FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Comparison of Physically Based Rendering Systems

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Robert Glanz
Matrikelnummer 01329106

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Hiroyuki Sakai

Wien, 26. März 2018

Robert Glanz	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Comparison of Physically Based Rendering Systems

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Robert Glanz
Registration Number 01329106

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Hiroyuki Sakai

Vienna, 26th March, 2018

Robert Glanz | Michael Wimmer

Erklärung zur Verfassung der Arbeit

Robert Glanz
Ortsstraße 17, 3701 Zaußenberg

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 26. März 2018

Robert Glanz

Danksagung

An erster Stelle möchte ich mich bei meinem Betreuer Hiroyuki Sakai bedanken, der mich während meiner Arbeit mit Ratschlägen und richtungsweißenden Entscheidungen unterstützt hat. Er gab mir viel kreative Freiheit bei der Umsetzung meiner Vision und stand mir bei technischen Fragen stets zur Verfügung. Ich bedankte mich außerdem bei Károly Zsolnai-Fehér, wessen inspirierender Vortrag über Rendering, mein Interesse zu diesem Thema geweckt hat. Mein Dank gilt auch Michael Wimmer, welcher mich in seiner Rolle als Supervisor ebenfalls beim Verfassen dieser Arbeit unterstützt hat.

Acknowledgements

First of all, I want to thank my assistant Hiroyuki Sakai, who supported me throughout my work on this thesis. He gave me a lot of creative freedom to realize my vision and he provided valuable advice for the implementation and the written part. I also want to thank Károly Zsolnai-Fehér, whose talk about rendering inspired me to work on this topic. Last but not least I want to thank Michael Wimmer, for the supervision and proofreading of this thesis.

Kurzfassung

In dieser Arbeit wird eine quantitative Evaluation durchgeführt, um die physikalisch basierten Rendering-Systeme zu finden, die am häufigsten in der Wissenschaft verwendet werden. Infolgedessen werden die Renderer Mitsuba, PBRT-v3 und LuxRender miteinander verglichen. Der Vergleich dient dazu eine mögliche Interoperabilität zwischen den Rendering-Systemen zu erforschen. Das Ziel ist es, eine Menge von gemeinsamen Materialbeschreibungen und Lichtmodellen zu finden und die Auswirkungen von dessen Parametern zu analysieren.

Abstract

In this thesis, a quantitative evaluation is performed to find the most relevant physically based rendering systems in research. As a consequence of this evaluation, the rendering systems Mitsuba, PBRT-v3 and LuxRender are compared to each other and their potential for interoperability is assessed. The goal is to find common materials and light models and analyze the effects of changing the parameters of those models.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Overview

Rendering is a part of computer graphics that focuses on generating images from a model. In contrast to photography, the model does not physically exist. The models are described in scene definition files, which are usually text-based.

Rendering is widely used in application areas such as computer games, different kinds of visualizations (e.g., flow visualization, volume visualization), and in the entertainment field (e.g., movies, visual effects, advertisements). Different areas have different requirements. For real-time rendering, the rendering process must be very fast to obtain smooth animations. So, the computer program must render images in milliseconds. In contrast to movies, the image quality is limited based on the computing capacity and the used rendering methods. It is a balancing act for programmers between producing plausible images and considering the rendering time. In movies, the time needed to synthesize the images is not of primary importance. However, the realism of the resulting images must be high. Physically based rendering is the scientific area where scenes are rendered with algorithms that generate graphics in a way that more accurately resembles the flow of light in reality.

Another important feature of a rendering system is its usability. In the scientific area, programs are not necessarily developed with usability in mind. That area focuses primarily on implementing the newest and most powerful algorithms. For rendering systems used in the industry, usability is important because users rely on these systems on a daily basis in a production context.

There are lots of different rendering systems. As a scientist, it is not trivial to select a suitable system for the scientific work. The decision has to be based on several factors, such as the feature set, the performance, and the scientific context. In this work, a quantitative overview of the most relevant rendering systems is given.

Another motivation for this work comes from the fact that depending on how the algorithms are implemented, rendering outputs might differ significantly. This work focuses on the question if an interoperability between different rendering systems can be accomplished and what steps are needed to get qualitatively comparable results. This work should give a baseline of material descriptions and methods to generate similar outputs and provide an answer to what commands achieve this goal.

Some rendering systems (e.g., Mitsuba) describe the scenes in XML-based files and others (e.g., PBRT) describe the scenes by a state-based logic, like some graphics APIs. This work also answers the questions what the equivalent commands for making similar outputs are and how to write them in the description files.

Not all systems have the same feature set. This work gives detailed information about some common materials and lights, and how to use them in a way that enables comparison.

1.2 Structure of the Work

In Chapter 2, some state-of-the-art rendering systems, which are used in the sciences and in the industry, are explained. In Chapter 3, a quantitative evaluation is done to provide an explanation regarding the prevalence of certain rendering systems.

As a result of this quantitative evaluation, the rendering systems Mitsuba, PBRT-v3, and LuxRender are described in more detail and in Chapter 4, they are compared to each other in a more detailed way.

These systems utilize different types of materials and lights. In Chapter 4, some common features are compared to each other and explanations about how to replicate rendered images across the rendering systems are provided. Furthermore, the performance of the three systems are tested to provide additional insights into the performance of the systems relative to each other.

CHAPTER 2

Related Work

In the field of physically based rendering, there are various renderers that can be used to create photorealistic images. In the scientific area, the most commonly used systems are Mitsuba [Jak10], LuxRender [VRB+08], and PBRT-v3 [PJH16].

In Section 3, a detailed description of the selection process is given to answer the question why these systems are used for the comparison. To give an overview about the three rendering systems, the main features and differences will be explained in the following section.

2.1 Mitsuba

Mitsuba [Jak10] is a rendering system that is based on the concepts of PBRT [PJH16] and is written in C++. Mitsuba focuses on scientific applications. So, the program is structured in a modular way, to make it easily extendible. Mitsuba provides a command-line interface as well as a graphical user interface. The GUI application is not a tool for generating scenes. It can only be used to open scene descriptions and change rendering settings. One additionally feature of the GUI application is that it can be used to interactively change the camera parameters. The scene description can be written in XML-based documents. It is also possible to convert Collada [Grob] DAE files to XML files, and Mitsuba can be integrated into the Collada workflow. Mitsuba supports Windows, Linux, macOS and is optimized for x86 and x64 platforms. The rendering system provides implementations of state-of-the-art global illumination algorithms like Metropolis Light Transport [VG97] and Stochastic Progressive Photon Mapping [HJ09]. Mitsuba also can render participating media and supports spectral rendering.

2.2 PBRT-v3

PBRT-v3 is a rendering system designed and implemented by Matt Pharr, Greg Humphreys and Wenzel Jakob. The software is the implementation of the concepts presented in the book Physically Based Rendering: From Theory to Implementation [PJH16]. The book is written in the style of literate programming. This method is based on a combination of documentation and implementation, facilitating understandability. In January 2014, Matt Pharr and Greg Humphreys were awarded with the Scientific and Technical Academy Award for the book[1].

The goal of PBRT is to give a design and a concrete reference implementation that should be complete, illustrative, and physically based. In 2004 the first edition of the book was published and was followed by the second edition in 2010. The third version was published in 2016 and included state-of-the-art rendering algorithms such as Metropolis Light Transport and methods to render participating media. In this version, ideas from Mitsuba were adopted and integrated. So, these two rendering systems are based on each other. PBRT is written in C++, and the scenes are described in text files that are generally structured in two sections. The first section holds the information about the camera, film, sampling, and light transport methods, and the second section contains the information about the materials, lights, shapes, and textures. More details are provided in Section 4.

PBRT has some exporter implementations to support other scene files. For example, a Cinema 4D exporter and a Wavefront OBJ exporter which allows exporting scenes from Blender.

2.3 LuxRender

LuxRender [VRB+08] is a physically based rendering system that is also based on PBRT [PJH16]. It was released in 2008 and it is developed by a team of about 15 people. Its original focus was to make the PBRT rendering system more usable for non-technical artists and also to make the project open to third-party applications. LuxRender is free and the developers want to keep it free in future. LuxRender is a member of the Software Freedom Conservancy [ACG+]. So, LuxRender can be used by research teams to implement new methods and algorithms.

LuxRender has many contributors, who helped to integrate LuxRender in 3D software like Blender, Cinema 4D, Maya, and more. A key feature of LuxRender is that the rendering process can be paused and continued at any time. The current state can be written into a file for previews. This fact makes it more usable than many other global illumination systems.

[1] 19 Scientific and technical achievements to be honored with Academy Awards: http://www.oscars.org/news/19-scientific-and-technical-achievements-be-honored-academy-awardsr/

The scene description is similar to that in PBRT, but not equivalent. So, a direct usage of PBRT scene files is not possible. At the current state, LuxRender supports Bidirectional Path Tracing, Path Tracing, Particle Tracing, and Photon Mapping.

LuxRender has a GUI application that provides a user friendly interface for artists. For example, it is possible to change lights in the GUI dynamically. In the LuxRender GUI, it is also possible to manually specify which regions should be sampled more often.

To provide a broader overview of rendering systems, the following two sections describe other systems that are used in science and in the industry.

2.4 Other Scientific Rendering Systems

In this section, state-of-the-art rendering systems that are primarily used in the scientific area are briefly explained.

2.4.1 Tungsten

The Tungsten rendering system [Bit14] is a physically based renderer developed by Benedikt Bitterli and was published in 2014. This system is written in C++ and uses the Intel Embree ray tracing library.

Tungsten supports state-of-the-art integrators like Primary Sample Space Metropolis Light Transport and Progressive Photon Mapping in addition to the basic integrators.

Unfortunately, Tungsten has a lack of documentation, which leads to decreased usability. So, the software is currently targeted at academic usage and not at production purposes.

2.4.2 Lightmetrica

Lightmetrica [Ots15] is a system for scientific usage written in C++ and is developed by Hisanari Otsu (2015). It currently supports Path Tracing, Light Tracing and Bidirectional Path Tracing.

The scene description is written in YAML [Eva], which makes it human-readable and editable in a simple way. The focus lies on modifying the scenes by hand.

2.4.3 Taichi

Taichi [Hu17] is a physically based computer graphics library written in C++. One focus of this project is the usability. So, the kernel of Taichi is implemented in C++, but for fast scripting it has a Python wrapper.

It is open source, which makes it easily accessible for research usage, and Yuanming Hu implemented about 40 graphics papers, e.g., different Metropolis Light Transport integrators and physically based animation algorithms, for Taichi.

2.4.4 appleseed

appleseed [BTB17] is a global illumination system by François Beaune, Eseban Tovagliari, and Luis Barrancos. It is open source and designed for animations and visual effects. It supports the integrators Bidirectional Path Tracing, Stochastic Progressive Photon Mapping, and Light Tracing. appleseed provides a large set of material models. Additionally, this system supports fully programmable shading via Sony Pictures Imageworks Open Shading. The software is written in C++ and Python.

2.5 Rendering Systems Used by the Industry

In this section some global illumination systems used by the industry are shortly presented.

- Hyperion: Physically based rendering system created by Walt Disney Animation Studios for rendering animated movies [Stub].
- RenderMan: Renderer created by Pixar Animation Studios for rendering visual effects in movies [Stua].
- Arnold: Physically based rendering system created by Solid Angle, which was bought by Autodesk in 2016, for rendering visual effects in movies [L.].
- Iray: Physically based rendering system created by Nvidia for rendering 3D objects. It is integrated in 3D modelling software packages like 3ds Max, Cinema 4D, and Maya [Cor].
- V-Ray: Physically based rendering system created by Chaos Group for rendering visual effects in advertisements, architecture, and movies. It can be integrated as plug-in in 3ds Max, Maya, Blender, Nuke, SketchUp, Revit, Cinema 4D, and Modo [Groa].
- Corona: Physically based rendering system created by Render Legion available for 3ds Max, Cinema 4D, and as a CLI application [a.s]. The renderer was originally a student project of Ondřej Karlík. Together with Adam Hotový, and Jaroslav Křivánek the project evolved to a commercial software.
- Mental Ray: Renderer with global illumination support created by Mental Images GmbH. The company was bought by Nvidia in 2007 [Gmb].
- Radeon ProRender: Physically based rendering system created by AMD. It can be integrated in 3ds Max, Maya, Blender, and Cinema 4D [AMD].

CHAPTER 3

Selecting Relevant Rendering Systems

Chapter 2 provided an overview of some state-of-the-art rendering systems used in the sciences and in the industry. For future work in the field of global illumination, it is important to answer the question which rendering systems are primarily used by scientists in the field of physically based rendering.

To answer this question, a program that collects all scientific papers that are published in one of the following digital libraries and are related to global illumination was implemented:

- ACM Digital Library [fCM]
- Eurographics: Computer Graphics Forum [Ass]

3.1 Implementation of the Process

For automatically retrieving relevant information about the papers published in the aforementioned digital libraries, a Java program was implemented that is able to parse and browse websites. For this purpose, the program uses three external dependencies:

- H2, which is used as a database for storing paper information,
- Selenium WebDriver, a programming interface for browsing and parsing websites to get paper information, and
- PDFBox, which is used to automatically search in papers for rendering systems.

monte carlo rendering	path tracing
ray tracing	metropolis light transport
volumetric path tracing	photon mapping
offline rendering	photorealistic rendering
markov chain monte carlo rendering	vertex connection and merging
physically based rendering	bidirectional path tracing
rendering equation	importance sampling

Table 3.1: Used keywords for searching the papers

The program consists of three steps. The first step is to open the specific paper URLs using Selenium WebDriver and to retrieve every interesting piece of information. For example, the title of the paper, the names of the authors, the year the paper was published, the download link for downloading the PDF file, and the used search keyword for finding the paper. In the second step, the papers are downloaded by another Selenium script. The addition of the second step is useful because the downloading process would need a long time, which could lead to breaks in the parsing process. Another advantage is the fact that the Selenium WebDriver can send commands to real web browsers (e.g., Google Chrome). This makes it possible to supply the download link and automatically download the files into a specified folder. The last step is to search the names of the rendering systems in the papers and to classify the papers accordingly.

The parsing process of the first step starts by opening the digital library URLs. The URLs include all information for searching the papers, for example the search keyword and the number of the subpage. The number of the subpage is needed because the libraries only display a fixed number of papers on one page (similarly to the display of Google search results, for instance). Two special cases can occur after loading the website. The first case could be that the website is not fully loaded. We implemented a search method that tries to find web elements for a certain time. If one of the elements is missing after a specified time period, the website will be reloaded and the search process starts again. The second case could be that the end of the subpages is reached. So, in the parsing step, the procedure also searches for the `next page` link. If this link does not exist or is disabled, the last page of the results for a specific search keyword is reached and the process starts again by using the next search keyword. The keywords for searching the papers in the online digital libraries are listed in Table 3.1.

In the third step, the names of the rendering systems are searched in the papers. This step is done by using the PDFBox Java library. In Chapter 2, the names of the searched rendering systems are listed. The classification of the hits is done manually. For instance, hits which do not correspond to a rendering system are deleted (e.g., Arnold can be a name of an author or the rendering system). To assign the paper to a specific subfield of global illumination, all the abstracts were read and the rest of the papers were skimmed.

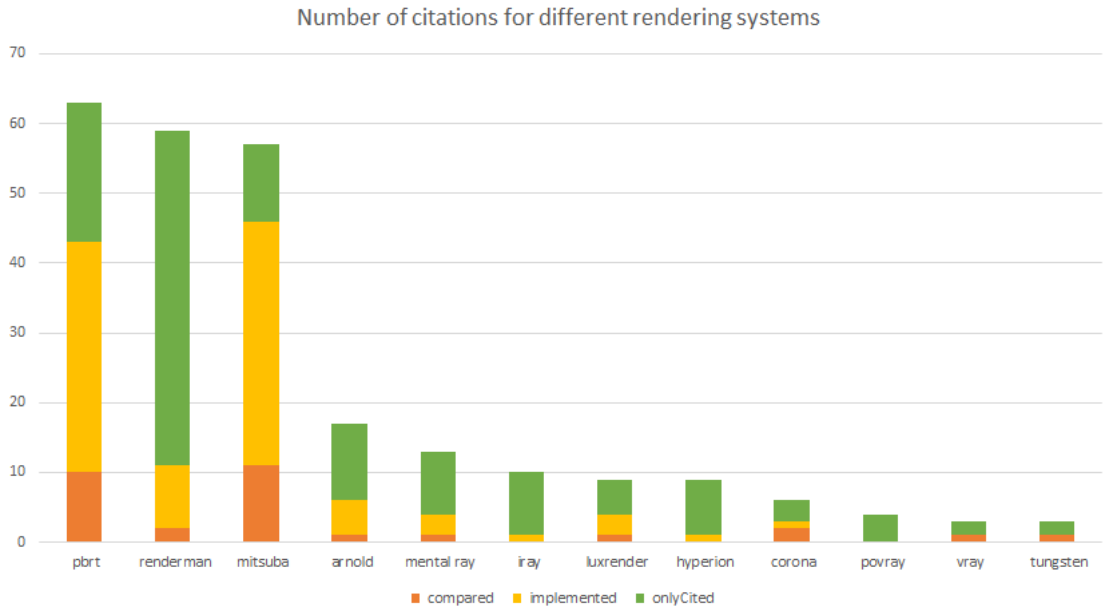

Figure 3.1: Number of citations for different rendering systems

3.2 Evaluation of the Selection Process

In this section, the results of this chapter are presented. The paper collection process collected 1687 papers in total. These papers were found based on the keywords listed in Table 3.1. So, there are also some papers found which are not related to at least one rendering system listed in Chapter 2. The evaluation of 1687 papers would have lasted a long time. Therefore, the papers that are not related to at least one specific rendering system were filtered out. In 251 papers at least one renderer is mentioned. These papers were evaluated by hand. During this step, the hits were classified as one of the following types: `implemented`, `compared`, and `only cited`. `implemented` means that the implementation of the presented method was based on one of the rendering systems mentioned in Chapter 2. `compared` means that the results of the presented method are compared to those of a rendering system mentioned in Chapter 2. `only cited` means that the rendering system is mentioned incidentally.

Some rendering systems mentioned in Chapter 2 were not mentioned in the found papers. Scientific renderers that had not been cited are Lightmetrica, Taichi, and appleseed; and Radeon ProRender is the only production rendering system that was not mentioned.

In Figure 3.1, the classified hits per rendering system are plotted. It can be seen that PBRT has the most hits, followed by RenderMan and Mitsuba. The rest of the systems are mentioned in less than 20 papers. LuxRender is the third most cited scientific renderer. RenderMan is a renderer used in the industry for more than two decades. RenderMan's ratio between `only cited` and `implemented` is very high, i.e., RenderMan is often

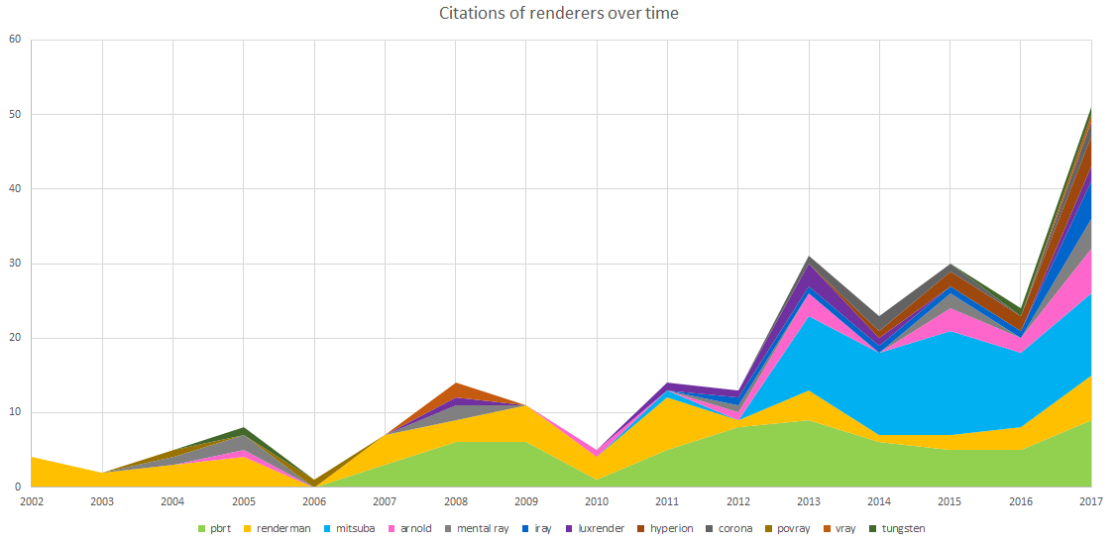

Figure 3.2: Citations of renderers over time

cited by papers, but it is not often used as a base implementation. One reason for that is that in the introduction sections of many papers, some rendering systems are cited to provide a reference, and because RenderMan is available for such a long time, these citations add up to a great number. It can be seen that in the last years, the number of citations of RenderMan are not significantly higher than those for the other rendering systems. This fact can be seen in Figure 3.2. Mitsuba and PBRT are more often used as a base implementation. This quantitative evaluation shows that these rendering systems are often used in research and that it makes sense to achieve interoperability between Mitsuba and PBRT. Interoperable systems are easier to compare to each other, which in turn would allow scientists to draw more accurate conclusions about the performance of a particular method in a shorter amount of time.

LuxRender is a popular renderer for artists, which is often used in connection with Blender, and it is also the third most cited scientific renderer. So, it is also a reasonable choice. LuxRender is based on PBRT, and the differences between PBRT and LuxRender are negligible (see Chapter 4). The advantage of LuxRender is that it has a better integration in Blender and a user-friendly graphical user interface.

To give a further argument to evaluate the differences and accomplish interoperability between PBRT and Mitsuba, Figure 3.2 shows the number of citations of the rendering systems over time. The chart starts with the year 2001, because the only rendering system cited before is Renderman (6 hits from 1994 to 2001). This figure shows that since the year 2013, Mitsuba gained a lot of interest. PBRT was released in 2004, and since 2007, the renderer is constantly used in research.

CHAPTER 4

Evaluation

In this chapter, the three rendering systems Mitsuba, PBRT-v3, and LuxRender will be compared to each other. This selection comes from the quantitative evaluation in Section 3.2, which resulted in the conclusions that Mitsuba and PBRT are the most used rendering systems for implementing new methods and LuxRender is the third most cited scientific renderer. Common material models and lights are evaluated and the differences will be explained.

4.1 Implementation

To be able to test different settings for different rendering systems and automatically compare the results, a Java program that helps to start the rendering processes, measure the rendering time, and compare the results in a single image was written. The tests were performed on a Windows PC with an Intel i7 870 CPU with 8GB of RAM.

The implementation is written in pure Java, i.e., the code does not depend on external libraries. For starting the rendering process of the different rendering systems, the Java `ProcessBuilder` was used. With this class, it is possible to run commands in the operating system console.

For making the call command simple, the Windows Environment variable is configured for Mitsuba and PBRT. The scene files are named with a simple naming convention to make the call even more generic and simple.

To make the comparison possible, all rendering systems are using a similar output `film`. The `film` parameter is used to specify characteristics of the generated image, like resolution, filename, and fileformat. In Mitsuba, there are three different types of films: `hdrfilm`, `tiltedhdrfilm`, `ldrfilm`. The film which is able to output a PNG image is `ldrfilm`. Mitsuba and PBRT-v3 use a simple tone mapper by default that converts the values to a reasonable, displayable range. Tests have shown that the output

of Mitsuba and PBRT-v3 is equal with this default setting. LuxRender is based on PBRT, but the `fleximage` film of LuxRender is not able to output the same images. All three tone mappers of LuxRender (`linear`, `contrast`, and `reinhard`) were tested. With the `linear` setting, it is possible to specify the tone mapper settings manually, but it is generally not possible to produce the same images as in Mitsuba and PBRT-v3. The best and simplest strategy is to set the output format of all three renderers to OpenEXR. This is a fileformat that has a high color depth and is able to store high-dynamic-range images. So by using this fileformat, only the resolution and the filename have to be specified. Mitsuba can output this fileformat with the `hdrfilm`, PBRT-v3 by setting the file ending to .exr, and in the LuxRender scene file the `write exr` setting of the `fleximage` has to be set to true. The conversion of OpenEXR to PNG is done by a tonemapping program, which is included in Mitsuba (`mtsutil tonemap`).

Another important setting in Mitsuba is `fovAxis`. This setting is set to `x-Axis`, which leads to problems if the outputimage has a bigger width than height. So, it is important to set the `fovAxis` to `smaller` for those cases.

To compare the results, the Java program joins the output images to a big image and adds three images, which show the differences, the runtime, and the total difference in percentage. So, the output is one big image that has three rows and three columns. The third column is always the difference from the two images left to it (e.g., first row: Mitsuba output, PBRT-v3 output, RMSE difference of Mitsuba and PBRT-v3). In Figure 4.1 an example output image can be seen.

4.1.1 Color Difference

In the output image, three different difference images are concatenated (see previous section). These images always represent the differences between the two images left to them. The color difference measure can be set to CIEDE2000 [SWD05], root-mean-square error (RMSE) or absolute difference (ABS). The values of the differences are scaled to the interval [0, 255]. So, a completely black image means that the two images are the same and a white difference image means that the images are inverse.

In the description area on the bottom of the difference image, all three color difference measures are displayed in percent. The first color measure is the absolute difference (ABS), which is given by the equation

$$distance_{abs} = \frac{|r_1 - r_2| + |g_1 - g_2| + |b_1 - b_2|}{3}, \tag{4.1}$$

where r_1, g_1, b_1 are the color channels of one pixel in the first image, and r_2, g_2, b_2 are the color channels of the corresponding pixel in the second image.

The second distance measure is the RMSE, which is given by the equation

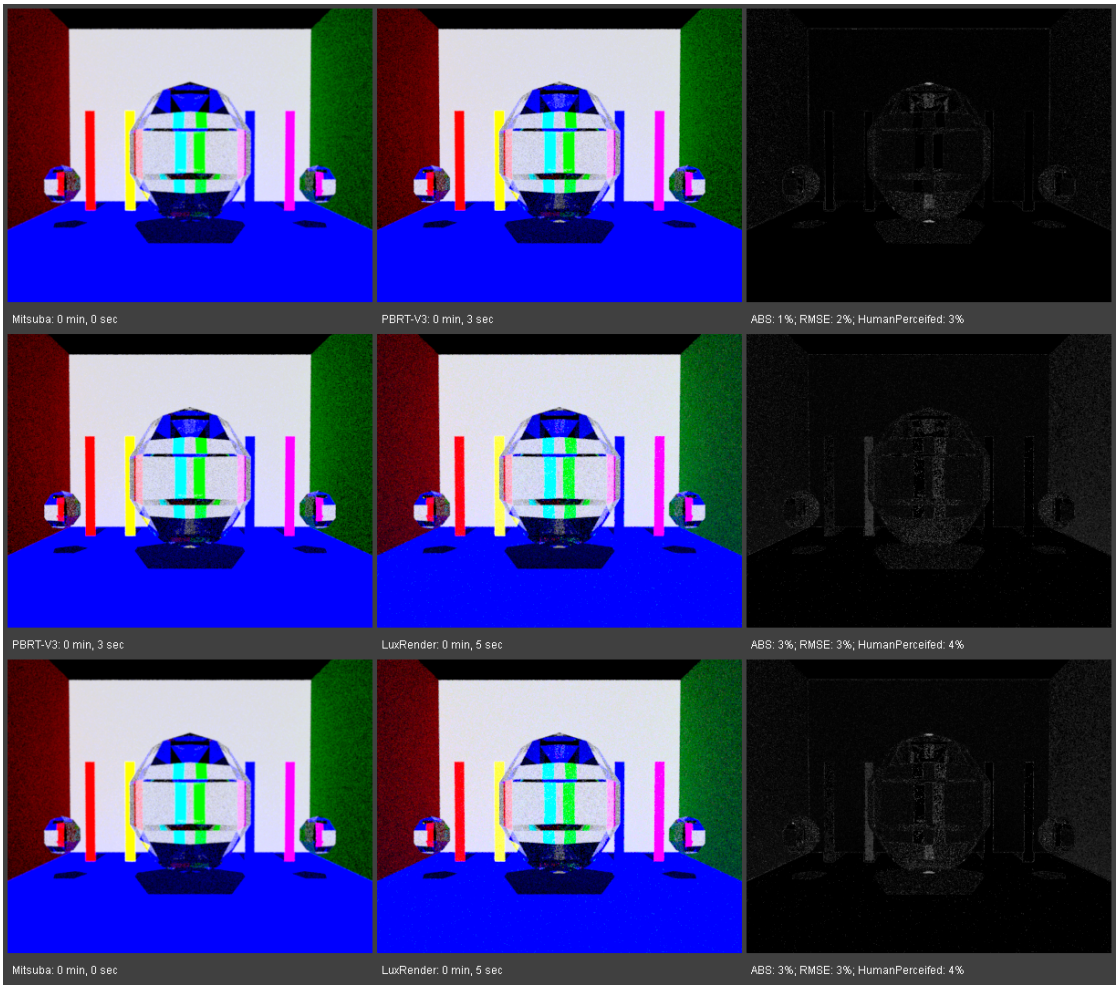

Figure 4.1: Output image generated by the implemented program demonstrating the layout

$$distance_{rms} = \sqrt{\frac{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}{3}}. \tag{4.2}$$

The third distance measure is the CIEDE2000 color difference [SWD05]. This measure is based on the LAB color space, where `L` is the luminance, `a` represents the position between red and green, and `b` is the position between yellow and blue. The basic idea of this distance measure is to use the RMS value in the LAB color space (CIEDE76). The CIEDE2000 has some weighing factors and fixes in its formula, which makes it to the most accurate color distance measure with respect to human color perception.

These distance measures are applied to all pixels in the image. The resulting image is the difference image. To reduce the difference image to a single value, the average of all

pixels of that image is calculated. This value represents the overall difference between two images.

4.2 Describing the Scenes

In this section, the basic setup for creating similar images in LuxRender, Mitsuba and PBRT-v3 is described. The first important setting is the `film` parameter. By using the OpenEXR outputs of the renderers, the external Mitsuba tone mapping software `mtsutil tonemapper` can tone map all outputs with the same parameters. So, it is not a requirement to set the tone mapper in the scene description files correctly. Directly setting up the tone mapping parameters in the scene description files would be time intensive because in LuxRender, differences in light intensities lead to corresponding adaptations in the resulting output intensities.

The second step is to choose the same integrator and sampler. This step should be no problem, because all three rendering systems support many different samplers and integrators. The names and the settings for the samplers and integrators are almost identical. The evaluation of the lights and materials was done by using the standard path tracer and a random sampler.

The third step is to define the correct camera perspective. All three rendering systems have similar options for specifying the camera. For example, all three have `lookAt` commands and `matrix` commands (perspective matrix). The simplest way to adjust the systems with respect to each other is to use the `lookAt` command. The `lookAt` commands are equal in Mitsuba and LuxRender. For PBRT-v3, the x-axis has to be flipped. This flipping can be done by using a scale of `(-1, 1, 1)`.

To load objects into the scenes, it is recommended to use PLY files. Mitsuba supports OBJ and PLY files, but PLY files are recommended. PBRT-v3 and LuxRender also support PLY files, which makes it possible to load the same models in all systems.

Loading the textures can be done by using the TGA file format, because all three rendering systems support it and the popular PNG file format is not supported in PBRT-v3. Since the x-axis points in the opposite direction compared to Mitsuba and LuxRender, it is necessary to flip the `v` coordinate by multiplying it with -1. This can be done by using the `vscale` setting of the `Texture` command as follows.

```
Texture "texraw"
        "spectrum" "imagemap"
        "string filename" [ "Textures/rainbow-tex-simple.tga" ]
         "bool trilinear" [ "true" ]
         "float vscale" [-1.0 ]
```

Another important aspect is to understand the scene descriptions of the rendering systems. For positioning, the multiplication order is important. To position objects equally across systems, the order of the transformation commands matter. If the positioning is done by using a matrix, no further adjustments have to be done.

Below, code examples for equal positioning of two planes.

```
# PBRT and LuxRender:
NamedMaterial "tex_wood"
Identity
Translate 0 1.6 0
Rotate 90 1 0 0
Rotate 90 0 0 1
Shape "plymesh" "string filename" [ "Models/plane.ply" ]

# Mitsuba:
<shape type="ply">
        <string name="filename" value="Models/plane.ply"/>
        <boolean name="faceNormals" value="true"/>
        <boolean name="flipNormals" value="false"/>
        <ref name="bsdf" id="tex_wood"/>
        <transform name="toWorld">
                <rotate x="1" angle="90"/>
                <rotate y="1" angle="90"/>
                <translate x="0" y="1.6" z="0"/>
        </transform>
</shape>
```

4.3 Surface Scattering Models

In this section, material models that are common in all three rendering systems are evaluated and compared to each other. The systems have different names for basically the same materials. These materials are:

- Diffuse
- Metal
- Glass
- Plastic

The other materials of the rendering systems are different regarding their results and their parameters. Therefore, it is not possible to use these materials for producing the same output images in all three rendering systems. Accordingly, the following materials of Mitsuba, PBRT-v3, and LuxRender are not evaluated in this work—Mitsuba: `thindielectric`, `coating`, `roughcoating`, `modified phong BRDF`, `ward`, `difftrans`, `hk`, `irawan`; PBRT-v3: `disney`, `fourier`, `hair`, `translucent`, `uber`; LuxRender: `carpaint`, `mattetranslucent`, `shinymetal`.

Mitsuba: `roughdiffuse`	PBRT-v3: `matte`	LuxRender: `matte`
`reflectance`	`Kd`	`Kd`
`alpha`	`sigma`	`sigma`

Table 4.1: Corresponding parameters of the rough diffuse material in Mitsuba, PBRT-v3, and LuxRender

4.3.1 Diffuse

This material is either based on the simple Lambertian reflection model, or it is based on the Oren-Nayar reflectance model [ON94] depending on the specification. If the material is based on the Lambertian reflection model, the incoming ray is reflected in all directions with the same probability. A real-world equivalent would be a material that looks the same from every angle (e.g., a piece of paper or cotton clothing).

In Mitsuba, there are two different materials for this type, a smooth diffuse material (`diffuse`) and a rough diffuse material (`roughdiffuse`). Another Mitsuba specific is that the material is only one-sided. So, if the camera points at the back face, the material appears completely black.

The diffuse material in PBRT-v3 and LuxRender is named `matte`. In contrast to Mitsuba, the roughness factor (Oren-Nayar [ON94]) is called `sigma`. All systems support a single color specification (rgb) or a texture which uses the object's `uv`-coordinates for specifying the colors on the surface. The corresponding parameters for the rough diffuse material are listed in Table 4.1.

The diffuse material has some differences in the evaluated rendering systems. The first distinctive difference is that the colors generated by LuxRender are different compared to those generated by Mitsuba and PBRT-v3. Fig. 4.2 shows the differences between PBRT-v3 and LuxRender. The two systems are based on the same code (PBRT-v1), but the results are different. It should be noted that the differences are pronounced differently for the three color channels. Fig. 4.2 shows an amplified RMS difference image that demonstrates this effect. It can be seen that LuxRender generates darker shades of blue for surfaces that are close to perpendicular relative to the camera (top row). The images with the red objects demonstrate differences in the bottom part of the sphere-shaped objects. The image generated by PBRT-v3 is darker in those regions. This effect is perceivable without looking at the difference image. In the image with the green objects, the difference in the middle and upper part is not perceivable without looking at the difference image, but in the difference image, the difference can be seen clearly.

If the roughness parameter is specified (not 0), it can also lead to perceivable differences. The difference between LuxRender and PBRT-v3 is not significant. Only the images generated by Mitsuba are significantly different. The theoretical background for this parameter is provided by the Oren-Nayar model [ON94]. The parameter has to be specified a little bit differently for the systems. In Mitsuba, the `alpha` parameter is specified in radians, and in LuxRender and PBRT-v3 the similar `sigma` parameter is

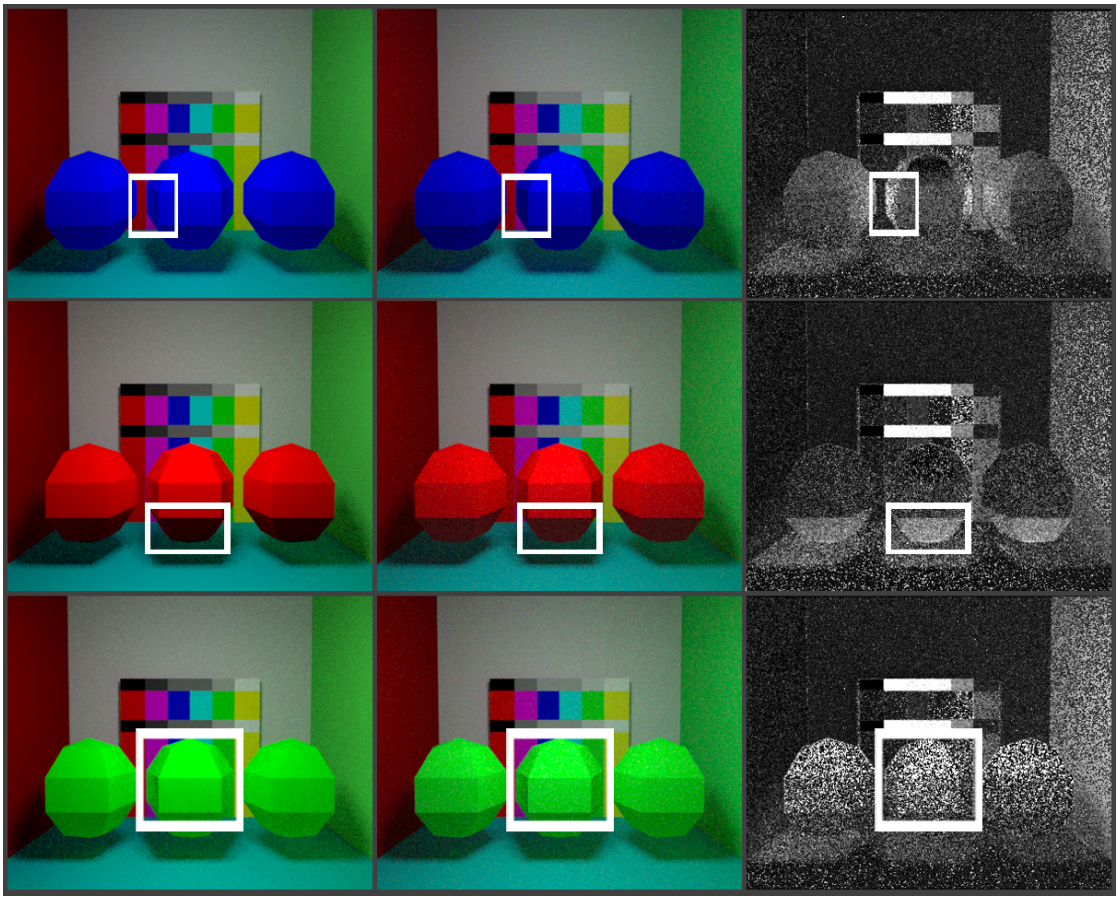

Figure 4.2: Comparison of PBRT-v3 (left column) and LuxRender (middle column). The right column consists of RMSE difference images.

specified in degrees. Another difference is that the surface colors in Mitsuba get brighter than in PBRT-v3 and LuxRender as the angle between the surface normal and the viewing direction gets bigger. In the marked areas of Fig. 4.3, these differences can be seen. The objects in this scene are coloured with the following intensities: 0.1 (left), 0.5 (center), and 1.0 (right). The effect is most pronounced on the center object.

In summary, the diffuse material is a little bit different in the three rendering systems. Most of the differences are small enough to not influence the overall look of the images. When trying to make the diffuse material of the rendering systems interoperable, the effect seen in Fig. 4.2 can be a problem.

4.3.2 Metal

The metal material model is used for creating objects that look metallic. On metal materials, the outgoing direction of a ray hitting the object is not completely random

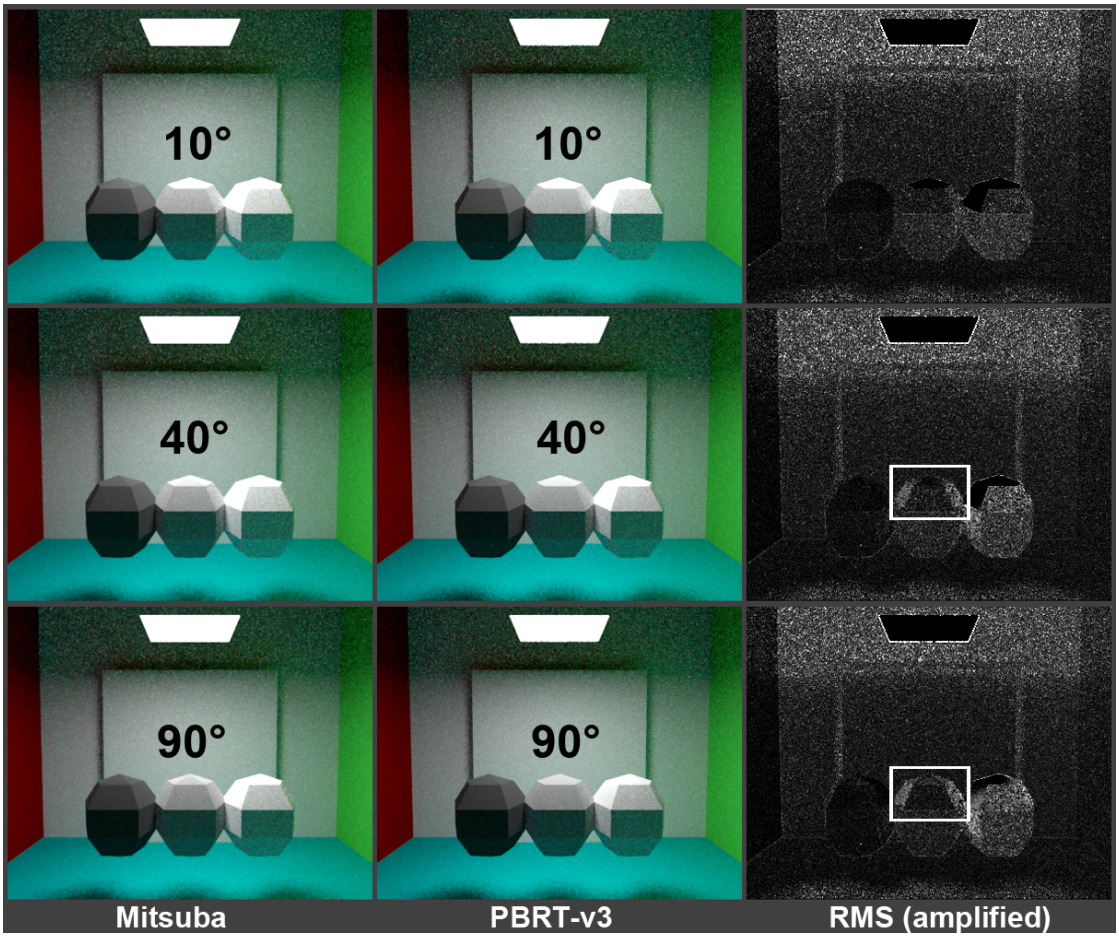

Figure 4.3: Roughness parameter comparison of Mitsuba and PBRT-v3

as it would be on a diffuse material, but the outgoing direction is also not the perfect reflection direction. The outgoing direction is different for different wavelengths of the light hitting the object. This effect can be described by the index of refraction (IOR). Another effect is that light rays with different wavelengths are absorbed differently.

In Mitsuba, there is a differentiation between rough metals and smooth metals. In PBRT-v3 and LuxRender, there is only one type, which is called `metal`. The corresponding parameters of the rough metal material is listed in Table 4.2. In Mitsuba, it is possible to specify a concrete distribution for specifying the surface roughness:

- Beckmann: Physically based distribution derived from Gaussian random surfaces [BS87]
- GGX: Distribution by Alter et al. [WMLT07]

Mitsuba: `roughconductor`	PBRT-v3: `metal`	LuxRender: `metal`
`distribution`		
`alpha`	`roughness`	
`alphaU`, `alphaV`	`uroughness`, `vroughness`	`uroughness`, `vroughness`
`material`	`eta`, `k` (metal name)	`name` (metal name)
`eta`, `k` (filename or values)	`eta`, `k` (filename, or values)	`name` (nkdata file)
`extEta`		
`specularReflectance`		

Table 4.2: Corresponding parameters of the metal material in Mitsuba, PBRT-v3, and LuxRender

- Phong: Distribution by Phong et al. [Pho75] adapted to a BRDF for physically based rendering
- AS: Anisotropic Phong-style microfacet distribution proposed by Ashikhmin and Shirley [AS00]

Mitsuba supports specifying the roughness with two parameters `u`, `v`, when the distribution is set to Anisotropic Phong-style AS (from Ashikhmin, Shirley [AS00]), otherwise the roughness can only be specified with one parameter `alpha`. In PBRT-v3, you can choose between specifying the roughness in general, or specifying it differently for the object's `u` and `v` directions. The parameter is equal in LuxRender and PBRT-v3 and is called `roughness`.

In all three rendering systems, it is possible to use built-in metals like copper, gold, and aluminium, or to specify the index of refraction (IOR) and the absorption coefficient manually by referring to a file.

If the user wants to use a built-in metal, the following parameters have to be set to a string value that indicates the material—Mitsuba: `material`, PBRT-v3: `eta` and `k`, LuxRender: `name`.

If the user wants to specify the material manually, in PBRT-v3 and LuxRender the same aforementioned parameters have to be used. In Mitsuba, the `eta` and the `k` parameters have to be used instead of the `material` parameter. In contrast to the built-in metal specification, the string values of the parameters have to be references to files that describe the metal. In Mitsuba and PBRT-v3, the so-called `.eta` and `.k` files, and in LuxRender the `nkdata` files must be used. These files are structured as tables, in which different wavelengths, different IORs, and absorption coefficients are listed. Because LuxRender uses a different file format, it is not possible to use the same files for all three rendering systems. Fig 4.4 is rendered with the built-in metal gold.

During the evaluation of the metal model, differences became apparent. The first step of making the metal material model in Mitsuba look similar to the model in PBRT-v3 and

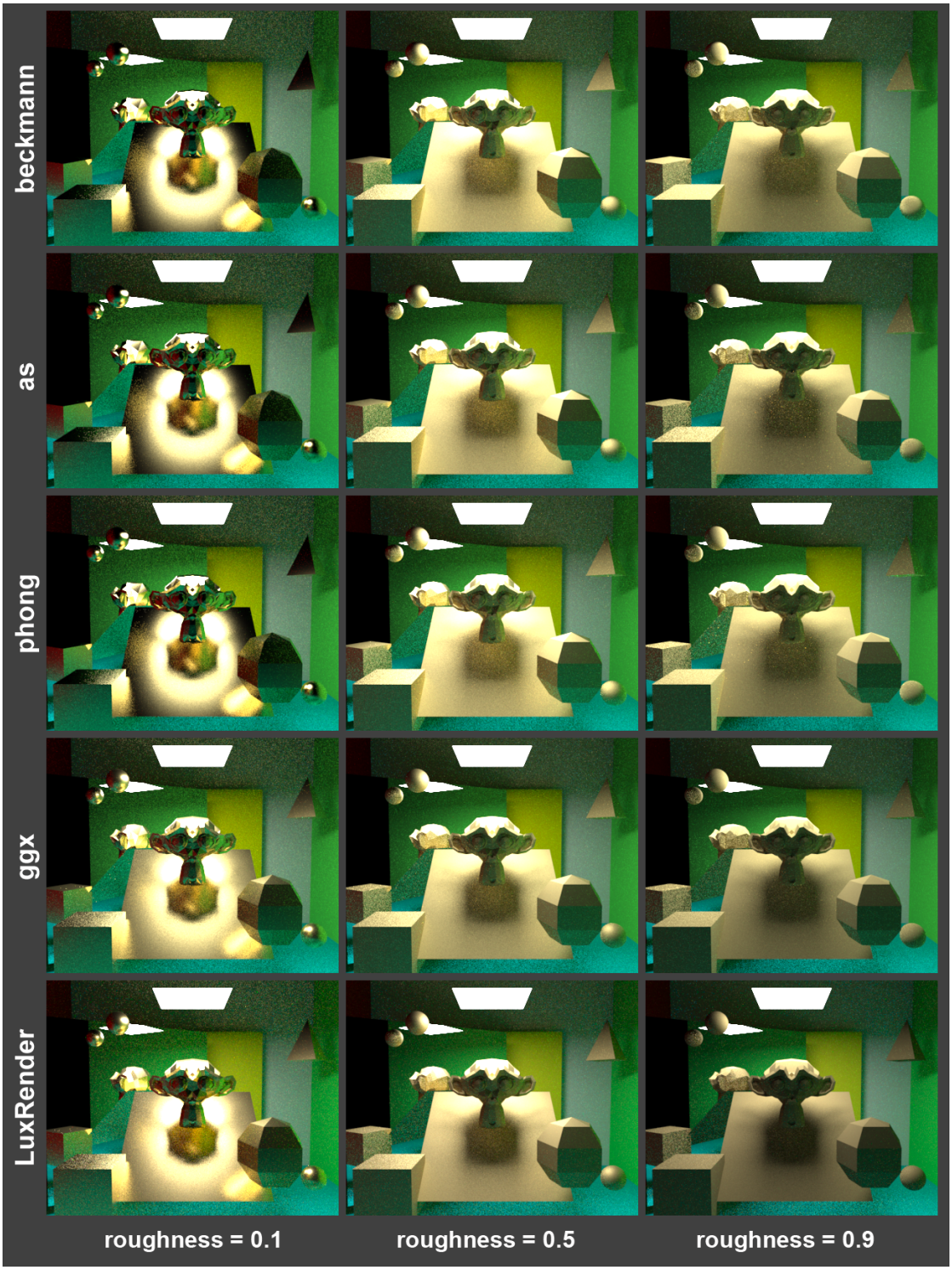

Figure 4.4: Mitsuba's different roughness distribution types compared to PBRT-v3 and LuxRender

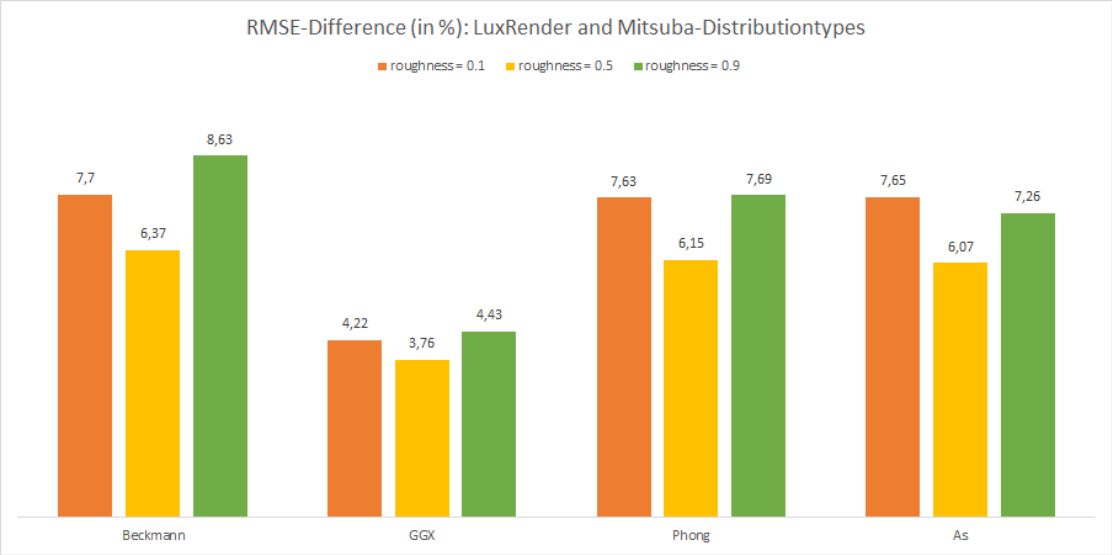

Figure 4.5: RMSE (in percent) of LuxRender and different Mitsuba distribution types (roughness factor was set to 0.1)

LuxRender was to find the right roughness distribution type in Mitsuba. The first four rows of Fig. 4.4 show images rendered using the different distribution types of Mitsuba. If the roughness factor is set the same in PBRT-v3 and LuxRender, the effect is almost the same. For the sake of a reference, the last row of the image shows the output of LuxRender. The columns represent the different roughness factors (left: 0.1, middle: 0.5, right: 0.9). The results generated using the GGX distribution looks most similar to LuxRender's output. For a quantitative evaluation, the RMSE between LuxRender and the Mitsuba distribution types are plotted in Fig. 4.5. In the plot, the hypothesis that the GGX distribution is most similar to LuxRender's distribution is strengthened. The plot also shows that a roughness factor of 0.5 results in a smaller difference compared to factors of 0.1 and 0.9.

The angle between the surface normal and the viewing direction has a significant influence on the result. Therefore, the test was performed with differently rotated planes (center plane). The result was always the same. The GGX distribution is the most similar looking distribution.

Another measured feature of this test was the performance. Table 4.3 shows the rendering time per sample in nanoseconds for the different Mitsuba roughness distributions, LuxRender, and PBRT-v3. This table shows that the rendering time is about one percent higher when the roughness factor is 0.1 compared to a roughness factor of 0.9. The rendering time with PBRT-v3 and LuxRender is 4.4 times and 5.8 slower than with Mitsuba. Another more general performance test is provided in Section 4.5.

Type	tps roughness 0.1[ns]	tps roughness 0.5[ns]	tps roughness 0.9[ns]
Mitsuba: Beckmann	0,87	0,84	0,76
Mitsuba: GGX	0,84	0,82	0,71
Mitsuba: Phong	0,85	0,84	0,76
Mitsuba: AS	0,85	0,84	0,71
PBRTv3	3,76	3,59	3,29
LuxRender	4,89	4,50	4,19

Table 4.3: Time per sample in nanoseconds for different Mitsuba distribution types, LuxRender, and PBRT-v3

Mitsuba	**PBRT-v3**	**LuxRender**
`alphaU`, `alphaV` / `alpha`	`uroughness`, `vroughness`	`uroughness`, `vroughness`
`intIOR`	`eta`	`index`
`extIOR`	fixed: `1` (air)	fixed: `1` (air)
`specularReflectance`	`Kr`	`Kr`
`specularTransmitance`	`Kt`	`Kt`
`distribution`		
		`cauchyb`

Table 4.4: Corresponding parameters of the glass material in Mitsuba, PBRT-v3, and LuxRender

4.3.3 Glass

The glass material can be used to render objects that reflect and transmit light. Mitsuba and LuxRender have two different types of materials for rendering glass objects. Mitsuba has the `dielectric` and `roughdielectric` materials and LuxRender has the `glass` and `roughglass` materials. In Mitsuba, the smooth version is only a simplification of the rough version, which has a faster algorithm. In LuxRender, the parameters differ greatly because the glass material in LuxRender can be used to render thin glass objects like the partly equivalent Mitsuba material `thindielectric`. Therefore, the following test are only based on the rough glass versions. The parameters for the different rendering systems can be seen in Table 4.4.

The common parameters are the roughness factor, the interior index of refraction (IOR) and the specular components. Tests showed that the change of the IOR parameter (range 0.1 - 2.5 tested) results in the same output. The roughness factors have different effects. For instance, in Mitsuba it is needed to specify the distribution type. Fig. 4.6 shows the different distribution types in Mitsuba, compared to the output of PBRT-v3 and LuxRender. This image (Fig. 4.6) is produced by a rectangular area light source and a roughness factor of 0.1. It can be seen that the output of PBRT-v3 is different on the sides of the object: these regions are darker. The most similar output of Mitsuba is generated by the distribution type GGX. GGX produces an image which is similar to

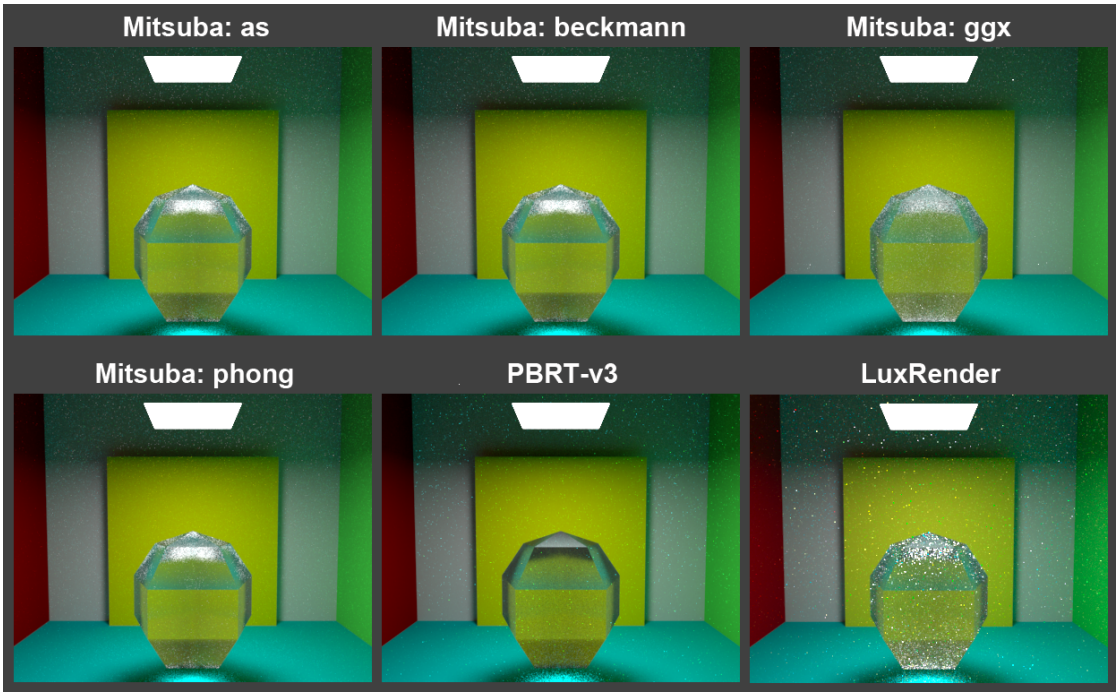

Figure 4.6: Glass (roughness 0.1) with different Mitsuba distribution types compared to PBRT-v3 and LuxRender

LuxRender's result. However, the noise in LuxRender is more severe, and a test with more samples showed that the output of LuxRender is not exactly the same. The result of PBRT-v3 can not be replicated by Mitsuba or LuxRender. So, GGX is a good choice for making the output most similar to the output of LuxRender.

The second test image, Fig. 4.7, compares different roughness factors. Here, only the GGX distribution is compared to the other renderers, because this one is always the most similar one compared to LuxRender. This was confirmed by rendering the scene with different locations of the light object and three different light source types (sphere-shaped area light, rectangular area light, directional light). In all of these tests, the GGX distribution was the most similar one compared to LuxRender. Fig. 4.7 shows that the output of Mitsuba and LuxRender is different at a roughness factor near zero (0.001) and is more similar at factors that are greater than 0.01. The image of PBRT-v3 shows that the reflection at the top of the object is colored gray and not blue like with the other renderers.

For making the glass material models interoperable, the roughness factor should be zero, because this setting results in almost the same outputs in all systems. If the roughness factor is not zero, the output images are not the same. Mitsuba and LuxRender are similar, if the Mitsuba distribution is set to GGX, but the glass objects looking different in PBRT-v3.

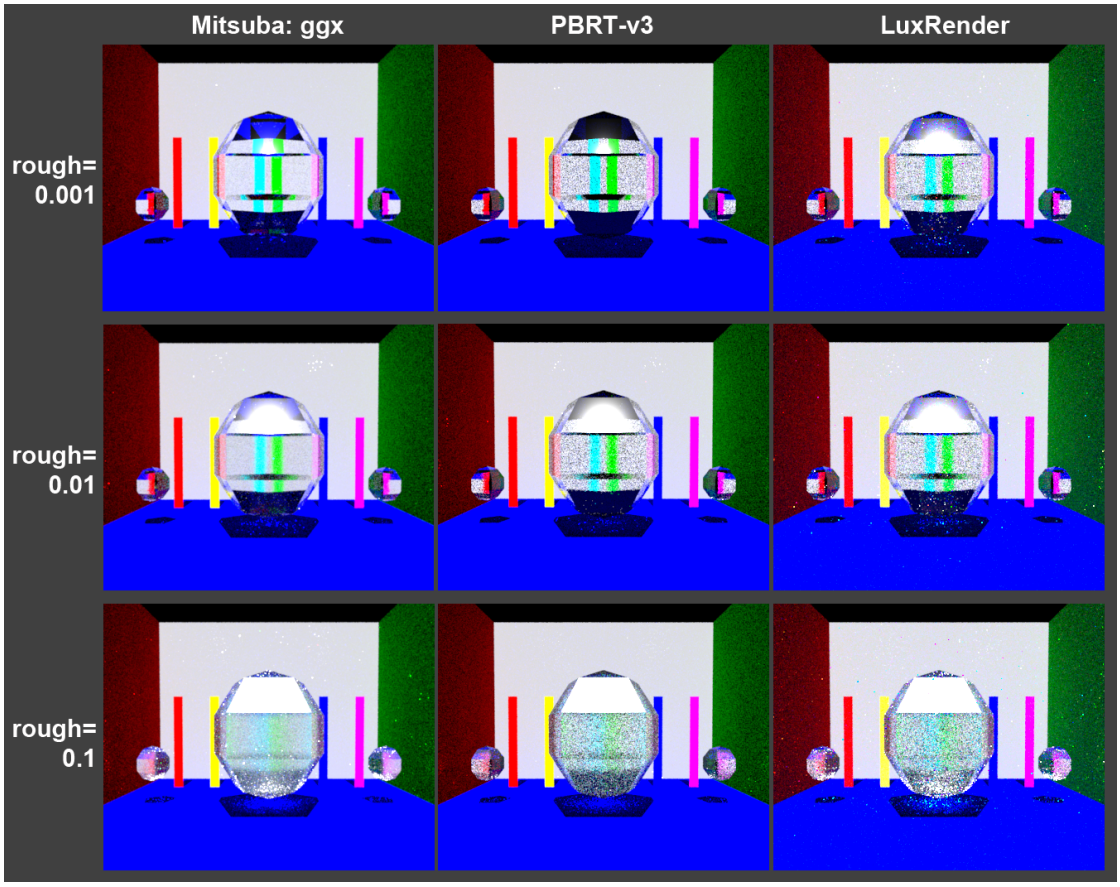

Figure 4.7: A comparison of the glass materials of Mitsuba, PBRT-v3, and LuxRender with different roughness factors (0.001, 0.01, 0.1)

4.3.4 Plastics

In this section, the different plastic materials are evaluated. Plastic materials are used to render objects in a way that the result looks like if they are made of plastic. The names for those types of materials are completely different in all three rendering systems. For finding plastic materials with similar effects, the `plastic`, `roughplastic`, and the `mixturebsdf` materials of Mitsuba; the `kdsubsurface`, `mix`, `plastic`, `subsurface`, and the `substrate` materials of PBRT-v3 and the `glossy`, and the `mix` materials of LuxRender are tested.

The result of this first test was that the PBRT-v3 `subsurface` material is not easily configurable and does not have common parameters with the other materials of the other rendering systems. The simplification of the `subsurface` material is the `kdsubsurface` material. This material is similar to Mitsuba's `roughplastic` and LuxRender's `glossy` material. The parameters for these three plastic materials are listed in Table 4.5.

Mitsuba: `roughplastic`	**PBRT-v3: `kdsubsurface`**	**LuxRender: `glossy`**
`distribution`		
`alpha`	`uroughness`, `vroughness`	`uroughness`, `vroughness`
`intIOR`	fixed: `1` (air)	fixed: `1` (air)
`extIOR`	`eta`	`index`
`specularReflectance`	`Kr`	`Ks`
`diffuseReflectance`	`Kd` (not working)	`Kd`
	`Kt`	
		`Ka`
	`mfp` (not working)	`d`

Table 4.5: Corresponding parameters of the plastic material in Mitsuba, PBRT-v3, and LuxRender

The evaluation of the parameters of the three materials showed that Mitsuba's GGX distribution should be used to achieve the most similar output. Changing the value of the roughness factor does not result in any differences. So, it can be used in all systems with the same value. The only restriction is that Mitsuba does not allow the user to set the roughness separately in the object's `u` and `v` direction.

Another test showed that the `mfp` and `Kd` parameters in PBRT-v3 do not affect the resulting image. So, the `Kd` (diffuse color) parameter was only tested in LuxRender against the corresponding Mitsuba parameter `diffuseReflectance`. This test showed that these two parameters are similar in their results. The specular parameters are equal in all three rendering systems. An example of the test can be seen in Fig. 4.8 (RMSE is used as difference measure).

The test of all materials showed that in PBRT-v3 the `plastic` material is similar to the `substrate` material. The difference between these two PBRT-v3 materials is that the roughness of the `substrate` material can be specified with two parameters (`uroughness` and `vroughness`) and the roughness parametrization of the `plastic` material is done with only the `roughness` parameter.

Tests have shown that the `plastic` material and the similar `substrate` material of PBRT-v3 can be replicated with a combination of Mitsuba's `roughdiffuse` and `roughconductor` materials and with a combination of LuxRender's `matte` and `metal` materials, where the `material` parameters of Mitsuba and LuxRender must be set to `none` (material = `none` disables the computation of Fresnel reflectances, which results in perfect specular reflection). Only the intensity of the specular color has to be reduced in PBRT-v3. The combination of two materials can be accomplished by using the `mixture` materials. Fig. 4.9 shows the result with adjusted PBRT-v3 specular color intensities. The difference between the three systems is near zero. The specular component of the plastic material of PBRT-v3 was set to [r=0.4, g=0.4, b=0.4] and the specular component of Mitsuba was set to [r=1.0, g=1.0, b=1.0]. In LuxRender, the specular component of

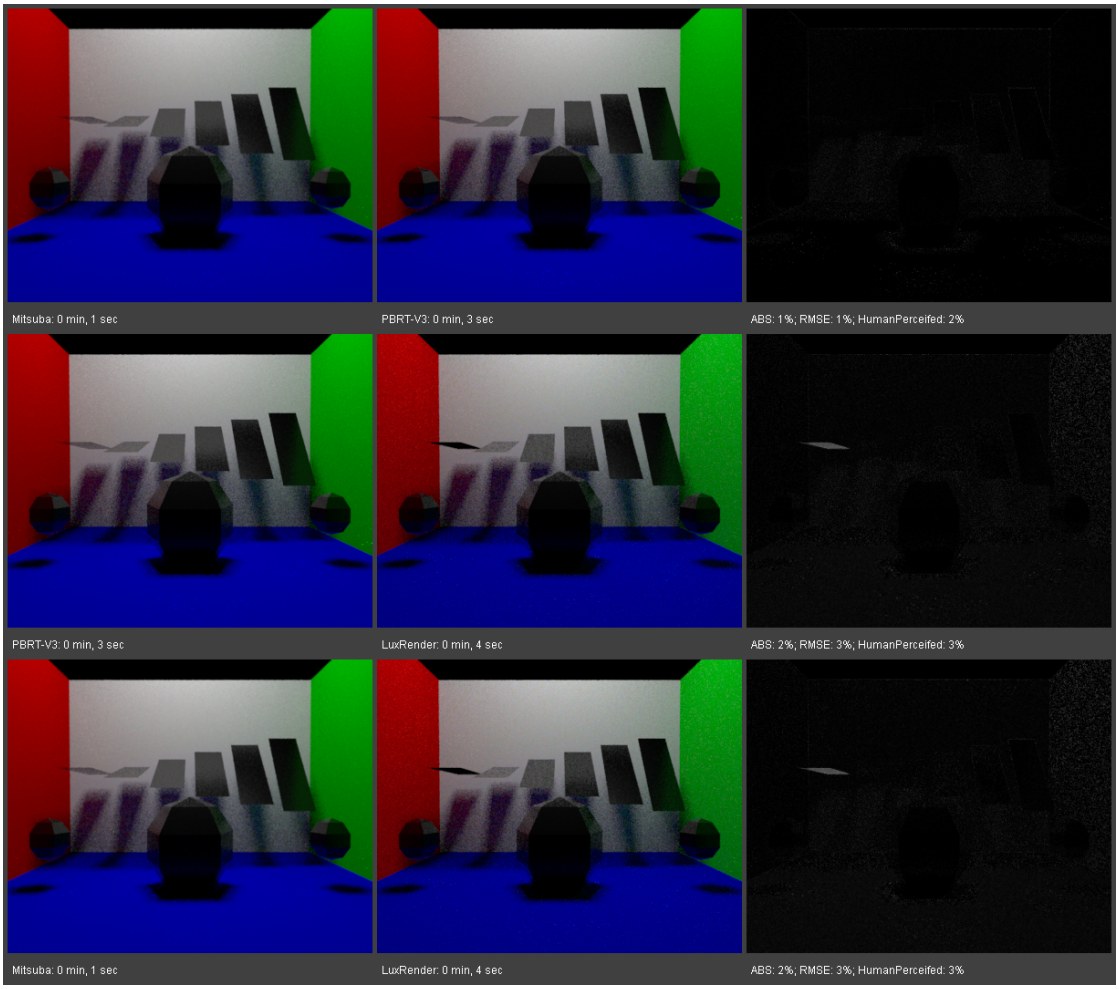

Figure 4.8: Plastic material comparison of Mitsuba (`roughplastic`), PBRT-v3 (`kdsubsurface`), and LuxRender (`glossy`)

the `metal` material can not be changed. So, in general, there are two different ways to describe platic materials that look similar across the different rendering systems.

4.4 Emitters

In this section, the emitters of the rendering systems are tested and compared to each other. The light sources that have similar effects on all three rendering systems are the point lights, area lights, directional lights, and spotlights. For the following light sources, no similarities could be found—Mitsuba: `collimated`, `sky`, `sun`, `sunsky`, `envmap`, `constant`; PBRT-v3: `infinite`, `projection`, `goniometric`; Luxrender: `sky`, `sun`, `infinite`, `projection`, `goniometric`. Accordingly, they will be disregarded in this section.

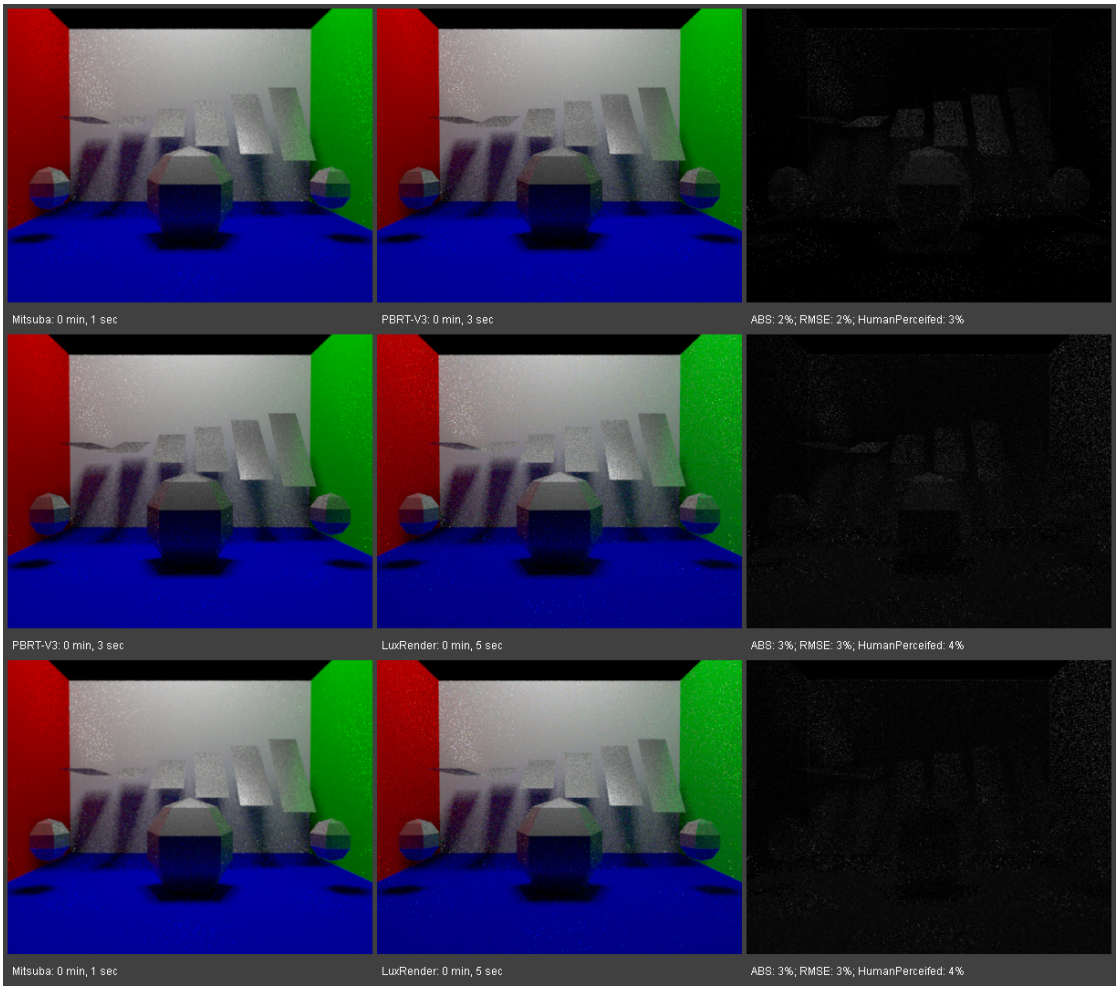

Figure 4.9: Another plastic material comparison: Mitsuba (`roughdiffuse` + `roughconductor`, material set to `none`), PBRT-v3 (`plastic`), and LuxRender (`matte` + `metal`, material set to `none`)

Mitsuba	PBRT-v3	LuxRender
`toWorld`		
`position`	`point`	`from`
`intensity`	`spectrum`	`I`

Table 4.6: Corresponding parameters of the point light sources in Mitsuba, PBRT-v3, and LuxRender

4.4.1 Point Light

Point light sources are used to light up specific areas in the scenes. The point light source has a fixed position and the light is emitted in all directions with the same probability. Point lights do not have a volume or area. The point light is just a point in the scene. The shadows from point lights are hard.

Table 4.6 shows the different parameters for the different rendering systems. In Mitsuba, there are two parameters for specifying the location of the light. In LuxRender, there are additionally some parameters for specifying the intensity of the light sources, i.e., the parameters `efficacy` and `gain`. The `gain` parameter can be used to multiply all three color channels with a factor. For instance, the intensity of (r=1, g=5, b=2) in Mitsuba and PBRT-v3 corresponds to an `intensity` of `I` = (r=0.2, g=1.0, b=0.4) and a `gain` factor of 5.0 in LuxRender.

To test the effects of different settings for the point light source, the used test scene contains red, green, yellow and grey planes and three spherical objects with different shades of gray.

Changing the color of the light source has no effect on the difference between Mitsuba and PBRT-v3. Although LuxRender is based on PBRT, there are differences when the scene is rendered with differently colored point lights.

In Fig. 4.10, the differences between PBRT-v3 and LuxRender are apparent. The results show that the color of emitters affect the color of the objects differently. For instance, in the bluish image generated by PBRT-v3, the red, yellow and green bars are completely black, because these three colors are solely based on red and green. However, in the bluish image generated by LuxRender, these bars are not completely black. The bars are not as bright as the bars in the image, but a slight tinge of red and green is still visible. The effects in PBRT-v3 and Mitsuba are similar. So, in PBRT-v3 and Mitsuba, materials are capable of completely absorbing particular color components whereas in LuxRender, materials seem to reflect a small amount of light for color components that should be absorbed.

4.4.2 Spotlight

The spotlight can be used to light up the scene from a specific point in a specific direction. The equivalent in the real world would be a flashlight, for instance.

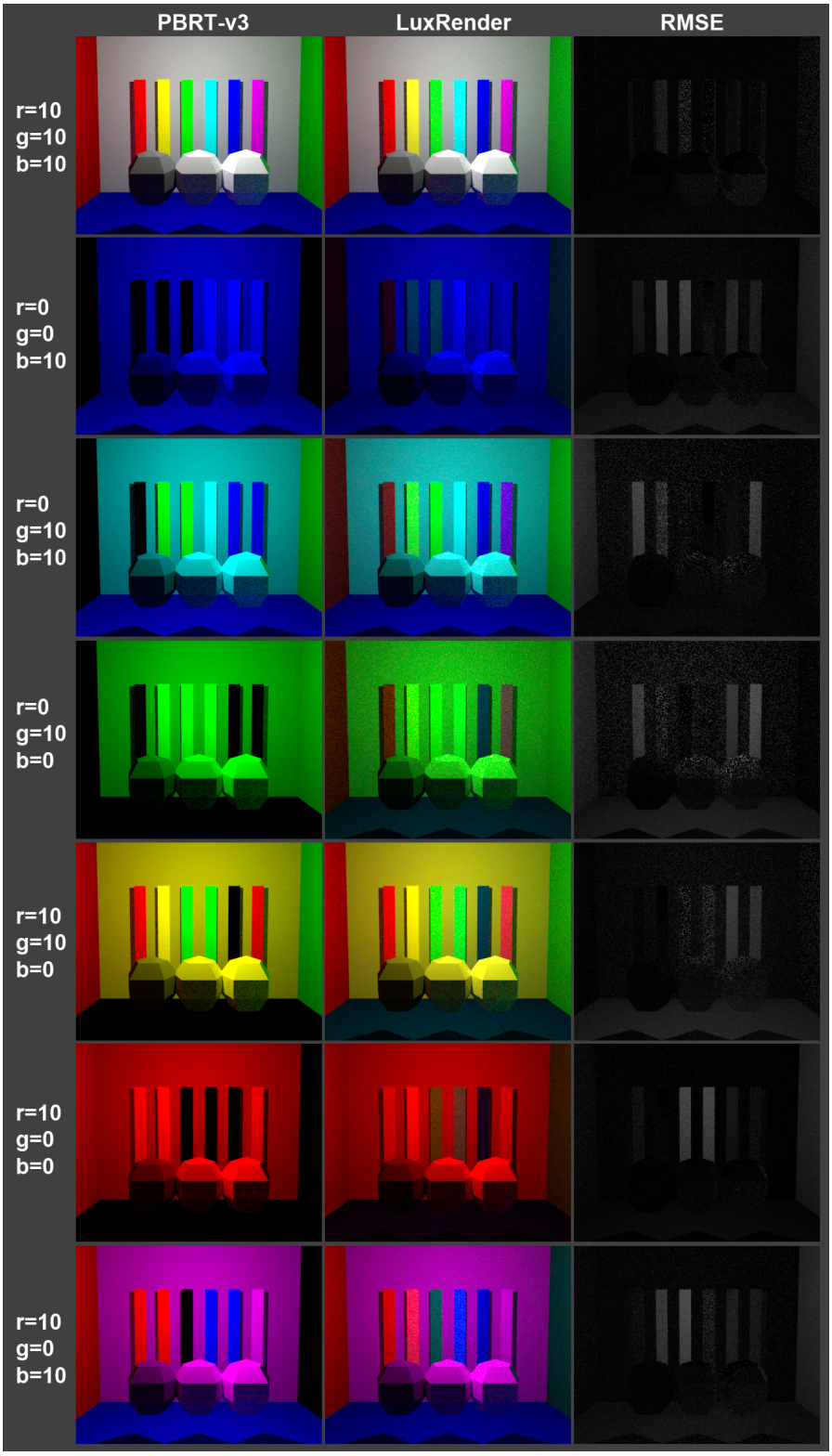

Figure 4.10: Comparison of point light sources for PBRT-v3, LuxRender, and Mitsuba

Mitsuba	PBRT-v3	LuxRender
toWorld	point from/to	point from/to
intensity	I	I
cutoffAngle	coneangle	coneangle
beamWidth	conedeltaangle	conedeltaangle
texture		

Table 4.7: Corresponding parameters of the point light sources in Mitsuba, PBRT-v3, and LuxRender

Table 4.7 lists the parameters of the spotlights in Mitsuba, PBRT-v3, and LuxRender. It can be seen that the spotlight in Mitsuba is the only one for which a texture can be specified. The other parameters have corresponding parameters in other renderers that have similar effects.

PBRT-v3 and LuxRender have similarly named parameters for similar effects. So, only Mitsuba has differently named parameters. The Mitsuba parameter `cutoffAngle` and the corresponding parameter in LuxRender and PBRT-v3 `coneangle` specify the opening angle of the spotlight. For generating the same output image with Mitsuba, PBRT-v3, and LuxRender, the value of these parameters should be the same. The `beamWidth` parameter of Mitsuba must be smaller than the `cutoffAngle`. The angle between `beamWidth` and `cutoffAngle` specifies the area where the intensity decreases from 100 percent to 0 percent. In PBRT-v3 and LuxRender, the parameter `conedeltaangle` specifies the area where the intensity decreases from 100 percent to 0 percent. To provide the most similar effect in PBRT-v3 and LuxRender compared to Mitsuba, the parameter `conedeltaangle` must be the difference of the Mitsuba parameters `cutoffAngle` and `beamWidth`.

Fig 4.11 shows that the gradient from full intensity to no intensity is not the same. The effects can not be 100 percent replicated. So, for generating similar images, the spotlight should be used without the falloff in all three rendering systems.

4.4.3 Area Light

The area light source is a light source that emits the scene from an arbitrarily shaped object. The shadows of the illuminated objects are soft. In reality, all shadows are soft. So, this light source is important to render photorealistic images.

The shape of the object can be arbitrary. In some renderers, there are special shapes that make the calculation faster. For instance in PBRT-v3, the sphere shape can be used to speed up the rendering process. Additionally, the resulting image has less noise with the same amount of samples per pixel. In Fig. 4.12, this effect is apparent.

The area light source needs an associated object. In Mitsuba, the description is done by adding the `emitter` tag inside the definition of a shape. In PBRT-v3 and LuxRender,

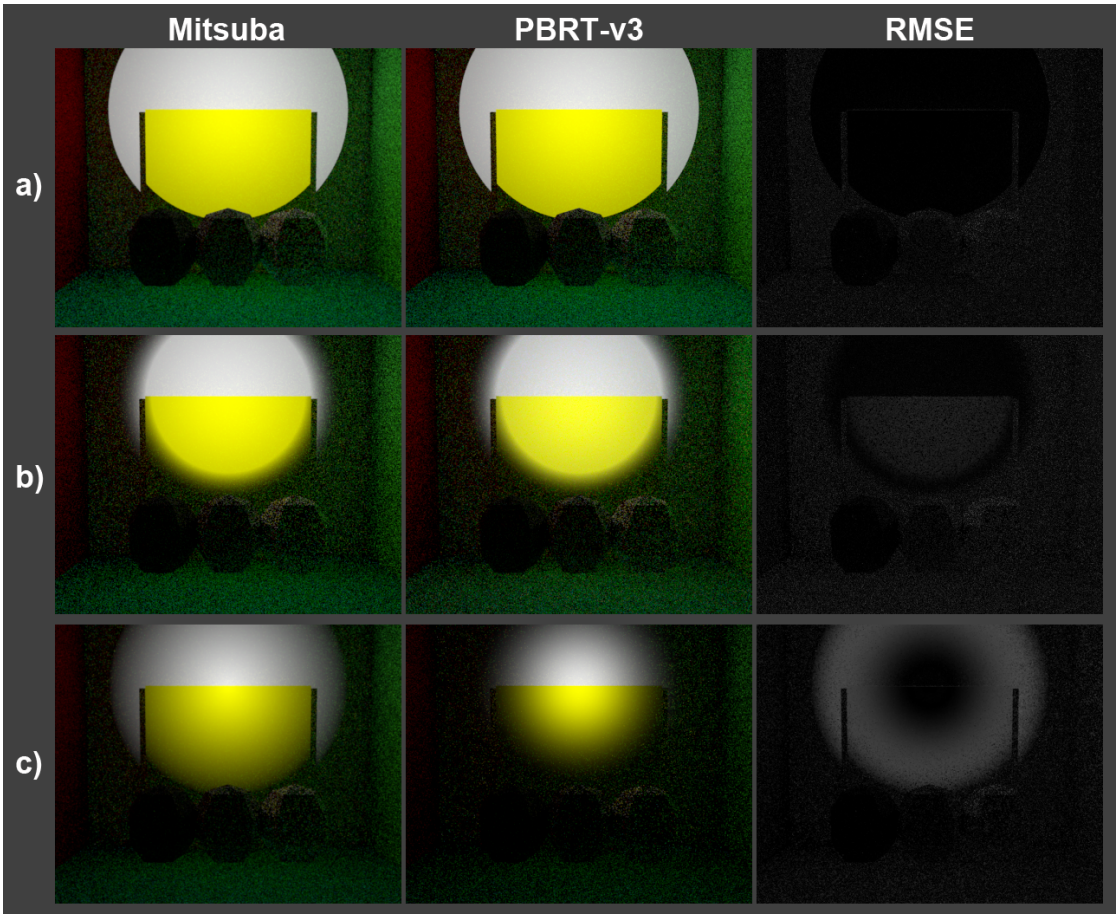

Figure 4.11: Comparison of different spotlights in Mitsuba and PBRT-v3. a) Mitsuba: cutoffAngle=40°, beamWidth=40°; PBRT-v3: coneangle=40°, conedeltaangle=0° b) Mitsuba: cutoffAngle=40°, beamWidth=30°; PBRT-v3: coneangle=40°, conedeltaangle=10° c) Mitsuba: cutoffAngle=40°, beamWidth=0°; PBRT-v3: coneangle=40°, conedeltaangle=40°

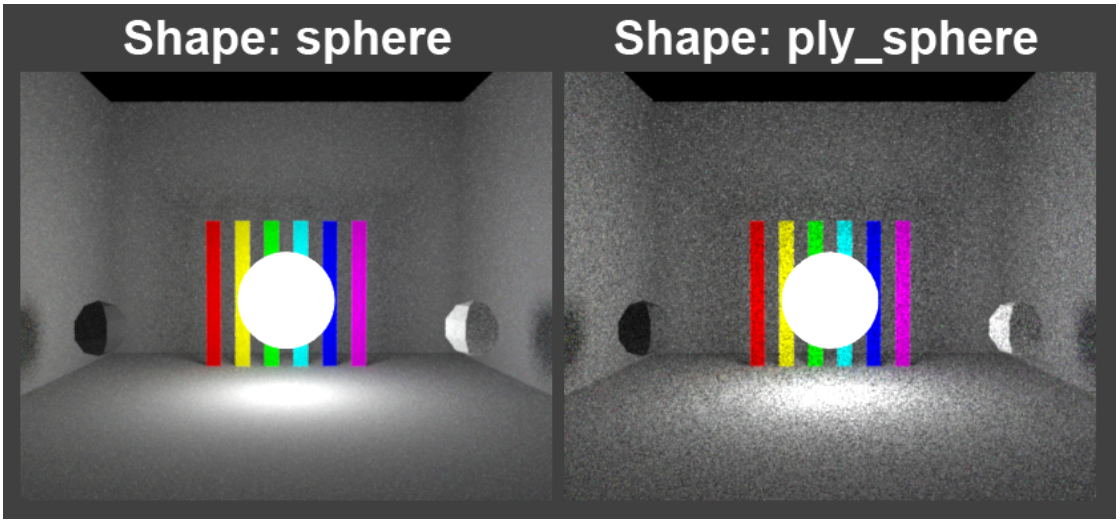

Figure 4.12: Sphere-shaped area light source (PBRT-v3) compared to area light source (PBRT-v3) based on a PLY object which forms a sphere (same SPP)

Mitsuba	**PBRT-v3**	**LuxRender**
`radiance`	`L`	`L`
	`samples`	`nsamples`
	`twosided`	

Table 4.8: Corresponding parameters of the area light sources in Mitsuba, PBRT-v3, and LuxRender

this is done by encapsulating the definition of the `AreaLightSource` and the `Shape` definition between an `AttributeBegin` and an `AttributeEnd` tag. Table 4.8 shows the area light parameters for the different rendering systems.

The only parameter that can be used in all three rendering systems is the specification of the light intensity. In Mitsuba, the parameter is named `radiance` and in PBRT-v3 and LuxRender, the parameter is named `L`. This parameter can be used for specifying the color and power of the light source. The same values of the radiance parameter in Mitsuba and PBRT-v3 nearly result in the same output. In LuxRender, the power of the light source can be specified separately with the `gain` parameter. This parameter is not listed in Table 4.8 because it is not a parameter that is specific to the area light. In LuxRender, the `gain` parameter can be used for all types of light sources. This parameter is necessary because the intensities of the Mitsuba and PBRT-v3 area light sources are different compared to LuxRender. In Mitsuba and PBRT-v3, the intensity of the light source increases if the size of the area light increases. In LuxRender, this effect can be compensated by using the aforementioned `gain` parameter. Tests showed that the `gain` parameter for an area of 1 x 1 should be set to about 0.00175.

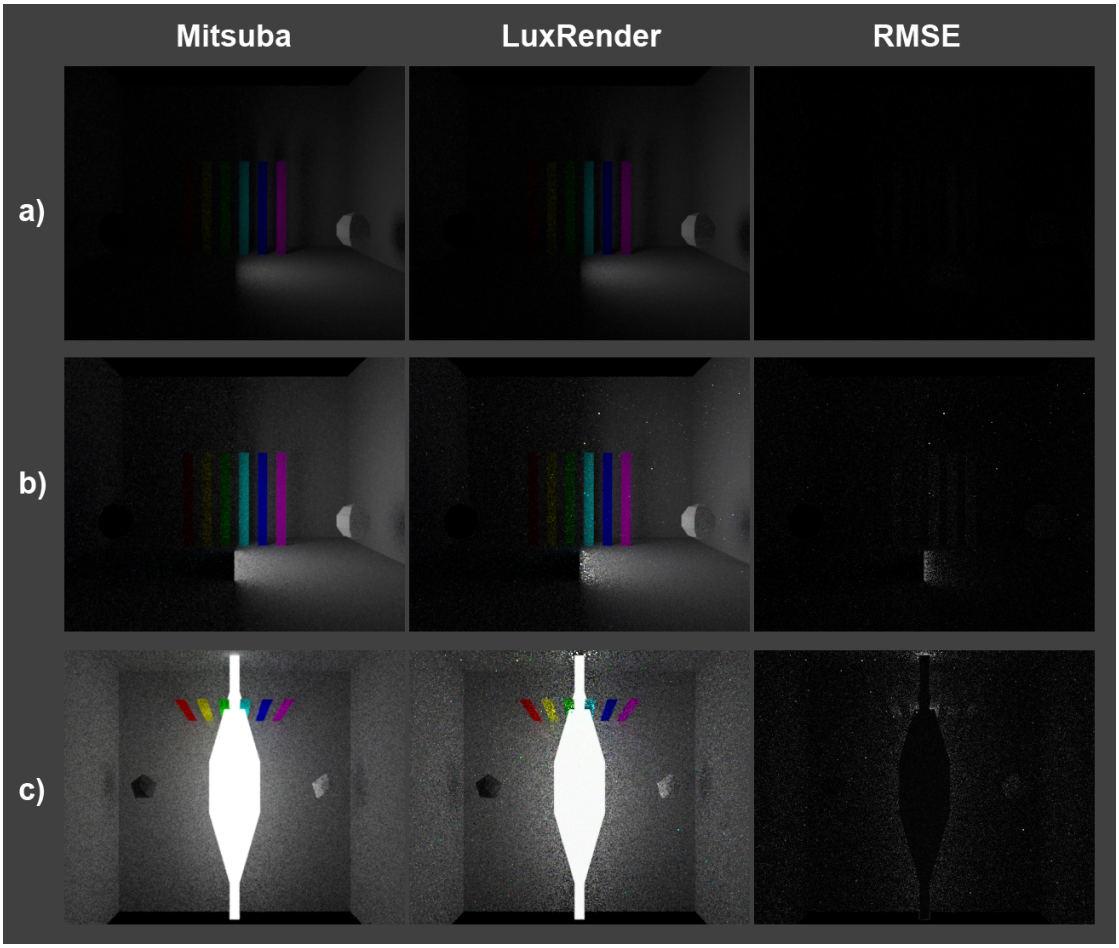

Figure 4.13: Area light source comparison of Mitsuba and LuxRender

Fig. 4.13 shows the comparison of the area light sources in Mitsuba and LuxRender. The images in row a) were rendered with a rectangular area light source with the size of 1×1. The gain parameter in LuxRender was set to 0.00175. The difference in the resulting images is nearly zero. The images in row b) were rendered with a 2×2 rectangular area light source. The gain parameter was set to 0.007. The image generated by LuxRender suffers from more noise, but the intensities are the same. The images in row c) show that the object can be arbitrary. The gain factor was set to $0.00175 \times area(longobject) = 0.00175 \times 51.246 = 0.08968$. The intensities of the resulting images are nearly the same.

So, for making the area light source interoperable between the three rendering systems, only a correction factor in LuxRender has to be set.

Mitsuba	PBRT-v3	LuxRender
`toWorld`	`point from/to`	`point from/to`
`direction`		
`irradiance`	`L`	`L`
`samplingWeight`		`importance`

Table 4.9: Corresponding parameters of the directional light sources in Mitsuba, PBRT-v3, and LuxRender

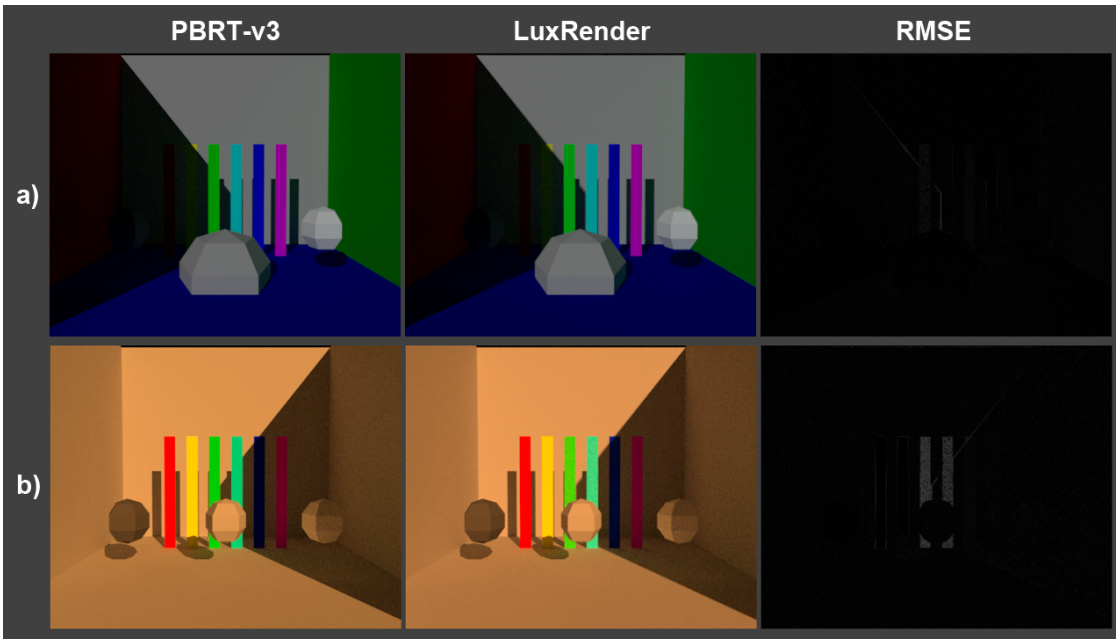

Figure 4.14: Directional light source comparison

4.4.4 Directional Light

The directional light source is used to light the scene with a light source that is located infinitely far away. It can be used to model light sources that are far away, e.g., the sun. The shadows of directional light sources are hard. The rendering systems only offer parameters for specifying the location, direction, and the power of the light source. In Table 4.9, these parameters are summarized.

To produce similar output images, Mitsuba and PBRT-v3 can be parametrized with the same parameters. The result is always the same, except for the different noise levels. In LuxRender, the effects are also similar, but the `L` parameter, which is used to specify the color and power of the light source, has to be adjusted using a constant factor (similarly to area light sources). The adjustment can be done by adding the `gain` parameter with a value of 0.0001.

Fig. 4.14 shows that there is no difference between PBRT-v3 and LuxRender if the gain factor is set to 0.0001. The images in row a) show a directional light with an intensity of (r = 1, g = 1, b = 1) and a direction of the light from right to left. The images in row b) show a directional light with an intensity of (r = 5.0, g = 2.0, b = 0.5) and which points in the opposite direction. The scene of the images in row b) has different, colored walls (all gray). It can be seen that the difference image is almost completely black. Only the colored bars are different. In Section 4.4.1, this effect is explained in more detail.

4.5 Performance Test

In this section, a comparison of Mitsuba, PBRT-v3 and LuxRender is done by using the materials and lights described in this chapter. The scene shown in Figures 4.15, 4.16, and 4.17 is based on the `Japanese Classroom` scene by NovaZeeke, which can be found on the website of Benedikt Bitterli [Bit]. The scene is edited in such a way that it only uses lights and materials described in this chapter. The scene is lit by a directional light source, a red point light source, area lights at the ceiling and a yellow, sphere-shaped area light in the right part of the scene. Fig. 4.15 shows the results of this scene rendered with a simple path tracer using a random sampler with 10 samples per pixel (SPP). Analogue to that, Fig. 4.16 and Fig. 4.17 show the results of this scene with 100 and 1000 SPP. The resolution of the images is 800×600 pixels. The Mitsuba renderings are significantly less noisy compared to the other rendering systems, especially at 10 and 100 SPP. It seems that the path tracer in Mitsuba is optimized to sample important regions more often, even if the random sampler is used. PBRT-v3 is about 4.3 times slower than Mitsuba and LuxRender is 5.8 times slower.

The test was performed on a Windows PC with an Intel i7 870 CPU with 8GB of RAM. The rendering time ratios between each renderer are nearly the same in Table 4.4, where a simplified version of the scene was used that contains only diffuse and metal materials.

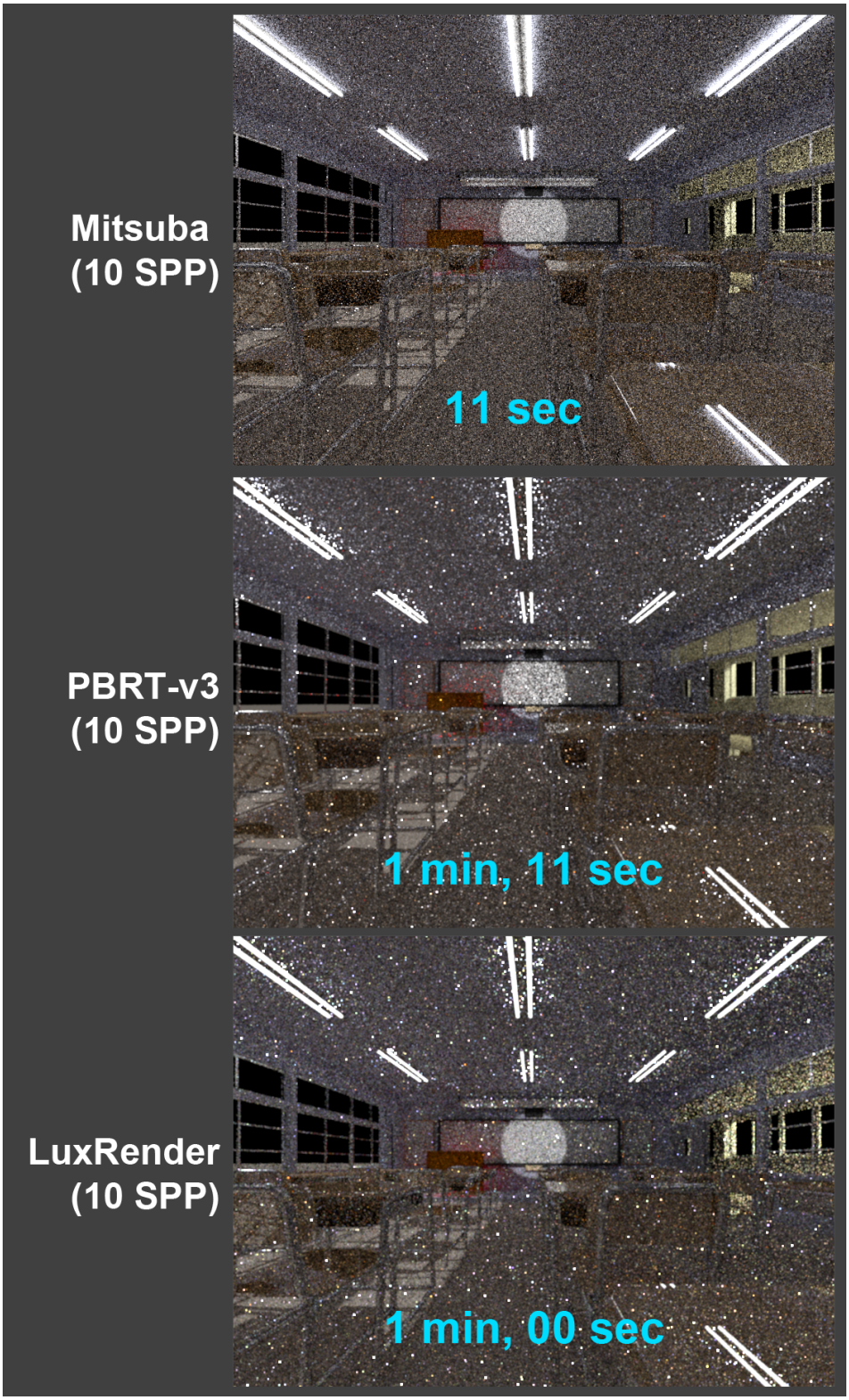

Figure 4.15: Classroom rendered with Mitsuba, PBRT-v3, and LuxRender with 10 samples per pixel

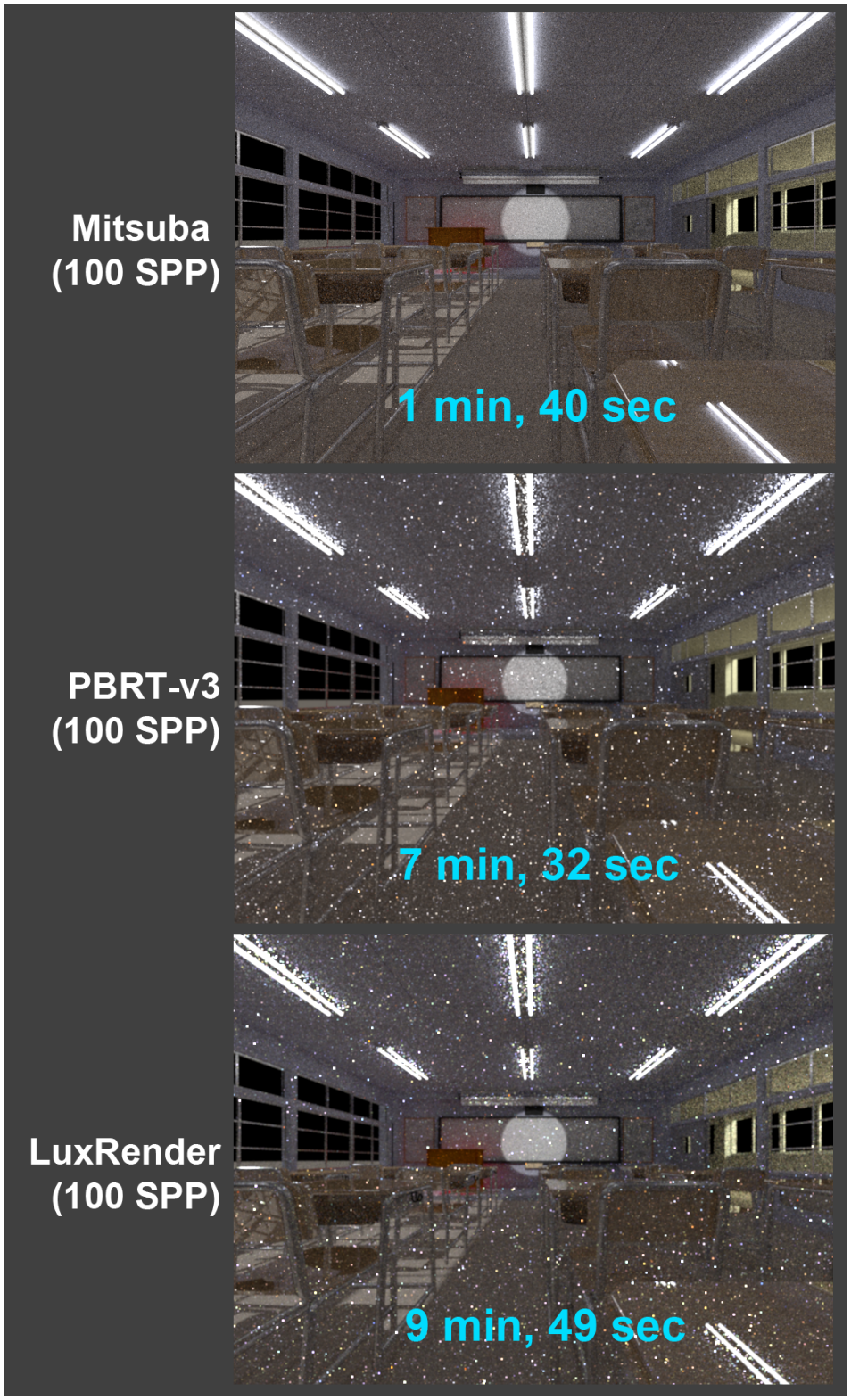

Figure 4.16: Classroom rendered with Mitsuba, PBRT-v3, and LuxRender with 100 samples per pixel

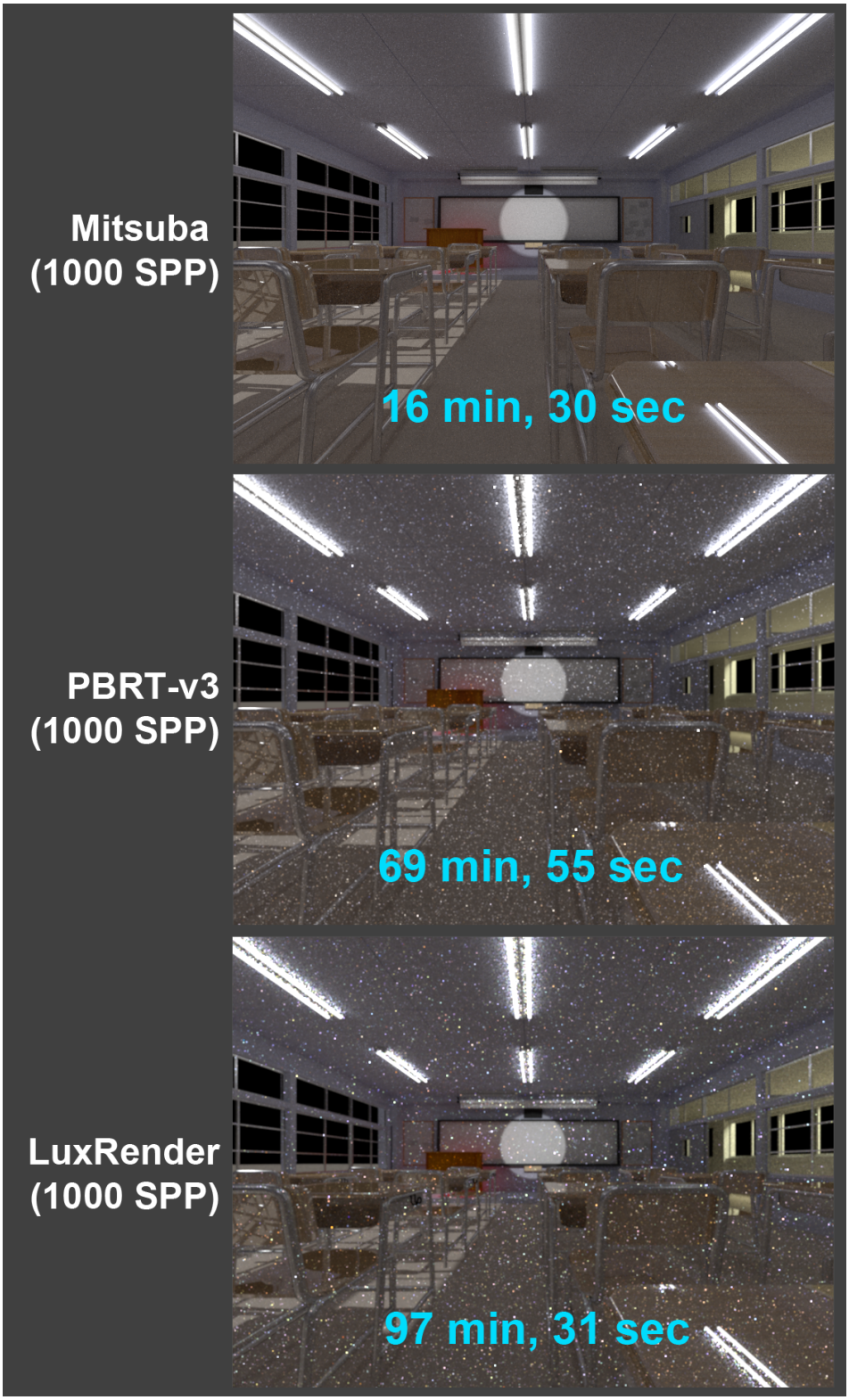

Figure 4.17: Classroom rendered with Mitsuba, PBRT-v3, and LuxRender with 1000 samples per pixel

CHAPTER 5

Conclusion

In this work, a quantitative evaluation was performed to find the most relevant physically based rendering systems in research. With respect to this evaluation, three rendering systems were compared to each other in a more detailed way. The goal was to present general advice for making these systems interoperable.

For finding the most relevant rendering systems, a program was implemented to collect papers which are related to global illumination. The result was that Mitsuba [Jak10] and PBRT [PJH16] are the most used rendering systems for implementing new methods. The third most cited open-source renderer was LuxRender [VRB+08]. This system is based on PBRT and has a user-friendly interface. So, the result of this evaluation was that these three systems are the most relevant ones.

For making Mitsuba, PBRT-v3 (the latest version of PBRT), and LuxRender interoperable, a set of common material models and lights had to be found. Tests showed that the materials diffuse, metal, glass, and plastic can be approximately replicated in all systems and that all three systems have the following light models: point lights, directional lights, area lights, spotlights. The effect of changing the parameters of these materials and lights were tested and differences were presented. Recommendations were given on how the parameters should be used and which usage of parameters can lead to differences. A complete interoperability is not possible, but by using the set of materials and lights presented in this work, the interoperability can be high enough to generate images that look almost the same.

Furthermore, a performance test was performed to gain knowledge about the differences in speed and the amount of noise in the output images. This performance test was performed on a scene that is based on the insights that were gained in the material and light tests. The result was that Mitsuba is the fastest renderer as well as the renderer that produces the least amount of noise. The speed of Mitsuba is much higher than that of PBRT-v3 and LuxRender. The reason for that could be that the system is better

optimized. The question why Mitsuba's images have significantly less noise is not in the scope of this thesis. It is interesting that the results are different although all systems were tested with the same integrator and sampler.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ACG+] Jeremy Allison, Kate Chapman, Mark Galassi, Bradley M. Kuhn, Mike Linksvayer, and Martin Michlmayr. Software freedom conservancy. https://sfconservancy.org/. [Online; accessed 12-August-2017].

[AMD] Inc. Advanced Micro Devices. Radeon prorender. https://www.amd.com/de/technologies/radeon-prorender. [Online; accessed 05-Februar-2018].

[a.s] Render Legion a.s. Corona renderer. https://corona-renderer.com/. [Online; accessed 05-Februar-2018].

[AS00] Michael Ashikhmin and Peter Shirley. An anisotropic Phong BRDF model. *J. Graph. Tools*, 5(2):25–32, February 2000.

[Ass] Eurographics Association. Eurographics - European association for computer graphics. https://diglib.eg.org/handle/10.2312/187. [Online; accessed 28-Sept-2017].

[Bit] Benedikt Bitterli. Rendering resources - benedikt bitterli's portfolio: Japanese classroom. https://benedikt-bitterli.me/resources/. [Online; accessed 05-Februar-2018].

[Bit14] Benedikt Bitterli. The Tungsten renderer. https://benedikt-bitterli.me/tungsten.html, 2014. [Online; accessed 20-August-2017].

[BS87] Petr Beckmann and Andre Spizzichino. *The Scattering of Electromagnetic Waves from Rough Surfaces (Artech House Radar Library)*. Artech Print on Demand, 1987.

[BTB17] François Beaune, Eseban Tovagliari, and Luis Barrancos. appleseed - open source production rendering. http://appleseedhq.net/, 2017. [Online; accessed 20-August-2017].

[Cor] Nvidia Corporation. Iray. https://www.nvidia.de/design-visualization/iray/. [Online; accessed 05-Februar-2018].

[Eva] Clark C. Evans. YAML ain't markup language. http://yaml.org/. [Online; accessed 20-August-2017].

[fCM] Association for Computing Machinery. Acm digital library. https://dl.acm.org/. [Online; accessed 25-Sept-2017].

[Gmb] Mental Images GmbH. Mental ray. https://www.nvidia.de/design-visualization/solutions/rendering/product-updates/. [Online; accessed 05-Februar-2018].

[Groa] Chaos Group. V-ray. https://www.chaosgroup.com/. [Online; accessed 05-Februar-2018].

[Grob] The Khronos Group. 3D asset exchange schema. https://www.khronos.org/collada/. [Online; accessed 06-November-2017].

[HJ09] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 141:1–141:8, New York, NY, USA, 2009. ACM.

[Hu17] Yuanming Hu. Taichi - a computer graphics library. http://taichi.graphics/, 2017. [Online; accessed 20-August-2017].

[Jak10] Wenzel Jakob. Mitsuba physically based renderer. https://www.mitsuba-renderer.org, 2010. [Online; accessed 20-June-2017].

[L.] Solid Angle S. L. Arnold. https://www.solidangle.com/arnold/. [Online; accessed 05-Februar-2018].

[ON94] 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*, SIGGRAPH '94, pages 239–246, New York, NY, USA, 1994. ACM.

[Ots15] Hisanari Otsu. Lightmetrica: A modern, research-oriented renderer. https://benedikt-bitterli.me/tungsten.html, 2015. [Online; accessed 20-August-2017].

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

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering From Theory to Implementation*. Morgan Kaufmann, 3 edition, 7 2016.

[Stua] Pixar Animation Studios. Renderman. https://renderman.pixar.com/. [Online; accessed 05-Februar-2018].

[Stub] Walt Disney Animation Studios. Hyperion. https://www.disneyanimation.com/technology/innovations/hyperion. [Online; accessed 05-Februar-2018].

[SWD05] Gaurav Sharma, Wencheng Wu, and Edul N. Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research and Application*, 30(1):21–30, 2005.

[VG97] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[VRB+08] Terrence Vergauwen, Jean-Francois Romang, David Bucciarelli, Jean-Philippe Grimaldi, Ricardo Lipas Augusto, and Asbjorn Heid. Luxrender. http://www.luxrender.net/en_GB/index, 2008. [Online; accessed 10-August-2017].

[WMLT07] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, pages 195–206, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.