rendering process which contain a lot more information about the wavelengths and the polarization states than standard EXRs. These, however, are not displayable and need to be tonemapped (tonemapper is included in the project) afterward.

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

Spectral rendering Uses Hero wavelengths spectral sampling

Fluorescence Supports fluorescent materials and volumes

Polarization Capable of tracking full polarization state of the light

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

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

3.3 Methodology

Before we proceed to the descriptions of individual test scenes, we should describe the general procedure of adding a test case to the benchmark. The following steps are used to design a test case scenario so that it is simple, coherent and complete:

Natural phenomenon We study the physical models of evaluated phenomena, as we need to understand their behavior in nature, i.e. under what circumstances it is perceivable, what does it look like, how to reproduce it, etc.

Computer graphics research To test a specific appearance computation, it already has to be developed and embedded inside a rendering system. Therefore, we search for publications that discuss the algorithms, the structures and the internal representations of physical quantities used for these computations. Furthermore, as we test two open-source renderers, we also study exact implementations that are actually evaluated in the rendering process as these give us all the in-depth information we need.

Choice of aspects Based on the research, we decide what aspects of the phenomena we are going to test. This heavily depends on the specific computation, but, generally, it is fairly simple to deduce the potential troubles with an algorithm from the scientific publications. They often include measurements and shortcomings of their implementations, which simplifies the choice of the to-be-tested aspects. Overall, we take into consideration such computational aspects that are reproducible, easily comprehensible, and most importantly, clearly visible in the final image as the actual assessment is done solely from the reference images.

Visualization of aspects As soon as we decide on the aspects that are to be exercised, we design the scenes that should visualize them. However, we can never fully test all possible combinations as there are, in fact, infinitely many. Therefore, we aim to form a minimum number of scenes while maximizing the coverage. This is quite simple — many of the phenomena exhibit similar behavior under various circumstances and so they can often be categorized. For example, thicker dielectric layers always exhibit lesser iridescent effects, so it does not make sense to test it on different base materials. These categories are then pretty straightforward to describe.

Manual evaluation After the scenes are done and they are rendered in the reference renderer, we manually evaluate each of them based on their physical models and the research papers.

3.4 Scenes

This section documents all the test case scenarios and test scenes that we use in our evaluation process. We analyze the important objects in the scenes, their meaning and we provide justifications for our decisions.

Each scene follows the same basic principle — the geometry is supposed to be as simple as possible. The reference images might not be conventionally eye-pleasing, as we are aiming for individual aspects of the specific features that should confirm the correctness of their computations. Also, the number of samples for each scene is generally low, as more samples simply make the picture prettier but do not change (after a certain threshold) the final color.

3.4.1 GGX Reflectance

While we mostly focus on the appearance phenomena caused by spectral rendering, we include the reflectance of rough surfaces in our benchmark, purely because it is still a widely discussed topic. Specifically, we look into the implementations of the GGX microfacet distribution which, in most cases, is considered to be the the state of the art for rough surfaces. The whole evaluation is based on the publication by Walter et al. [33].

This test case scenario consists of five scenes.

The first four scenes are done according to the data measured by Walter et al. [33]. They are variations of the same scene — a single rough copper plane under an area light illuminant. Depending on its rotation, various reflected light intensity is visible throughout the plane, from the strongly illuminated tails at the bottom to the more sparsely distributed direct illumination at the top.

We provide four different rotations of the plane — 50, 60, 70, and 80 degrees rotated around the x-axis. We believe that such granularity is necessary to be completely sure that none of the viewing/illumination angles break the implementation. The copper material provides a visible contrast to the dark background, as it is more colorful than other metals, such as aluminum or silver. This makes the differences clearly visible even to the naked eye. The roughness is set to 0.2, which can be expressed as considerably rough — less roughness is meaningless, as the differences may not be that visible, and more roughness could create an unrealistically rough surface that could be hard to differentiate from a diffuse one. While we could simply put all four planes in one image, we decided to separate them. This allows us to isolate the light source and observe the variously rotated plane under the same circumstances. All four scenes are shown in Figure 3.5.

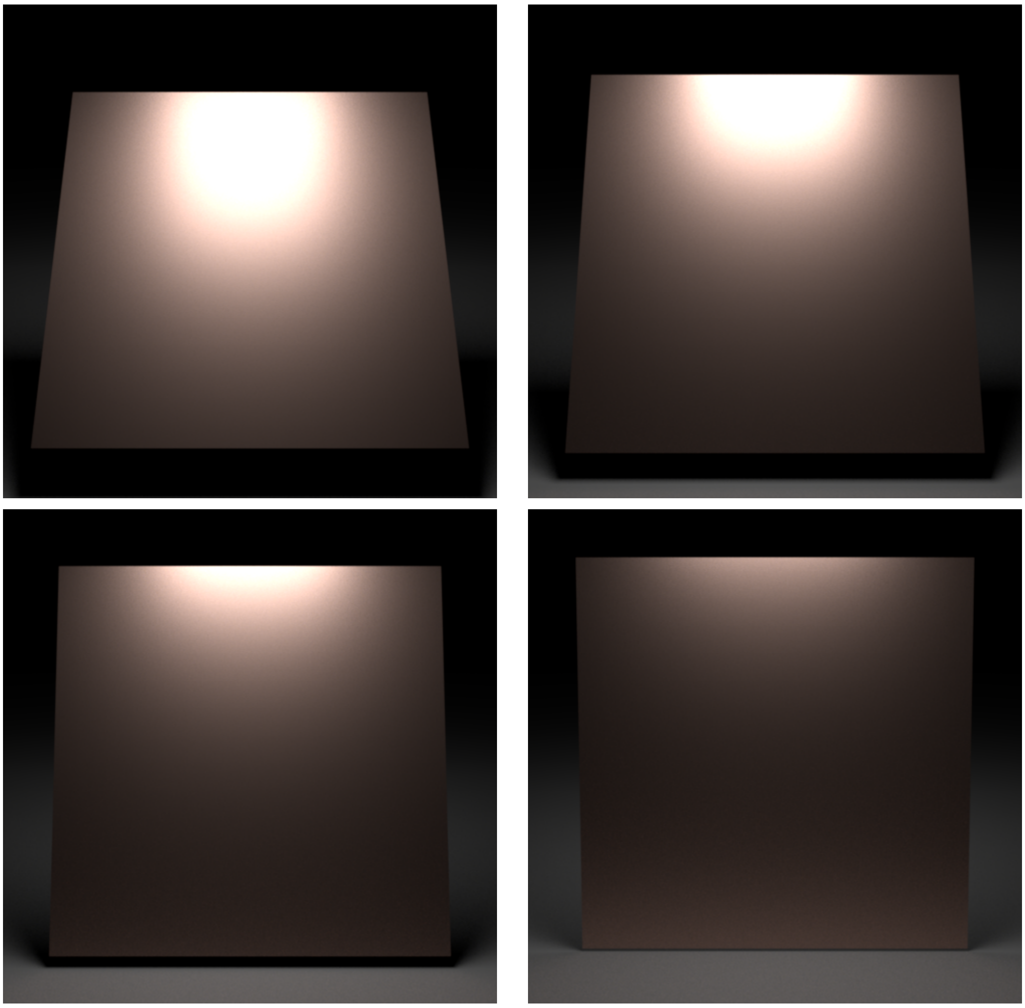


Figure 3.5: Four test scenes consisting of a copper plane with GGX distribution

The fifth scene demonstrates a rough dielectric — it consists of a single equally rough dielectric sphere under an area light illuminant. We decided to use a volumetric object instead of the plane for the dielectrics, because a dielectric is at least partially transparent and the colors of a plane would predominantly merge with the background. A sphere, however, forms a thick layer in front of the background. Furthermore, testing the same cases for the dielectrics does not make much sense, as we are not testing the material but the microfacet distribution. A sphere also provides various illumination angles where we can potentially spot the inconsistencies between them. This scene converges pretty slowly for such simple geometry, as the original GGX implementation needed a quite large number of samples (256 in our case) to render a plausible result. An improvement has been introduced by Heitz [11] that does not change the distribution, but greatly decreases variance. As the article by Walter et al. [33] mostly compares the GGX approach to the Beckmann distribution, we demonstrate our scene, rendered for both distributions, in Figure 3.6

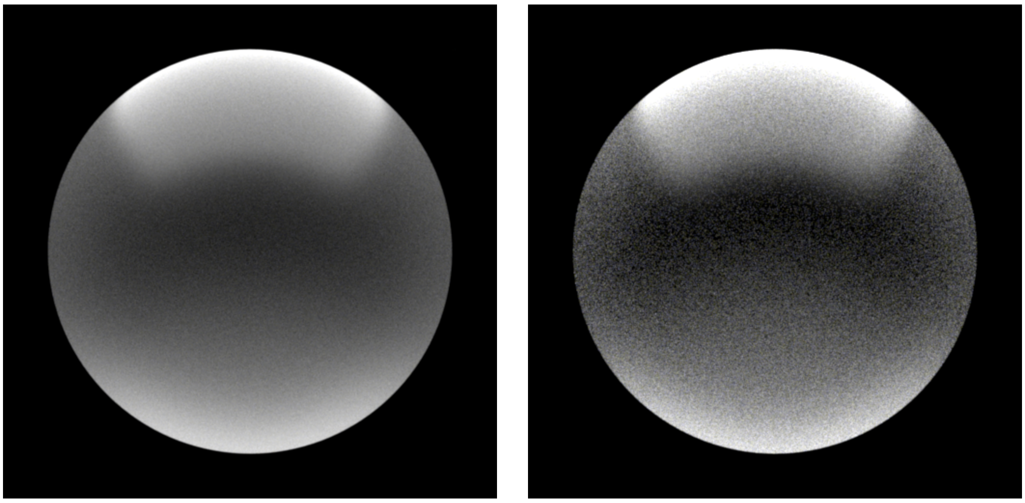


Figure 3.6: The test scene consisting of a dielectric sphere with GGX distribution (left) compared to its Beckmann equivalent (right)

3.4.2 Spectral accuracy

As the direct representation of spectral colors is not possible on our screens, we need to make sure that the final image contains colors that correspond to the real ones. Distinct renderers may internally represent their color values in a very different manner, which might affect the ultimate color that we see.

For these purposes, we use a well-defined set of colors introduced by McCamy et al. [17] called *Macbeth chart* or simply *Macbeth colors*. Its basic variant consists of 24 color patches divided in a 4x6 grid, each representing a color that can be commonly found in nature — human skin, flowers, greyscale range, etc. It is primarily used for color calibration, therefore it is designed to be stable and invariant under different lighting conditions.

We introduce two scenes that evaluate the correctness of spectral colors — both contain a Macbeth chart illuminated by two different standard CIE illuminants, D50 and D65 [24]. As both illuminants and Macbeth colors have exact definitions, they are easily integrated into any renderer that supports spectral color representation. Another advantage is that we know their corresponding RGB values so the results can be easily checked against them.

We assume that the white point of the renderer is CIE D65 (whitepoint of sRGB) for both scenes. Thus, the scene illuminated by the D50 illuminant should have an orange-ish overlay and the scene illuminated by the D65 illuminant should have a completely white background.

Both scenes are shown in Figure 3.7. Once the renderer passes this test, we can assume that his representation of spectral colors is indeed correct. Even though this may seem trivial, it does not make much sense to test different illuminant/material combinations. These would only be variations of our tests with the difference that their correctness needs to evaluated manually as we do not have the color definitions beforehand.

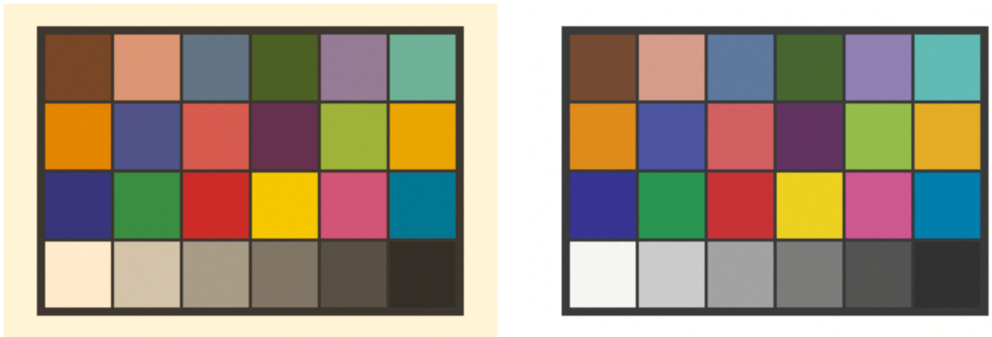


Figure 3.7: The test scenes containing a Macbeth chart under CIE D50 illuminant (left) and CIE D65 illuminant (right)

3.4.3 Polarization

Unlike GGX or iridescence, polarization cannot be described by a BSDF. It requires full tracking of the polarization states during the rendering process and ideally a tool that would display these states. Fortunately for us, both ART and Mitsuba2 provide a way to display the results of the Stokes vector, as well as polarization filters that are ideal for these kinds of experiments.

The first two scenes use Brewster's angle (explained in section 2.2) — they contain a perfectly reflective dielectric plane (IOR of glass = 1.52) that is illuminated by a single area light under Brewster's angle. By definition, the light reflected from the plane is perfectly p-polarized. The difference between the two scenes is in the linear polarizer situated in front of the camera — in case the transmission axis is horizontally oriented (parallel to the reflected light), the reflection of the light source is visible without any attenuation. However, if the transmission axis is vertically oriented (perpendicular to the reflected light), there is no reflection of the light source on the plane, as the polarizer will not let it through. Both scenes are shown in Figure 3.8.

The third scene is a lot more specific, as it heavily depends on the renderer's capability to represent the polarization states via Stokes vectors and to visualize their components. This scene contains two smaller dielectric spheres that are in front of a larger smooth conducting sphere illuminated by a constant daylight. Due to the constant surrounding light, lots of reflections are happening on the dielectric spheres which consequently polarize the light.

Note that to ensure physical plausibility, the scenes are rendered monochrome, with 550nm wavelength colors only. In ART, it is possible to extract a single wavelength information image from its custom spectral image format. In Mitsuba2,

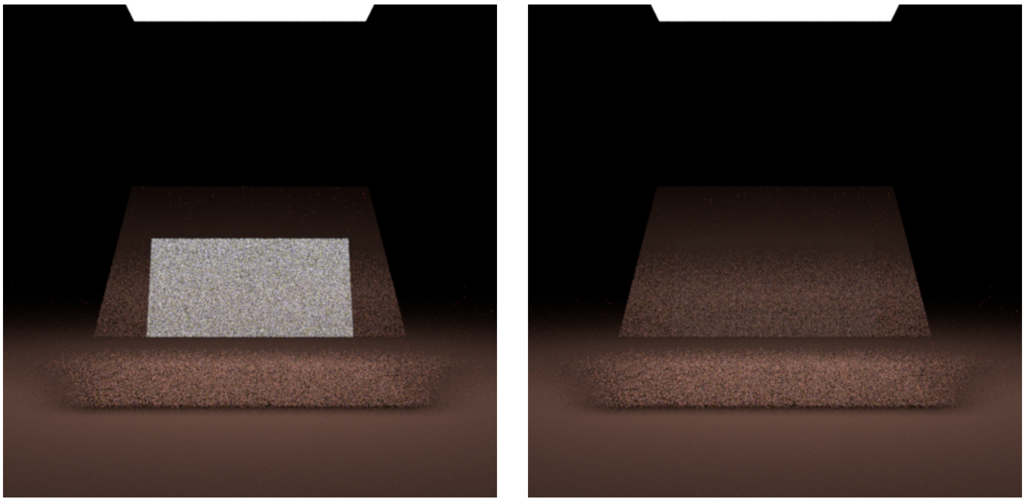


Figure 3.8: The test scenes with a visible polarized light (left) and without a visible polarized light (right)

we used a workaround by specifying only the 550nm values of all colors inside the scene and by running the rendering process in the monochrome mode.

Both Mitsuba2 and ART output their results as four distinct images, each representing a different element of the Stokes vector (the complication with the compatibility of the outputs and its solution is explained in subsection 3.6.3). All four images are shown in Figure 3.9.

3.4.4 Fluorescence

Fluorescence is one of the four phenomena that are possible to reconstruct thanks to spectral rendering. As it can be perceived only under rare circumstances, its implementation has been purposely avoided in the vast majority of the renderers. However, as physical realism begins to be a must-have in the commercial world, the interest in its integration grows.

We provide three different test scenes that exercise the fluorescent materials and their properties accordingly to their natural behavior. In this case, only ART scenes are provided, as Mitsuba2 does not support the feature.

The test scenes are inspired by the fluorescence implementation by Mojzík et al. [19]. They all consist of a single yellow-based sphere, whereas its properties are essentially various combinations of the absorption spectrum, emission spectrum, and the surrounding constant illumination. The following description explains each combination:

D50 illuminant, 370nm absorption, 650nm emission 3.10 The sphere absorbs 370nm wavelengths and re-emits 650nm while it is placed under the CIE D50 horizon light illuminant. As shown in the figure, the sphere displays a yellow to orange color which is a combination of the yellow base and the emitting red color (650nm wavelength is perceived as red). The sphere is not completely red due to the lower spectral power of the D50 illuminant around 370 nanometers wavelengths.

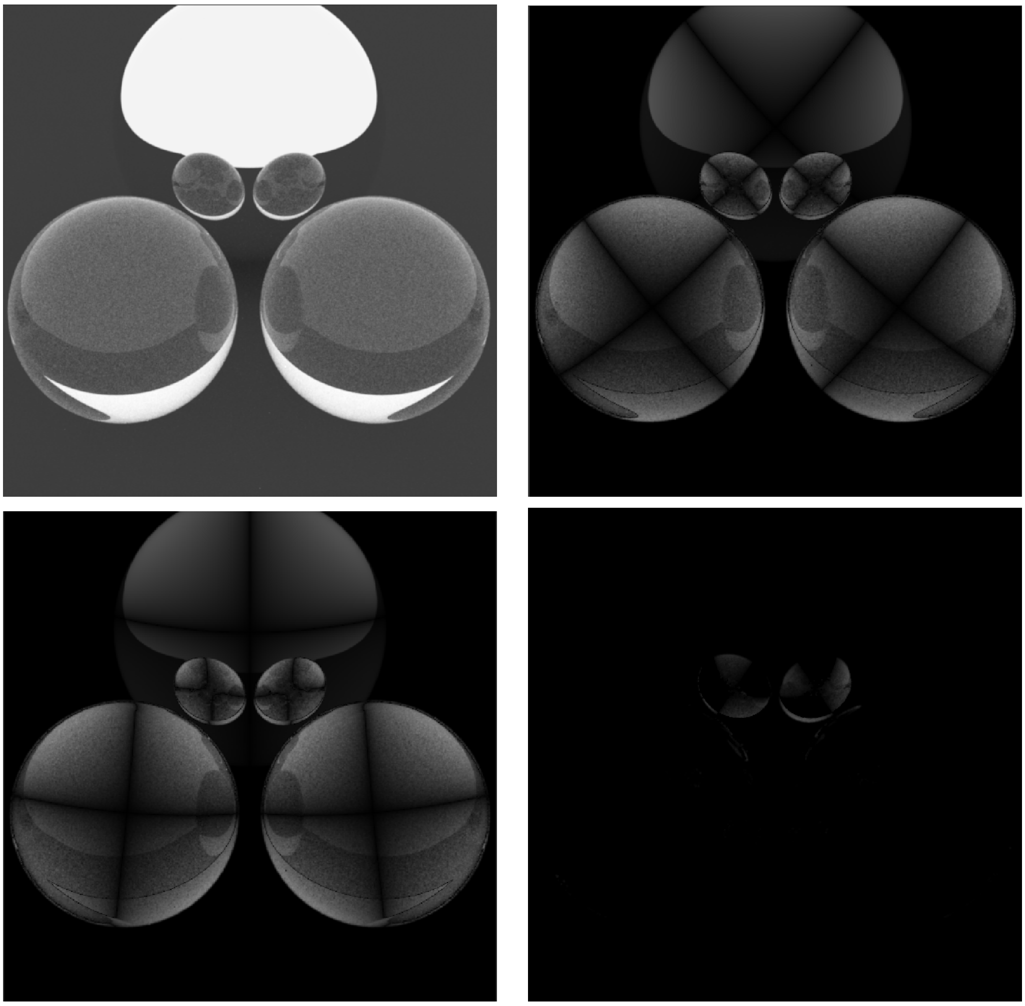


Figure 3.9: The four Stokes vector outputs of a test scene containing polarizing spheres

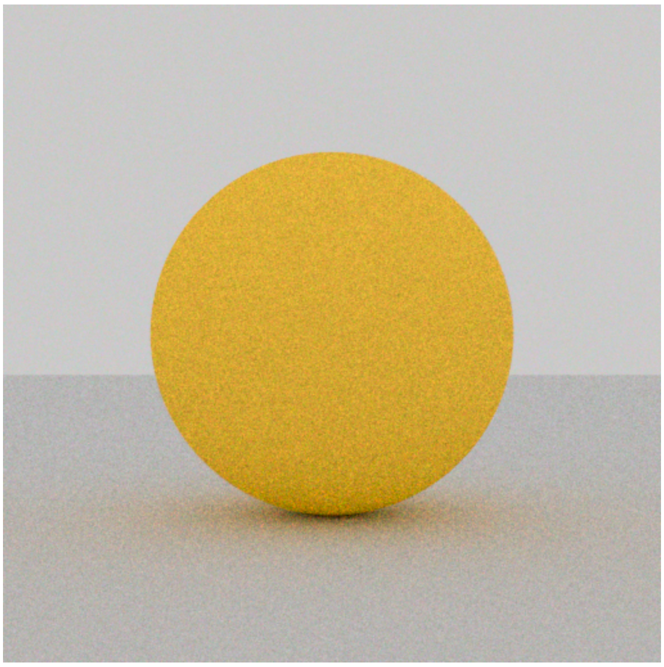


Figure 3.10: D50 illuminant, 370nm absorption, 650nm emission

450nm illuminant, 450nm absorption, 650 emission 3.11 The sphere absorbs 450nm wavelengths and re-emits 650nm while it is placed under a monochrome light, emitting 450nm wavelengths only (perceived as blue). Intuitively, as the absorption spectrum and the illuminating spectrum collide, the sphere should emit a bright red color.

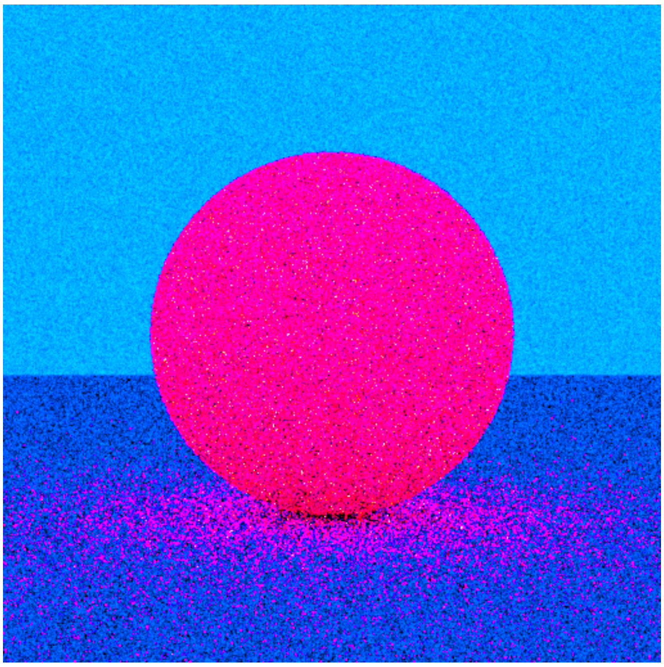


Figure 3.11: 450nm illuminant, 450nm absorption, 650 emission

450nm illuminant, 370nm absorption, 650 emission 3.12 The sphere absorbs 370nm wavelengths and re-emits 650nm, while it is placed under a monochrome light, emitting 450nm wavelengths only (perceived as blue). In this case, the two spectral domains do not collide at all and therefore, there is no light to absorb. Due to the missing emission and the sphere's opaque surface, it appears to be black. Note that a dielectric would be completely transparent. However, we do not test this case as it does not concern fluorescence directly, but rather the material itself.

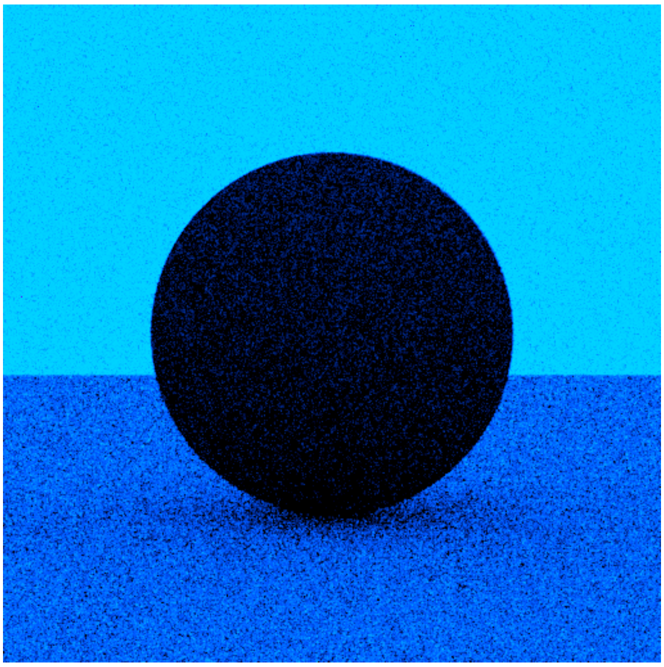


Figure 3.12: 450nm illuminant, 370nm absorption, 650 emission

Many more combinations testing different emitting/absorbing spectral domains and illuminants are possible. However, we believe that they would only be variations of the three test case scenarios mentioned above — normal light with a fluorescent sphere, monochrome light with a sphere absorbing different wavelengths and monochrome light with a compatible sphere.

3.4.5 Iridescence

The evaluation of iridescence presented some complications, as neither Mitsuba2 nor ART natively support the iridescent effects. The only implementation we found was for Mitsuba version 0.6, introduced by Belcour and Barla [3].

We have reimplemented the plugin to Mitsuba2 and used the Mitsuba 0.6 implementation as a reference. The results of the tested scenes were identical, so we can safely consider the Mitsuba2 images as the ground truth.

The implementation simulates the iridescence caused by the light interference in a very thin dielectric film on top of a rough conducting base. Three parameters can be set for this iridescent BSDF — the IOR of the exterior, the IOR of the film and the height of the film. Unfortunately, the final colors of the iridescent effects change rapidly with every slight variation of the parameters of both the conducting base and the dielectric film. Despite that, there are some consistent changes with the gradual increase of the film height and the film IOR, which we expose in our test scenes.

Each scene consists of four spheres surrounded by constant daylight to see the colors as brightly as possible. These spheres have a very low roughness coefficient ($\alpha = 0.1$) so that the rough surface does not distort the iridescent effect.

Film height 3.13 As we have explained in section 2.4, the increasing thickness of the film reduces the visibility of the iridescent effect as there is less light interference. All spheres in this scene have identical base and film IOR — $\eta_{base} = 1.9, \kappa_{base} = 1.5, \eta_{film} = 1.33$. The first two spheres with the film height equal to 300nm and 550nm are displaying equally visible iridescent effects. The difference is, of course, in their colors as the light interferences are largely varying. The film height of the third sphere is quite high, 1500nm, which clearly diminishes the iridescent effect, dominantly displaying its base color. The fourth sphere has no film layer on top of it. It displays only the base and it is used for reference.

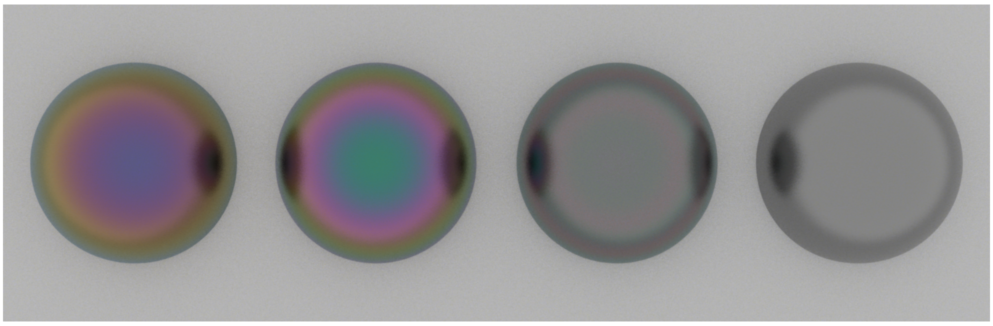


Figure 3.13: Different film heights

Film IOR 3.14 Another apparent change happens upon adjustments to the film IOR. With decreasing IOR of the film, more color fringes can be spotted on the sphere. From the definition of IOR (essentially, how much faster light travels in the vacuum than in the current medium), we can deduce that the faster the light travels through the medium, the interferences happen at a higher rate and therefore we can see more colors. The spheres in this scene have identical base and film height — $\eta_{base} = 1.9, \kappa_{base} = 1.5, film\_height = 550nm$. The film IORs ($\eta_{film}$) are equal to 1.2, 1.5, 1.8 and 2.8, where each consecutive sphere displays one less color fringe than the previous one (from 5 to 2). Please note that the counted amount of the color fringes is not an absolute measure but rather a rough approximation done by naked eye — there are a lot more colors gradiently transitioning between the fringes and the user may count them differently.

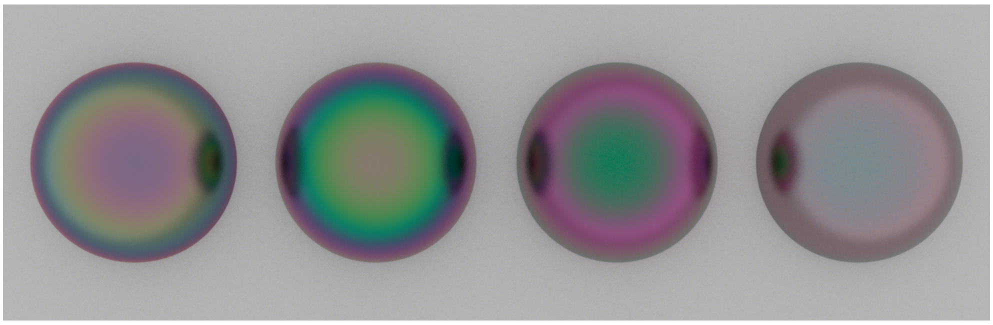


Figure 3.14: Different film IORs

Materials 3.15 The last scene demonstrates a direct comparison between a well-defined material and its equivalent with a thin-film layer on top of it ($\eta_{film} = 1.33, film\_height = 550nm$). There are two such pairs, showing copper and mercury, whose properties η_{base} and κ_{base} are provided by Mitsuba. Both materials emit the same iridescent effect simply put on a different color base.

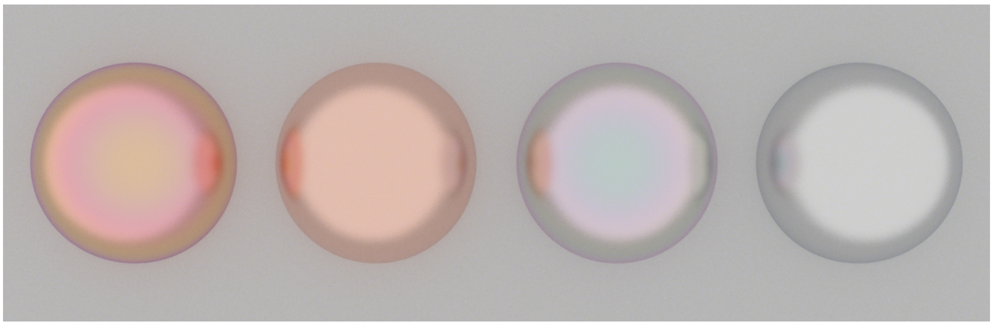


Figure 3.15: Different base materials

Due to the custom implementation of this plugin for Mitsuba2, we do not evaluate ART, as it does not support iridescence at all.

3.4.6 Dispersion

Unfortunately, we do not provide any test scenes for dispersion. Mitsuba2 does not support the feature at all, which is also explicitly stated in their documentation. Although ART supports dispersion, we have encountered some issues with the darker colors of the dispersive materials and so we cannot declare the images displaying dispersion generated by ART as the ground truth.

3.5 Use cases

In the following sections, we demonstrate the basic uses cases of our testing suite.

3.5.1 Use case Regress test

A user who recently changed an essential part of the renderer (e.g. sampling strategy) wishes to confirm that he did not break any other functionality. He downloads our benchmark and inserts his latest executable. Then, he runs all test case scenarios and compares the results with the reference images. Refer to the user guide for more information about the usage of the benchmark A.

3.5.2 Use case New feature

A user wishes to implement one of the tested features to a rendering system that is not yet evaluated for that specific scenario, but generally supported by the benchmark (e.g. implementation of GGX microfacet distribution to ART). He wishes to check the correctness of his implementation, so he follows these steps:

1. If provided, find the code snippets that are attached to the benchmark suite and integrate them to the renderer.

2. Duplicate the scenes prepared for other renderers that test the feature and store them in a new `/data/cases/feature/renderer/` folder. This step should be straightforward — the objects inside the scenes are fairly basic and explained in comments.

3. For each effected scene, add a JSON snippet to `configuration.json` file containing the newly supported renderer's name and its parameters (look at Figure 3.2).

4. Insert the latest executable to the benchmark and run it.

The benchmark automatically detects the new configuration. Either from the reference images or the difference images, the user may now run the evaluation to assess the accuracy of the newly implemented feature.

3.5.3 Use case New scenario

A user who wishes to add a new test case scenario must follow these steps:

1. Add a new folder to the benchmark suite `/data/cases/new_scenario/`

2. Add a JSON snippet for the new scenario to the `configuration.json` file

From now on, the benchmark automatically evaluates this feature as well. If the user wants to provide the reference images, these need to be stored in the `/data/references` folder.

3.5.4 Use case Improved feature

A user who improved a specific feature and wishes to confirm his assumptions may use the benchmark in a standard way.

The scene descriptions may be considered as templates used to create the reference images. In case some structural changes were done inside the renderer, such as renaming of variable names, the scenes can be easily adjusted to comply with the new renderer.

If the user finds out that the results indeed improved, he may replace the existing scene descriptions, as well as the reference images, with the better, adjusted ones.

3.5.5 Use case New renderer

A user who wishes to add support for a new renderer into the benchmark suite must follow these steps:

1. Create a new folder with the name of the renderer for each test case scenario in the `/data/cases/`.

2. Rewrite the template scenes from other renderers to his own scene format.

3. Store these scenes in the newly created folders according to their names.

4. Add the renderer's name to the array containing all supported renderers at the beginning of the `configuration.json` file.
5. For each scene, add a JSON snippet to the `configuration.json` file containing the newly supported renderer's name and its parameters.
6. Insert the latest executable to the benchmark and run it.

3.6 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 not the main aim of this thesis — therefore, they are not yet in a state that can be used for a pull request, as this process requires a significant amount of time.

3.6.1 GGX for ART

We have encountered the use case described in subsection 3.5.2. We have decided to add the GGX distribution to the ART renderer and, coincidentally, it correlated with the mentioned scenario. As we had the scenes and the reference images prepared for Mitsuba2, we simply replicated them for ART and implemented the GGX. Then, as mentioned in the use case, we iterated the evaluation process and the adjustments in the code until the results were satisfactory.

The implementation can be found in the attachments of the thesis in section B.3.

3.6.2 Iridescence for Mitsuba2

Mitsuba2 does not have a native support for 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 yet been an effort to rework the plugin. Thus, we took the initiative and re-implemented it.

Along with the publication done by Belcour and Barla [3], they released a supplementary plugin for Mitsuba 0.6 to demonstrate their results.

There were some major changes to the spectral sampling strategy and the overall object structure done in Mitsuba2, which had to be adapted in the new, re-implemented version of the plugin.

The correctness of the rework has been confirmed similarly to our evaluation workflow — we prepared the test scenes for iridescence, rendered them for both Mitsuba2 and Mitsuba 0.6, and considered the latter version to be the ground truth. The implementation can be found on `https://github.com/marcel1hruska/mitsuba2` or in the attachments of this thesis in section B.2.

3.6.3 Multi-channel EXR support for JERI

While designing the polarization test scenes, we have encountered a compatibility issue between the Stokes vector output format of Mitsuba2 and ART. While ART stores the polarization states in distinct EXR images, Mitsuba2 creates a single multi-channel EXR, where each channel contains the Stokes vector information. Such a visualization of the rendering results requires a custom solution.

First of all, we have added general support for multi-channel EXR images to the JERI framework, 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 structure called `otherChannels` that maps the channel name with its contents.

2. The user may specify the channel name to display its contents. This can be done similarly to other JERI-defined properties.

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

Secondly, we created a highly custom wrapper over the first addition to resolve the incompatibility between the outputs. By default, we assume that the results are stored in four distinct images (similarly to ART), e.g. a file named `sphere.s0.exr` means that it is the output of the first element in the Stokes vector (the radiance). We test whether such a file exists — if not, we assume from the name of the file that the user actually wants to display the channel named `s0` of the file `sphere.exr`. Therefore, we attempt to find it instead of the former one.

This behavior is transparent to the user. If no version of the file exists, JERI displays nothing.

This addition is a part of the compiled JavaScript code and not the TypeScript source. It can be found in the benchmarks' files `/jeri/exr-worker.js` and `/jeri/jeri.js`. Our code is marked by comments containing the text `multichannel custom support`.

4. Results

This chapter overviews the results of the evaluation of ART via direct comparison with Mitsuba2. As we have noted before, the fluorescence is exclusive to ART, iridescence to Mitsuba2 and therefore we are not evaluating these, as there is no other renderer to compare them with.

Unfortunately, we do not provide any measures that would assess the accuracy of the techniques in an absolute manner, such as accuracy percentage or correctness threshold. Such a metric would require extensive research in the image validation, which is out of the scope of this thesis. For anyone interested, the verification of the rendering algorithms is discussed in an article by Ulbricht et al. [32].

4.1 Error maps

The only assessment we provide is via difference images. Even though these do not exactly state whether the computation is correct or not, they might help to expose some inaccuracies.

The difference images shown in the following sections are created by L1 and SSIM error maps, both included in the JERI framework and consequently in our benchmark.

4.1.1 L1

L1 loss function, also known as the *least absolute deviations* method, computes the absolute difference between the given data and the predicted function:

$$L1 = \sum_{i=1}^{n} \mid y_i - f(x_i) \mid \tag{4.1}$$

Generally, it is largely used in the statistical optimizations to find a function that behaves similarly to the given data.

In our case, the error map is an image that shows the direct difference between the color values for each pixel. This is considered by us to be a basis functionality as it might expose the color inaccuracies that are barely visible to the naked eye.

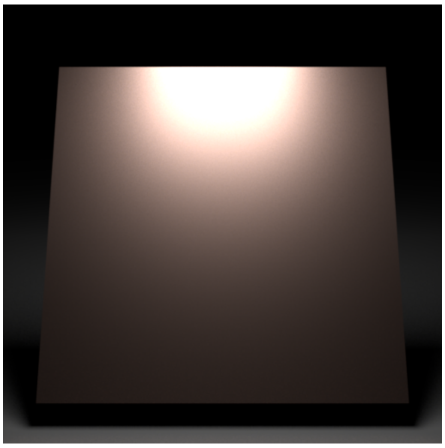
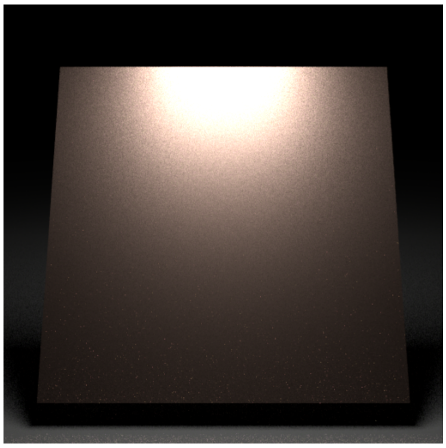
4.1.2 SSIM

Structural similarity index measure (SSIM) is a lot more advanced method, specifically developed by Wang et al. [34] to measure the loss of data between a reference image and its processed (usually compressed) equivalent. As it is a perceptual metric comparing two images capturing the same scene, we include it in our benchmark.

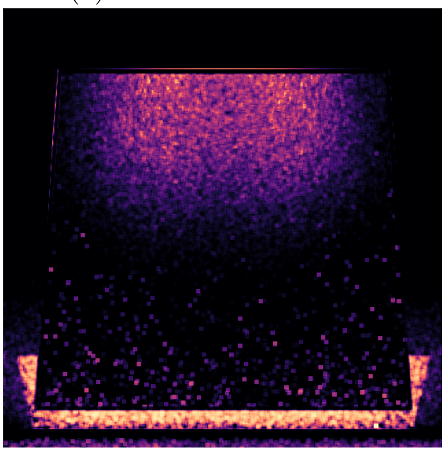
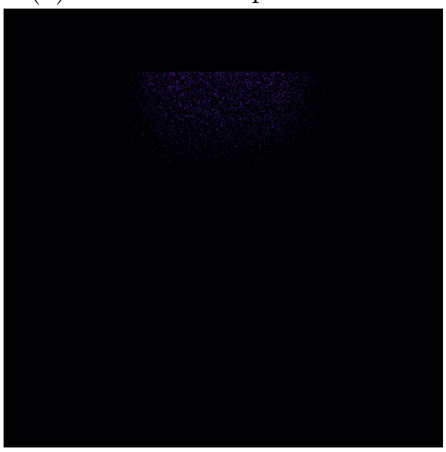
However, this technique cannot be considered as an absolute metric of correctness either, as we often do not let the images to converge completely because of performance reasons. It is only useful to look at the artifacts exposed by SSIM or to manually increase the number of samples for each scene.

4.2 GGX

As the GGX implementation for ART was developed by us, we needed to evaluate its correctness. In this section, we discuss the copper plane scene (rotated by 60 degrees) and the glass sphere scene, their implementations in Mitsuba2 and ART, and their difference images. All of these images are displayed in Figure 4.1 and Figure 4.2.

(1) Mitsuba2 - reference (2) ART - our implementation

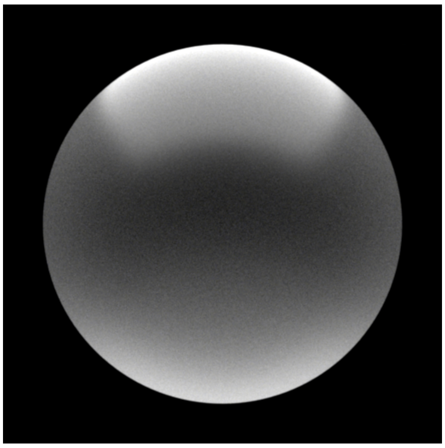



(3) SSIM (4) L1

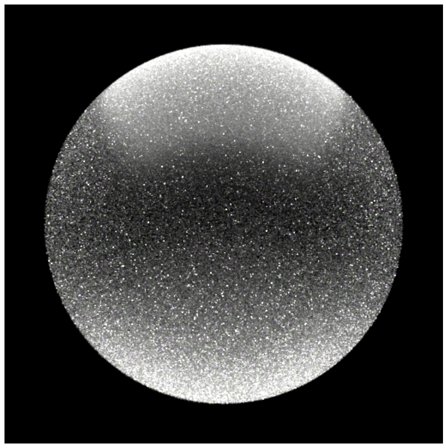
Figure 4.1: Comparison between the reference image and the ART implementation of the GGX copper plane (rotated by 60 degrees) scene

We should note that Mitsuba2 implements the improved variant of GGX developed by Heitz [11], which significantly reduces variance. We implemented the original, basic variant [33], which logically creates a lot of noise for the same amount of samples. Consequently, this diminishes the usability of the SSIM method, as the result simply can not converge to the reference state. This is true especially for the glass sphere, as there are lot more reflections under various angles than on a simple plane.

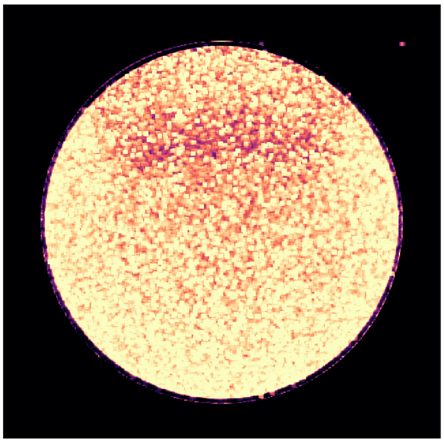
However, the L1 function nicely shows that our implementation should be correct. In case of the plane, we see only a few dots, signaling that some samples might have brighter colors. In case of the sphere, we can see larger groups of dots at the bottom, which, as the colors match, is probably caused by the general rough surface model and not the microfacet distribution.



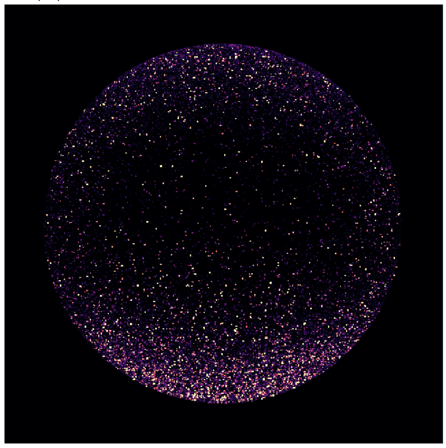
(1) Mitsuba2 - reference



(2) ART - our implementation



(3) SSIM



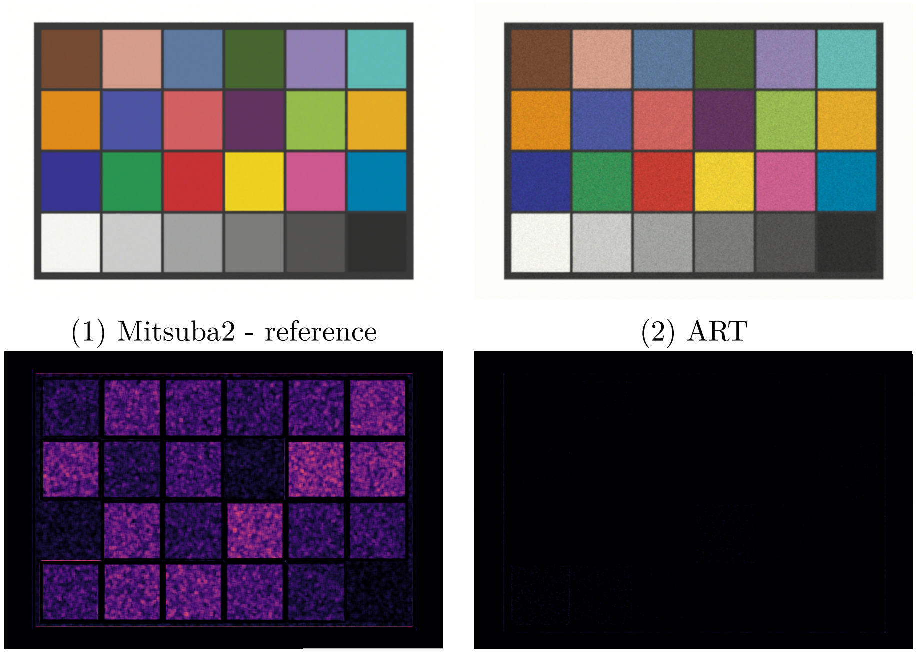
(4) L1

Figure 4.2: Comparison between the reference image and the ART implementation of the GGX glass scene

4.3 Spectral Accuracy

Spectral accuracy is significantly harder to quantify and measure, as there are various mechanisms inside a rendering system that might influence final colors. Both testing scenes for spectral accuracy are compared in Figure 4.3 and Figure 4.4.

At first sight, the two images are extremely similar — the spectral definitions for the Macbeth colors and the illuminants are same for both renderers, which can be confirmed from the L1 comparison. But, if we look at the SSIM, there are slight variations in different color patches. The Macbeth blue and black are accurate, but for some reason the others, especially yellow and yellow-green are slightly more saturated in the reference images. This might indicate a potential flaw in the stochastic approach of spectral sampling in Mitsuba2, but as the differences are barely there, we do not consider the results to be incorrect. Note that the two SSIM images are almost identical, which shows that the relative comparison under the same lighting conditions matches.



(1) Mitsuba2 - reference (2) ART

(3) SSIM (4) L1

Figure 4.3: Comparison between the reference image and the ART implementation of the Macbeth chart D65 scene

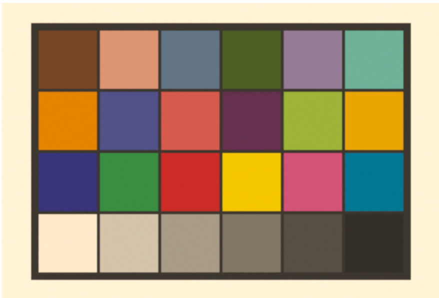
4.4 Polarization

Unfortunately for us, it is quite complicated to directly compare the outputs of the Stokes vectors, as the final format heavily depends on the renderer. To make the matters as simple as possible, the scene that displays polarization states is made monochrome and tracks a single-wavelength light only. We compare the visualization of the horizontal/vertical polarization stored in the Stokes vector in Figure 4.5.

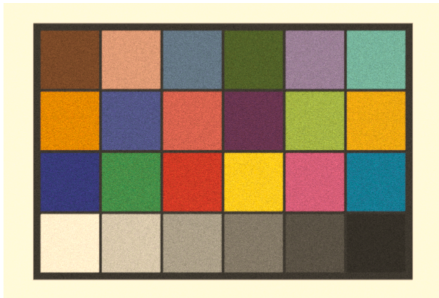
The second discussed scene is shown in Figure 4.6 and demonstrates the polarizing plane scene with a visible reflection.

The polarizing plane scene might seem to be more attenuated in case of ART, especially under the plane. However, this does not concern us, as the purpose of the scene is to expose the behavior of unpolarized light upon interaction with a surface under Brewster's angle rather than the material's transmission properties. Neither of the difference images shows obvious inconsistencies but both are suggesting that the reflection of the light on the plane is at least partially incorrect. Again, this might be caused by the overall attenuation of ART's polarizing filter.

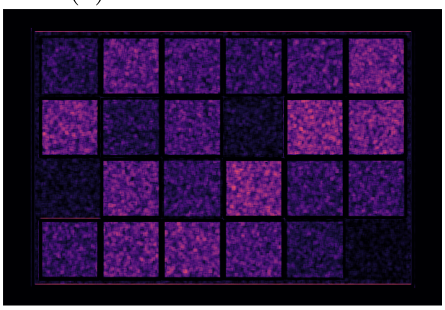
For the simplified scene with polarizing spheres, we can see that the results are almost identical. However, there is a visible difference in the vertical polarization on the spheres (properly exposed by SSIM) and we could conclude that Mitsuba2 does not track completely unpolarized light properly. However, bear in mind that the images we compare are not direct results of the rendering process, but rather a byproduct of a specific structure inside it. Therefore, it might simply be a color output inconsistency rather than a polarization tracking flaw.



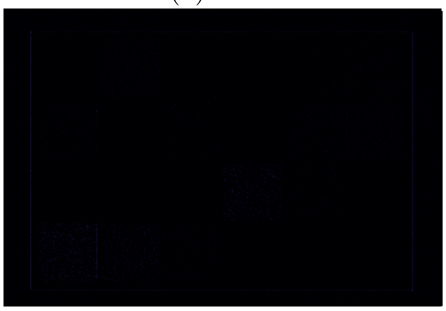
(1) Mitsuba2 - reference



(2) ART

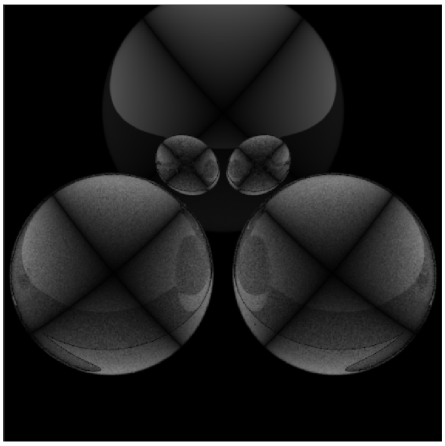


(3) SSIM

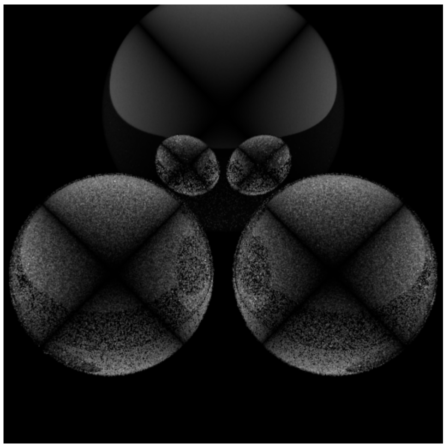


(4) L1

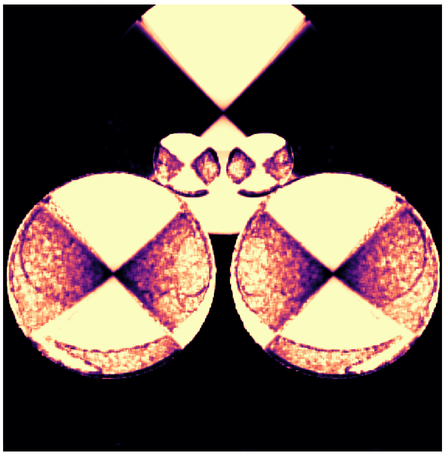
Figure 4.4: Comparison between the reference image and the ART implementation of the Macbeth chart D50 scene



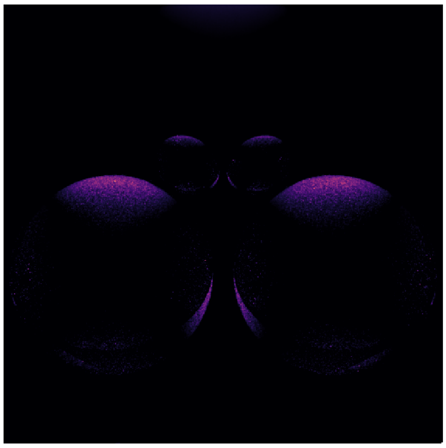
(1) Mitsuba2 - reference



(2) ART

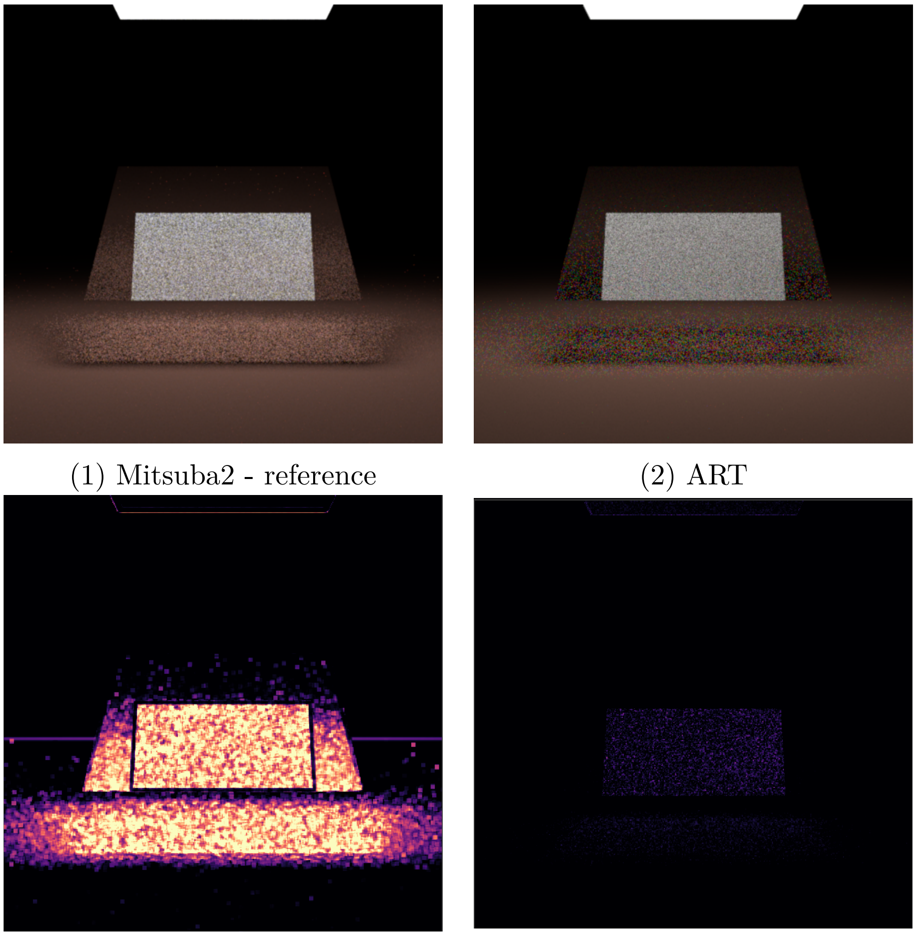


(3) SSIM



(4) L1

Figure 4.5: Comparison between the reference image and the ART implementation of the second Stokes vector element of the polarizing spheres scene



(1) Mitsuba2 - reference
(2) ART
(3) SSIM
(4) L1

Figure 4.6: Comparison between the reference image and the ART implementation of the polarizing plane (filter rotated by 90 degrees) scene

Conclusion

Both ART and Mitsuba2 display several unique properties and implementation details, such us custom spectral sampling techniques. Despite that, we have shown that it is possible to methodically test the appearance computations of the distinct renderers. Each tested phenomenon has a properly defined physical model, which allows us to expose some of the exact aspects of its computations and to evaluate its functionality.

We have successfully created evaluation scenes that focus on the following scenarios: polarization, GGX reflectance, iridescence, fluorescence and overall spectral accuracy. Each of these scenarios contains a small number of scenes that can be rendered in a short amount of time while they, to our best knowledge, demonstrate all essential aspects of the tested feature. Unfortunately, dispersion had to be omitted and left for future work, due to its unverified implementation.

The straightforward descriptions of the scenes, their basic geometry along with their in-depth documentation should be enough to comprehend their purpose in the evaluation process. As an addition, the benchmark contains various data to help a potential user to extend the evaluation suite or his rendering system coherently and correctly. These include code snippets, unified geometry, and spectral values for various colors that are used in the scenes.

Even though we do not include an absolute metric that would explicitly determine the correctness of individual computations, the reference images along with the difference images provide enough information for a reasonably skilled user to assess the accuracy by himself.

4.5 Future Work

Despite the various functionalities that the benchmark has, there are several possible extensions that we consider interesting or useful. However, they were not essential for the purposes of this thesis and we purposely avoided them. Providing more time, these would be fine assets to the benchmark, further extending its capabilities and effectiveness.

Enhanced results Right now, the results visualizer consists of a very basic UI where the user may look at the images and interact with them. Several features could be implemented, e.g. performance counter, comments explaining each scene, highlights of the scene, etc.

Dispersion As the dispersion is the only phenomenon that we have talked about but have not evaluated, it would be appropriate to add it to the benchmark as soon as the implementation for Mitsuba2 and/or ART is fully functional.

More renderers The addition of multiple renderers heavily depends on the supported features of the specific renderer and the interest of its developers. More renderers provide more cross references, which could potentially expose previously hidden inaccuracies and consequently improve the effectiveness of the whole evaluation framework.

Common scene format Including more renderers would be significantly simplified by describing the scenes in a common scene format (e.g. Universal Scene Description by Pixar Animation Studios [28]). This approach would, of course, need a conversion tool from the universal format to the renderer-specific one.

Bibliography

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

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

[3] 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.

[4] 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.

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

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

[7] Kate Devlin, Alan Chalmers, Alexander Wilkie, and Werner Purgathofer. Tone Reproduction and Physically Based Spectral Rendering. In *Eurographics 2002 - STARs*. Eurographics Association, 2002. doi: 10.2312/egst. 20021054.

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

[9] 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.

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

[11] Eric Heitz. Sampling the ggx distribution of visible normals. *Journal of Computer Graphics Techniques Vol*, 7(4), 2018.

[12] 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.

[13] 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.

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

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

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

[17] Calvin S McCamy, Harold Marcus, James G Davidson, et al. A color-rendition chart. *J. App. Photog. Eng*, 2(3):95–99, 1976.

[18] Mitsuba. Mitsuba2 documentation. `https://mitsuba2.readthedocs.io/en/latest/`. Accessed: 2020-6-7.

[19] 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.

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

[21] 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.

[22] Computer Graphics Group of Charles University in Prague. Art documentation. `https://cgg.mff.cuni.cz/ART/assets/ART_Handbook.pdf`. Accessed: 2020-20-7.

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

[24] International Commission on Illumination in cooperation with Technical Committee ISO/TC 274. Colorimetry – part 2: Cie standard illuminants. `http://cie.co.at/publications/colorimetry-part-2-cie-standard-illuminants`, 2020.

[25] 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.

[26] 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.

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

[28] Pixar Animation Studios. Universal scene description documentation. `https://graphics.pixar.com/usd/docs/index.html`. Accessed: 2020-22-7.

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

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

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

[32] Christiane Ulbricht, Alexander Wilkie, and Werner Purgathofer. Verification of physically based rendering algorithms. In *Computer Graphics Forum*, volume 25, pages 237–255. Wiley Online Library, 2006.

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

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

[35] Alexander Wilkie. CGG Course: Introduction to Color Science. `https://cgg.mff.cuni.cz/~wilkie/Website/NPGR025.html`. Accessed: 2020-11-7.

[36] Alexander Wilkie, Sehera Nawaz, Marc Droske, Andrea Weidlich, and Johannes Hanika. Hero wavelength spectral sampling. In *Computer Graphics Forum*, volume 33, pages 123–131. Wiley Online Library, 2014.

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

A. User Guide

The user may clone/download the benchmark from the github repository `https://github.com/marcel1hruska/render_benchmark` or he can find it in the attachments of thesis B.1. As the whole benchmark is written in Python3, there is no need to compile the project.

The benchmark can be run with a simple invocation of the `benchmark.py` python file. It accepts the following parameters:

--help (-h) Print 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

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

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

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

For the user's convenience, all of these options may be specified as JSON in the `settings.json` file, located in the root directory. Note that the ones issued in the CLI take higher priority and override them.

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

Then, the user may invoke a second script, `visualize.py`, that opens a website with the results of the last benchmark run along with the reference images and their difference images. It accepts the following parameters:

--help (-h) Print the help message

--outputs (-o) <output_folder_name> Specify the name of the `outputs-yyymmdd-hhmmss` folder that is to be visualized. If omitted, the latest folder is picked.

B. Attachments

All attachments are listed below and can be found in the `attachments` folder.

B.1 Benchmark

The `render_benchmark` folder contains the whole testing suite. It can also be downloaded from `https://github.com/marcel1hruska/render_benchmark`. If the user downloaded the thesis from github, he needs to do so recursively as the `render_benchmark` is a submodule.

B.2 Iridescence for Mitsuba2

The `Iridescence_Mitsuba2` folder contains the iridescence implementation for Mitsuba2. To add the functionality, replace the code of Mitsuba2 version 2.1.0 by the contents of this folder. Note that only the `iridescence.h` and `iridescence.cpp` files contain our code — the rest of the code mostly belongs to Mitsuba2 and only small parts that activate the iridescent effects were created by us.

To make matters more simple, we created a forked repository on `https://github.com/marcel1hruska/mitsuba2`. The user needs to checkout the `iridescence` branch and compile it according to the Mitsuba2 documentation [18].

B.3 GGX for ART

The `GGX_ART` folder contains the GGX implementation for ART. To add the functionality, replace the code of ART version 2.0.3 by the contents of this folder. Note that the files also include the implementation of the Blinn microfacet distribution which belongs to ART and is not a part of this thesis.

Unfortunately, we do not have a fork of the repository as the origin is not on github.