UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LUIZA DE AZAMBUJA HAGEMANN

A Converter for Physically-Based Renderer Scenes

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Manuel Menezes de Oliveira Neto

Porto Alegre
June 2018

ABSTRACT

Physically-based rendering systems use proprietary scene description formats. Thus, by selecting a given renderer for the development of a new technique, one is often constrained to test and demonstrate it on the limited set of test scenes available for that particular renderer. This makes it difficult to compare techniques implemented on different rendering systems. We present a system for automatic conversion among scene description formats used by physically-based rendering systems. It enables algorithms implemented on different renderers to be tested on the same scene, providing better means of assessing their strengths and limitations. Our system can be integrated with existing development and benchmarking APIs, lending to full orthogonality among algorithms, rendering systems, and scene files.

Keywords: Physically-based rendering. scene conversion. PBRT. Mitsuba. LuxRender.

P

RESUMO

aceholder.

Palavras-chave: l. physically-based rendering, meta-research, PBRT, mitsuba, luxrender

LIST OF FIGURES

Figure 1.1 An example of a complex scene created by Laubwerk Plants Kits, extracted from [**?**] 9
Figure 1.2 Example of automatic scene conversion obtained with our system. *Coffee Maker* rendered with PBRT v3 (left). Rendering produced by Mitsuba (center) and LuxRender (right), using converted scenes (from PBRT v3) for these rendering systems. 11

Figure 3.1 Visual illustration of the Phong equation, extracted from [Wikipedia 2018] 17
Figure 3.2 Visual illustration of the Ray Tracing algorithm (left) and an example of a scene rendered with the algorithm (right), extracted from [Wikipedia 2018] ... 17

Figure 4.1 Our scene conversion pipeline. An input scene description is imported into a canonical representation, which, in turn, can be exported to a target rendering system format. 20
Figure 4.2 Structure of our canonical scene representation, consisting of rendering options and scene data. The attributes stored for each component are shown on the rectangles on the right. 21

Figure 5.1 *The Wooden Staircase* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system. 26
Figure 5.2 *Teapot* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system. 28
Figure 5.3 *Veach, Bidir Room* (top) and *Cornell Box* (bottom). Input scene descriptions for Mitsuba (a). Renderings produced by PBRT v3 (b) and LuxRender (c), from scene descriptions converted by our system. 29
Figure 5.4 *The Breakfast Room* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system. Mitsuba's sun directive produces a warm-colored lighting. 30
Figure 5.5 *Little Lamp* scene. Input scene description for Mitsuba (a). Renderings produced by PBRT v3 (b) and LuxRender (c), from scene descriptions converted by our system. 30
Figure 5.6 *Veach, MIS*. Renderings by PBRT v3 (a), Mitsuba (b), and LuxRender (c). Converting metal colors to LuxRender is a challenging task, not currently supported by our system. 31

LIST OF ABBREVIATIONS AND ACRONYMS

PBR Physically Based Rendering

MC Monte Carlo

CONTENTS

1 INTRODUCTION....8
1.1 Physically-based Rendering....8
1.2 Rendering a Scene....9
1.3 The Need for Automatic Scene Conversion....10
1.4 Thesis Structure....12
2 RELATED WORK....13
2.1 Similar Conversion Tools....13
2.2 Meta-Research Systems in Computer Graphics....14
3 THEORETICAL FOUNDATIONS....15
3.1 The Rendering Pipeline....15
3.2 Phong Reflection Model....16
3.3 Ray Tracing....17
3.4 The Rendering Equation....18
3.4.1 Monte Carlo Ray Tracing....18
4 AUTOMATIC SCENE CONVERSION....19
4.1 System Architecture....19
4.2 The Import Module....19
4.3 Canonical Scene Representation....20
4.4 The Export Module....23
4.4.1 Matrix Conversion....23
4.4.2 Material Conversion....23
4.4.3 Shape Conversion....24
4.4.4 Global Emitter Conversion....25
5 RESULTS....26
5.1 26
5.2 Discussion....27
5.3 Limitations....29
6 CONCLUSION....32
REFERENCES....33

1 INTRODUCTION

In Computer Graphics, researchers have long pursued the goal of synthesizing images indistinguishable from real photographs. In order to produce physically accurate images, the process of image synthesis - also called *rendering* - has to simulate the interaction of light with the representation of a three-dimensional scene. *Physically-based rendering (PBR)* is a complex process that requires thorough knowledge of optics, material properties, geometry and light propagation.

1.1 Physically-based Rendering

Physically-based rendering is often implemented using Monte Carlo (MC) Ray Tracing [Lafortune 1996], which uses MC integration [**?**] to estimate the environment illumination function. This is performed by tracing (or sampling) the path of several light rays starting from the camera position, simulating the effects obtained from its encounters with virtual objects. While able to produce a high degree of realism, this technique also has a very high computational cost. This method will be further discussed in Chapter 3.

Over the years, PBR became quite popular and was widely incorporated by the entertainment industry. From movies to videogames, from advertisement to interior design, PBR made it possible for artists to bring their vision one step closer to reality.

As PBR popularity grew, several new renderers were developed. Following the creation of *PBRT* and the publishing of the book *"Physically Based Rendering: From Theory to Implementation"* [Pharr e Humphreys 2015], several other research-oriented renderers were created. Among them is *Mitsuba* [Jakob 2014], one of the renderers chosen for this research.

Following the lead of Pixar's *Renderman* [Upstill 1989], many commercial and performance-oriented renderers appeared on the market. Focused on animation techniques and visual effects for movies, these renderers provide well-established, stable rendering techniques. These renderers, such as

LuxRender [Grimaldi e Vergauwen 2008] and *Octane* [**?**], have been extensively used by the animation and gaming industries.

Even with different applications, the vast majority of modern physically-based renderers follows the same general guidelines for defining scene directives and world descriptions. Scene directives establish parameters such as which integration and sampling

techniques the renderer must use, the view matrix and other camera properties. World descriptions describe the objects and materials used to render them. This ensemble of descriptions is called a **scene**.

1.2 Rendering a Scene

3D scenes are usually created by artists using some modeling software (such as Blender [Blender Online Community 2018], 3ds Max [O'Connor 2015] or Maya [Palamar 2015]). But even with an artist's expertise, creating scenes is still a complex process. For instance, scenes created for building overviews and interior design often compile hundreds of 3D models and dozens of customized materials and textures, as one can see in Figure 1.1. Each material and texture has to be carefully defined, taking into account the renderer's limitations and particularities. Most physically-based renderers describe scenes using a similar structure.

Figure 1.1: An example of a complex scene created by Laubwerk Plants Kits, extracted from [**?**]

After a scene is created, all the hard work invested by the artist has been, unfortunately, tailored for a specific render. Should the artist choose to renderer the scene using a different rendering system, the scene file so diligently created would have to be rewritten or heavily modified.

Converting a scene file from one renderer format to another is a hard and time-consuming task. Unfortunately, the various rendering systems available use proprietary

scene description formats. Aside from adapting material and light properties, which can be hard since sometimes renderers do not provide the same features, the 3D object formats supported may not be the same. Manually converting a scene from one format to the other can be extremely time consuming, taking up to several days per scene [Bitterli 2014].

1.3 The Need for Automatic Scene Conversion

Currently, PBR and MC Ray Tracing are the only practical solution for simulating global illumination effects in complex environments. Due to its high computational cost, an image can take a long time to render - sometimes up to a few days, depending on the complexity of the scene or the technique used. This means that it is practically impossible to use these images in real-time applications.

There are some techniques that aim to improve the time spent rendering PBR images, like improved sampling [Heck, Schlömer e Deussen 2013, Pilleboue et al. 2015] and reconstruction strategies [Sen e Darabi 2012, Rousselle, Manzi e Zwicker 2013, Kalantari, Bako e Sen 2015, Bitterli et al. 2016]. These techniques are implemented on top of an existing PBR system as a way of using aspects of the MC Ray Tracing algorithm (such as ray-object intersection calculations) that are orthogonal to the proposed methods.

Since converting scenes between renderers is a very time-consuming task, these new techniques are often constrained to demonstrate their results on a limited set of scenes available for the chosen renderer. This apparently simple limitation has profound implications, as it constrains a direct comparison between MC rendering techniques that have been implemented using different rendering systems.

We present *a system for automatic conversion among scene file formats used by PBR systems*. Our solution intends to expand the repertoire of scenes available for testing, validation, and benchmarking of PBR algorithms. Currently, our system handles conversions among PBRT v3, Mitsuba, and LuxRender, which are three of the most popular physically-based renderers.

Extending it to support additional renderers is straightforward. Our solution (discussed in Section 4) consists of *importing* any source scene description into a canonical representation, which can then be *exported* to other scene formats.

Figure 1.2 illustrates the use of our system to perform automatic conversion of a scene represented in the PBRT v3 format. The images at the center and on the right show, respectively, the renderings produced by Mitsuba and by LuxRender, from con-

Figure 1.2: Example of automatic scene conversion obtained with our system. *Coffee Maker* rendered with PBRT v3 (left). Rendering produced by Mitsuba (center) and LuxRender (right), using converted scenes (from PBRT v3) for these rendering systems.

(a) PBRT v3 (b) Mitsuba (c) LuxRender

verted scenes files. Note the correct representation of the various materials (glass, plastic, and metal).

Our work does not introduce a new physically-based rendering technique per se. Instead, it falls in the area of *meta-research* systems, which are systems designed to facilitate and improve the research process. Meta-research systems are quite common in computer graphics and computer vision [Scharstein, Szeliski e Hirschmüller 2002, Baker et al. 2011, Rhemann et al. 2009, Erofeev et al. 2014], where they have led to significant progress in these fields.

Recently, Santos et al. [Santos, Sen e Oliveira 2018] introduced a framework for developing and benchmarking MC sampling and denoising algorithms. This is achieved by providing an API that decouples the developed techniques from the used rendering system. While it allows a technique to be tested on any rendering system that supports the proposed API, each rendering system is still constrained to a limited set of test scenes. Our system is orthogonal to and complements this API, aspiring to reach full orthogonality among algorithms, rendering systems and scene files.

The **contributions** of our work include:

- A system for automatic conversion among scene file formats used by Monte Carlo physically-based rendering systems (Chapter 4). It enables algorithms implemented on different rendering systems to be tested on similar scene descriptions, giving developers and end user a better assessment of the strengths and limitations of MC rendering techniques;

- The proposal of a mechanism intending to achieving orthogonality among MC rendering algorithms, rendering systems and scene files (Chapter 4). This could be achieved when integrated with the API provided in [Santos, Sen e Oliveira 2018].

1.4 Thesis Structure

This work includes a detailed description of how our solution for converting scenes between renderers was implemented, from system architecture to result images and analysis. Chapter 2 discusses previous scene-conversion efforts, and reviews similar meta-reseach systems introduced to improve research in computer graphics. Following the current state-of-the-art, we delve a little into the history of rendering in Chapter 3, starting with the first algorithms and then moving on to currently used physically-based renderers. Our system is described in Chapter 4, where we explain how our pipeline works and how we converted some renderer-specific directives. Chapter **??** discusses some results obtained with our system, and Chapter **??** presents our conclusions and directions for future exploration.

2 RELATED WORK

In this chapter we discuss relevant, similar systems developed as meta-reseach, such as other attempts to convert scenes and exporting a generic scene format into different renderer-specific outputs. This chapter is meant to contextualize the reader in the state-of-the-art in converting PBR scene files.

2.1 Similar Conversion Tools

With the addition of new physically-based renderers to the market, the number of incompatible file formats began to grow. There were a few initiatives to solve this problem, the greatest one being the proposal of a common file format.

Back in 2004, Arnaud and Barnes created COLLADA [Arnaud e Barnes 2006], a XML schema file format that intended to unify representation of digital assets among various graphics software applications. Ever since it became property of the Khronos Group, several companies included a COLLADA module on their 3D modeling softwares or game engines. However, there were few physically-based renderers that adhered to this file format, one of the few being Mitsuba. That might have happened because COLLADA files only include information about the geometry present in the scene - they don't store any information about other rendering options, such as camera positioning or integration techniques.

To the best of our knowledge, no previous system has performed automatic scene conversion among the major MC rendering systems. Bitterli has converted 32 scenes of various complexities and origins from Tungsten to PBRT v3 and Mitsuba using some scripts [Bitterli 2014]. These scripts, however, are specific for conversions from Tungsten to these two renderers and are not publicly available.

RenderToolbox3 is a MATLAB tool developed for assisting vision research [Heasly et al. 2014]. It imports a scene containing geometric objects described as COLLADA XML files, and allows one to associate to them reflectance measurements from a MATLAB Psycophysics Toolbox [Brainard 1997]. Such reflectance measurements are converted to multispectral reflectance representations compatible to PBRT and Mitsuba. A script then renders the objects with the associated multispectral representations using PBRT or Mitsuba. RenderToolbox3 is a visualization tool for exploring the impact of different reflectance and illuminating properties on human perception.

[Heasly et al. 2014] is a system that imports COLLADA XML files into a MATLAB system and uses different rendering techniques in order images as close to reality as possible. Their system uses integrated PBRT and Mitsuba rendering is capable of storing their data structure into their respective scene file formats. The system does not convert scene files among rendering systems.

2.2 Meta-Research Systems in Computer Graphics

Several systems have been developed to support research in computer graphics. Some well-known examples include Cg [Mark et al. 2003], Brook [Buck et al. 2004], and Halide [Ragan-Kelley et al. 2012].

Cg is a general-purpose programming language designed to support the development of efficient GPU applications, and has stimulated a lot of research efforts in shader-based rendering techniques [Policarpo, Oliveira e Comba 2005, Policarpo e Oliveira 2006, Wyman 2005, Oliveira e Brauwers 2007].

Brook [Buck et al. 2004] is system for general-purpose computation that allows one to exploit the inherent parallelism of modern GPUs without having to deal with GPU architecture details. These kinds of systems were an inspiration that led to the development of CUDA [Nickolls et al. 2008].

Halide [Ragan-Kelley et al. 2012] is a system designed to optimize image-processing applications on multiple hardware platforms by separating the algorithm description from its schedule. The system has been recently extended to support differentiable programming for image processing and deep learning [Li et al. 2018].

All these systems focus on generating efficient code while freeing the user from GPU architectural details. All goal, in turn, is to make high-quality scenes availability independent of one's choice of rendering system.

Santos et al. have recently presented a framework for developing and benchmarking MC sampling and denoising algorithms. They use an API to decouple algorithms from rendering systems, allowing for the same algorithm to be tested on multiple rendering systems. By doing so, they also increase the set of scenes an algorithm can be tested with. However, in order to use a given test scene, the rendering system for which the scene was created would have to be used as well.

3 THEORETICAL FOUNDATIONS

This chapter will introduce important concepts for the reader. We start by introducing the rendering pipeline, then following into the development of more complex algorithms. We do not delve too deep in the implementations of the techniques discussed in this chapter - we provide a general understanding of the process in order to better situate the reader.

3.1 The Rendering Pipeline

In computer graphics, we say that the computer graphics pipeline is a model that describes the steps taken by a graphics system in order to render a 3D scene into a 2D screen (which is most often our computer monitors) [Shreiner e Group 2009]. This process is analogous to the way our eyes process the information around us and form images that are then processed by our brain and engraved into our minds.

Likewise, the computer must interpret some sort of description of an object or scene and pass this information through this pipeline into the final data format understood by our monitors. Most of the pipeline steps are implemented in hardware, allowing optimizations required for real-time rendering.

A graphics pipeline can be divided into five core parts:

1. model and camera transformations
2. lighting or shading
3. projection
4. clipping
5. screen mapping

Each individual part is essential to comprehend the workings of this pipeline. We, however, will be focusing our particular attention into the first two parts - *model and camera transformations* and *lighting*.

In part 1, the *model and camera transformations*, we first process each object composing our scene and apply model transformations in order to position them in what is called the **world space**. The world space is defined by a coordinate system, a reference system to these objects in three-dimensional space.

With our scene now properly set, we then need to know how the camera - our eyes

- is viewing the scene. Where is the camera? How far can the camera see? What is hidden from the camera? We need to represent the scene through the camera's reference system, which we call the **camera space**. Then, we can more easily determine which part of the scene will be within the camera's boundaries and discart the rest in order to improve performance.

In order to go from world space to camera space, one must apply camera transformations. Transforming from one to the other means one must change coordinate systems, which in Linear Algebra can be done by multiplying our models' coordinates (or vertexes) by a specific transformation matrix.

In part 2, we must compute the amount of incident light in our objects, which will determine the color present at a given point. An earlier proposed way to compute lighting in the computer graphics pipeline was the *Phong Reflection Model*. However, a lack of photorealism in the resulting images pushed for the development of more complex models for light interaction in the environment, such as the proposal of the *Rendering Equation* and the creation of global illumination algorithms (*i.e.Ray Tracing* and *MC Ray Tracing*).

3.2 Phong Reflection Model

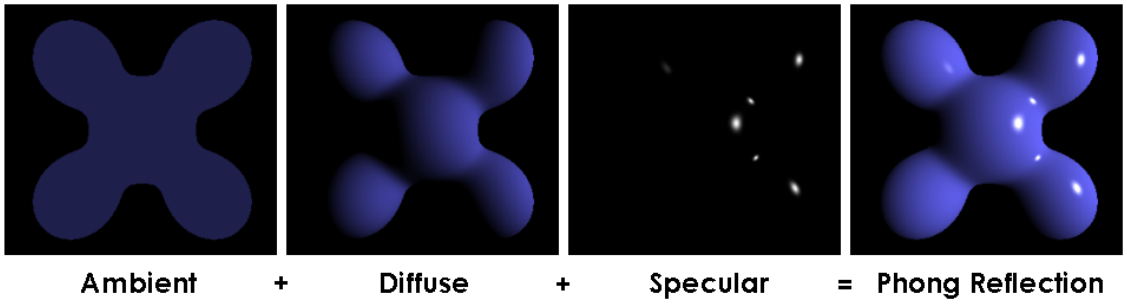
The *Phong Reflection Model* [Phong 1975] was published as an empirical model of local illumination, describing how an object's surface reflects light as a combination of its material's properties - ambient, diffuse and specular reflection. This model can be represented by the following equation:

$$I = k_a I_a + \sum_{m \in lights} (k_d(L \cdot N) + k_s(R \cdot V)^n) I_m \tag{3.1}$$

where k_a, k_d and k_s represent, respectively, the ambient, diffuse and specular properties of the object; L represents the direction of the incident light; N represents the object's normal vector; R represents the direction to which the incident light is reflected; V represents the position from which we are observing the scene; and I represents the incident light's intensity. This visual effects produced by this equation can be observed in Figure 3.1.

Phong's model was widely adopted: its efficiency proved to to be a deciding factor in generating real-time images for 3D scenes. The model, however, was unable to correctly represent global illumination. While the ambient term in the equation is a way of

Figure 3.1: Visual illustration of the Phong equation, extracted from [Wikipedia 2018]



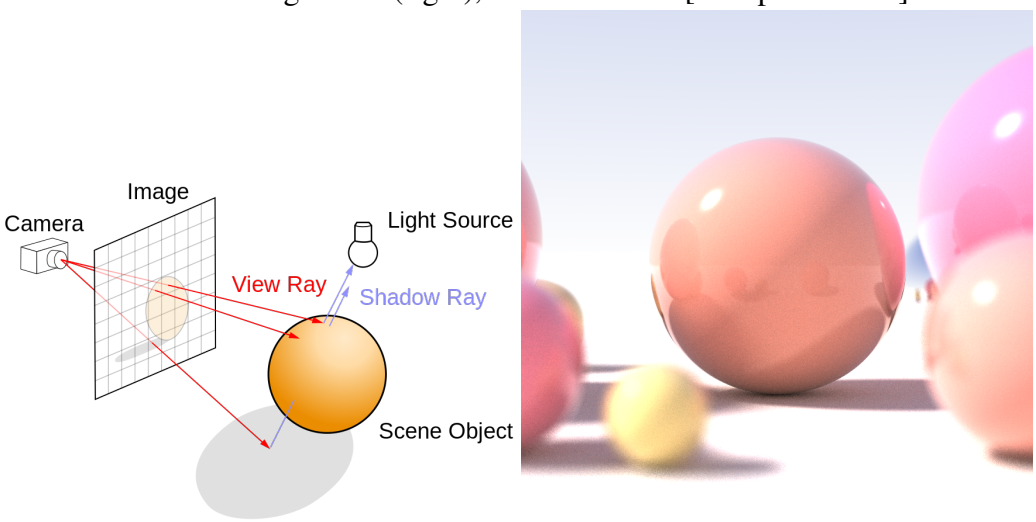
accounting for it, the term alone does not model physically-correct interactions between light and the environment.

Algorithms that compute global illumination have a much higher computational cost, but produce more realistic images. One of the precursors of this type of algorithm was *Ray Tracing*, which we will review in our next section.

3.3 Ray Tracing

The *Ray Tracing* algorithm [Whitted 1980] consists of shooting rays from an imaginary "eye" (in this case, the camera) into the scene through a virtual screen. We trace the path of this ray until we find the closest object to intersect with it. Once the object has been identified, we - like in Phong Reflection Model - estimate the incoming light, examine the properties of the object and combine this information to determine the color in the output image. This process is illustrated in Figure 3.2.

Figure 3.2: Visual illustration of the Ray Tracing algorithm (left) and an example of a scene rendered with the algorithm (right), extracted from [Wikipedia 2018]



The algorithm assumes that a surface may absorb some of the incident light, with results in a loss of intensity of the reflected light. This surface can also reflect the incident ray in more than one direction. If the object has translucent properties, the incident ray may be refracted, which results in an alteration of direction in the ray's path. These features mean that Ray Tracing, unlike its predecessors, is able to simulate a large selection of optical effects.

It does not, however, come without disadvantages. In order to calculate the amount of light that arrives at an object, the most commonly used method is the Phong Reflection Model. This means that, while the algorithm creates a physically correct interaction of light rays with the environment, the final image is usually not photorealistic.

Photorealism can only be achieved when an algorithm approximates the *Rendering Equation* - which describes every physical effect of light flow. This method will be furtherly discussed in our next section.

3.4 The Rendering Equation

WORK IN PROGRESS

3.4.1 Monte Carlo Ray Tracing

4 AUTOMATIC SCENE CONVERSION

This chapter introduces our system and the scene conversion pipeline. We detail the scene import process, the intermediate state of the data and how we convert this information into the desired output renderer.

4.1 System Architecture

Our system is a Proof of Concept (PoC) that consists of two main components: an *import module* that reads an arbitrary scene file format and generates an equivalent description in a *canonical* scene representation; and an *export module* that takes our canonical representation and exports it to a target rendering system file format. The complete process is illustrated in Figure 4.1.

Currently, our system supports *PBRT v3* [Pharr, Jakob e Humphreys 2016], *Mitsuba* [Jakob 2014], and *LuxRender* [Grimaldi e Vergauwen 2008], as these are three of the most popular rendering systems. This architecture, however, is quite flexible. Supporting additional rendering systems only requires specializing the import and export methods to handle the new formats. Next, we describe the main components of our system.

4.2 The Import Module

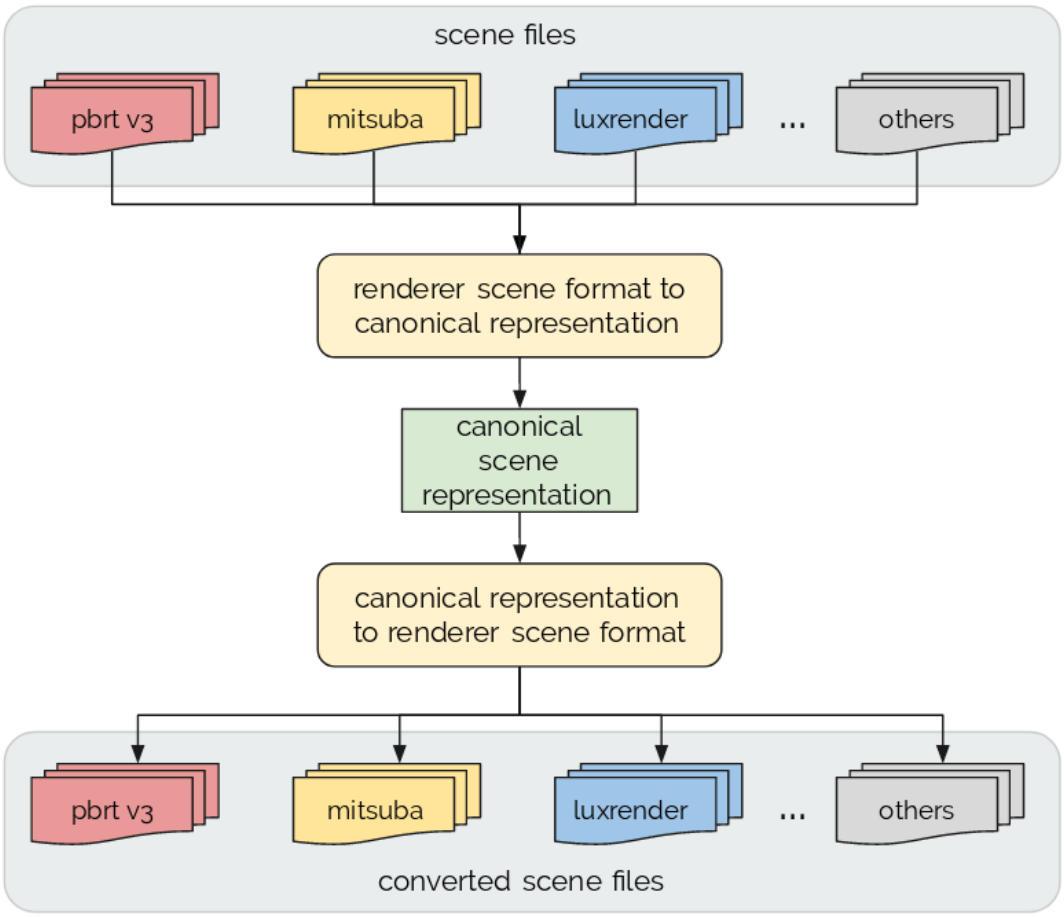
Most physically-based renderers subdivide the scene description in two main sections: *scene-wide rendering options* and *world block*. The former defines the rendering settings, while the latter describes the scene geometry and materials.

The import module parses the input scene files and translates each directive into a canonical representation. Since rendering systems use proprietary file format, both the import and export modules have to be specialized for each renderer.

PBRT and LuxRender scene descriptions consist of structured text statements. We generated parsers for these systems using PLY [Beazley 2001], a Python implementation of Lex and Yacc.

Mitsuba, meanwhile, is a heavily optimized, plugin-oriented renderer. Its file format is, essentialy, an XML description of which plugins should be instantiated with the specified parameters. Since there are several XML-parsing libraries for Python, we chose

Figure 4.1: Our scene conversion pipeline. An input scene description is imported into a canonical representation, which, in turn, can be exported to a target rendering system format.



to use ElementTree [The ElementTree XML API], a Python XML parsing tool.

4.3 Canonical Scene Representation

While most renderers have a similar structure, they differ in a few supported features and in the parameters used to configure the rendering process. Thus, we need a canonical representation that covers the features supported by all renderers.

COLLADA [Arnaud e Barnes 2006] is an XML schema intended as a representation for exchanging digital content among graphics applications. However, COLLADA files only include information about the scene geometry. No information about other rendering options, such as camera positioning or integration techniques, is available.

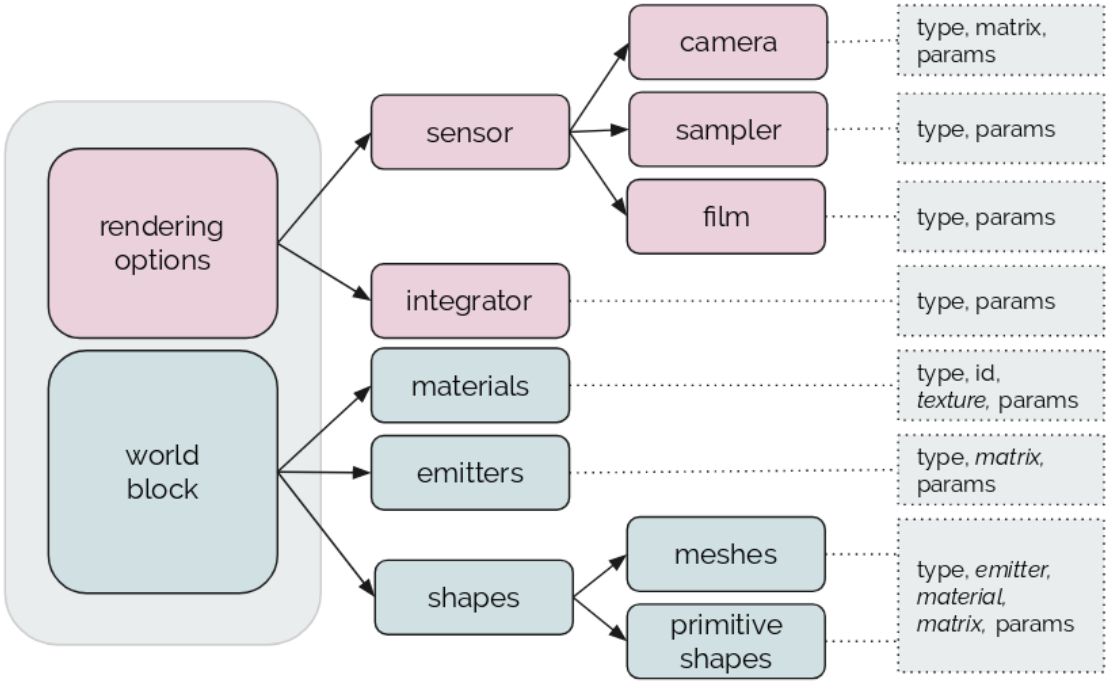
In order to establish a common ground for conversion, we defined a canonical scene representation. It is illustrated in Figure 4.2 and can easily extended to incorporate

any directives not covered in our current implementation.

Our canonical representation mirrors the general structure of scene files and divides the scene data into scene-wide *rendering options* and *world block*.

This is illustrated in Figure 4.2, where the attributes stored for each scene component are shown on the rectangles on the right.

Figure 4.2: Structure of our canonical scene representation, consisting of rendering options and scene data. The attributes stored for each component are shown on the rectangles on the right.



The *Rendering Options* specify the integration and sampling techniques used for rendering, as well as camera and film properties. These include, for instance, camera position, camera matrix, image resolution, field of view, etc. Table 4.1 summarizes all types, parameters, and additional attributes associated with each component of our canonical scene representation.

The *World Block* describes the materials, global emitters, and shapes present in the scene.

A *material* (*e.g.*, glass, plastic, metal, etc.) may have one or more associated textures. *Global emitters* represent all kinds of light sources, except area light sources, which are represented as shapes. These include conventional environment, spot, directional, and point light sources, as well more specific ones such as *sun* and *sky*.

A *shape* can be a polygonal mesh or a geometric primitive such as a rectangle, disk, cube, or sphere, for instance.

Table 4.1: Types, parameters and additional attributes of the components in our canonical scene representation (Figure 4.2).

Component	Type	Parameters	Others
camera	environment, orthographic, perspective, realistic	focal distance, fov, lens aperture, near/far clip, shutter open/close	view matrix
sampler	halton, random, sobol, stratified	samples per pixel, scramble	-
film	hdr, ldr	file extension (png, ...), filter, image height, image width	-
integrator	bidirectional path tracer, direct lighting, metropolis light transport, path tracer, photon mapping	max depth, number of iterations, number of Markov chains, photon count, photon mapping lookup radius, russian roulette depth	-
materials	glass, matte/diffuse, metal, substrate/glossy, translucent, uber	η, id, IOR, k, kd, ks, reflectance, roughness, transmittance, ...	texture (id, type, params)
emitters	directional, distant, environment mapping, sky, spot, sun	filename, from (origin), intensity, radiance, to (direction)	model matrix
shapes	mesh (ply/obj)	filename	model matrix, area emitter, unnamed material
	rectangle, disk, triangle mesh, cube, sphere	center, normals, points, radius, uv mapping, ...	

4.4 The Export Module

The export module is at the core of our system. While the import module deals with a single proprietary scene representation at a time, the export module has to map between materials and scene properties from two proprietary representations.

In this case, there are several delicate cases to consider. Matrix transformations, native shapes, environment mapping coordinates and, mostly, materials are some of the components that vary greatly between renderers. In several situations, there is no direct mapping between them. Still, our system should provide an output representation that, once rendered with the target system, best approximates the results obtained by the source rendering system with the input scene description.

Achieving such results required extensive experimentation with parameters of the various systems. Next, we discuss a few relevant aspects one should consider.

4.4.1 Matrix Conversion

There are several issues to consider when converting matrices between renderers. Do the two renderers use different coordinate systems (either left-handed or right-handed)? Do they represent matrices in the scene file using a direct representation or its inverse-transpose? How is the object-world transformation represented for shapes?

Mitsuba uses a right-hand coordinate system, while PBRT and LuxRender use a left-hand one. This means that, when converting between Mitsuba and the other two, one has to mirror the x-axis of all camera matrix transformations. This is also the case for environment map positioning and object-world transformations. Moreover, Mitsuba's scene files contain a world-to-camera transformation matrix (*i.e.*, view matrix), while PBRT and LuxRender scene files use the view matrix inverse transpose.

4.4.2 Material Conversion

Materials are the most delicate aspect of scene conversion. Materials have spectral and roughness properties that absolutely must be correctly mapped. However, most renderers have very different implementations for common subsurface scattering models (BSDFs), making it hard to predict the mapping between the parameters of two such

implementations.

Mitsuba uses a more physics-oriented approach: a material can be diffuse, conductor, dielectric, plastic, translucent, or a bumpmap. It also has other types of materials, but those are not supported in the current implementation of our system. The material type in Mitsuba changes as the material contains any form of surface roughness, becoming a "rough" version of itself (for instance, a rough metal becomes a roughconductor). PBRT and LuxRender materials have roughness parameters, making it unnecessary to change the material's name.

To represent the material's reflectance, PBRT and Mitsuba use one index of refraction (η) and one absorption coefficient (k) per color channel. LuxRender, however, uses a so-called "*Fresnel texture*", specifying a single value of η and k for all channels. Alternatively, LuxRender allows the specification of a single RGB color value for the material's reflectance. Therefore, correctly converting metal colors between LuxRender and PBRT or Mitsuba is not well defined, and is not supported in the current implementation of our system.

4.4.3 Shape Conversion

Shape directives can be split into two categories: *primitive shapes*, which can be used to specify primitives such as *rectangles, disks, cubes*, and *spheres*; and *3D meshes*, which are stored in external files.

Converting primitive shapes requires more attention than converting external 3D meshes. Mitsuba has directives for rectangle, disk, cube and sphere, while PBRT and LuxRender do not.

Mitsuba's primitives are defined by some parameters (*e.g.*, vertex positions, radius) which can be modified by a transformation (model) matrix. To reproduce these primitives in PBRT and LuxRender, an *internal triangle mesh* must be used. This is done by specifying the position, normal, and texture coordinates for each vertex in the mesh representing a given primitive. One should note that these internal meshes do not use the same representation as the 3D meshes stored in files.

Converting PBRT and LuxRender internal triangle meshes into Mitsuba primitive shapes is a more involving process. Since Mitsuba's primitives have predefined coordinates, converting vertices from PBRT or LuxRender internal meshes into these coordinates requires obtaining the transformation matrix that maps PBRT or LuxRender vertices

to Mitsuba's predefined points. Our system takes care of this automatically.

Converting external 3D meshes is simple, as all rendering systems have directives for this purpose. PBRT, however, does not support Object File Wavefront 3D (.obj) files. In this case, our system issues a warning, making the user aware of the need to convert .obj files off-line.

4.4.4 Global Emitter Conversion

Global emitters can be used to emulate environment lighting, such as the sun, the sky, or an environment map. Converting global emitters can be tricky, mainly because different rendering systems do not implement the same algorithms and directives. For instance, Mitsuba and LuxRender implement *sun* and *sky* directives, while PBRT does not.

A sun directive can be simulated in PBRT using a distant light. A sky directive can be simulated using an environment map of a clear sky. While PBRT and LuxRender access environment maps using spherical coordinates, Mitsuba uses a latitude-longitude format. Thus, a conversion between the two representations is required.

Converting a PBRT distant light into a sun directive for Mitsuba or LuxRender is straightforward. However, converting a PBRT environment map into a sky directive lends to an ambiguous situation, as the converter would require additional information to decide whether the environment map should be treated as a regular environment map, or as a sky directive. Our system solves this ambiguity by asking the user if the environment map should be converted to a sky emitter.

Another sensitive point to consider is converting environment mapping indexation. PBRT and LuxRender access environment maps using spherical coordinates, (θ, ϕ) expressed in cartesian coordinates, while Mitsuba uses a latitude-longitude format. A conversion between the two representations is also required.

5 RESULTS

This chapter will showcase the test results obtained when evaluating our system. We will also discuss the limitations of our system.

5.1

Figure 5.1: *The Wooden Staircase* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system.

(a) LuxRender (b) PBRT v3 (c) Mitsuba

Our system is available on-line [PBR Scene Converter]. We have tested it on a large number of scenes, including the 32 scenes available at Bitterli's rendering resources website [Bitterli 2016]. Here, we include a few examples to illustrate its results on scenes that explore different types of materials, 3D meshes and primitive shapes, image and procedural textures, and various lighting styles. They include most elements typically found in scenes used by physically-based rendering systems. The time required to convert a scene is about 0.5 seconds on a typical PC (Intel i5 3.8 GHz). The scenes were rendered using Mitsuba 0.5.0, PBRT v3, and LuxRender v1.6 on Ubuntu 14.04 LTS. All scenes were rendered using between 5,000 and 8,000 samples per pixel (spp). For any given scene, the same number of samples per pixel was used with all rendering systems.

Figure 1.2 shows a coffee maker containing various materials, including glass, plastic, and metal, as well as textures. The input scene description was provided in the format for PBRT v3, whose rendering is shown on the left. The images at the center and on the right were produced by Mitusuba and LuxRender, respectively, from scene representations automatically converted by our system. Note how the object details have been faithfully preserved in these renderings.

The *Wooden Staircase* scene (Figure 5.1) contains many geometric objects and textures. A LuxRender scene description was provided as input and its rendering is shown in (a). The images shown in (b) and (c) were produced by PBRT v3 and Mitsuba, respectively, from scene representations automatically converted by our system.

The *Teapot* scene (Figure 5.2) contains a shiny object, environment lighting, and a procedural texture. The input scene description was also provided in the LuxRender format. Figures 5.1b and 5.1c show the renderings produced by PBRT v3 and Mitsuba, respectively, from scene descriptions converted by our system.

Figure 5.3 shows two scenes, *Veach Bidir Room* and *Cornell Box*. The first includes caustics, while the second only contains diffuse surfaces. A Mitsuba scene description was provided as input for each of these scenes, whose renderings are shown on the first column of Figure 5.3. Columns (b) and (c) show, respectively, the renderings produced by PBRT v3 and LuxRender using scene descriptions converted by our system.

5.2 Discussion

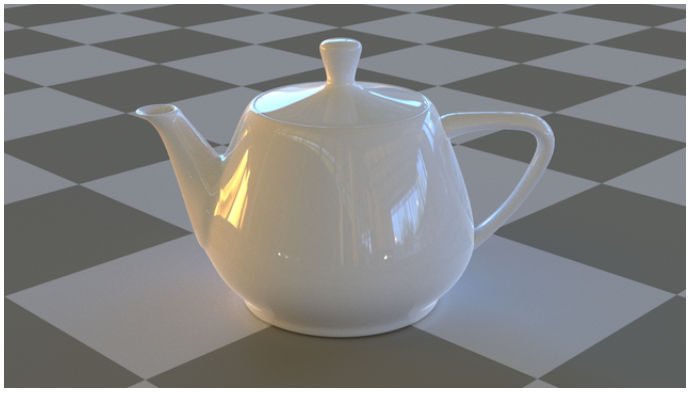
The renderings produced by different rendering systems may exhibit significant differences in color or shading due to features unsupported by some renderers. For instance, consider the use of a light source to emulate the sun. In PBRT and LuxRender, this directive is implemented as a distant white light. Mitsuba, in turn, emulates the sun using a distant environment light implemented according to a technique described in [Preetham, Shirley e Smits 1999], which produces a warm-colored, distant light source. Thus, when the sun directive is used, Mitsuba renderings present a different color compared to the other two. This situation is illustrated in Figure 5.4.

LuxRender does not properly handles a combination of sun directive and local light sources. This is illustrated in Figure 5.4c, where hard shadows have turned soft. The difference in colors are due to the sun directive, as discussed above.

The supplemental materials included with this submission show all the examples

Figure 5.2: *Teapot* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system.

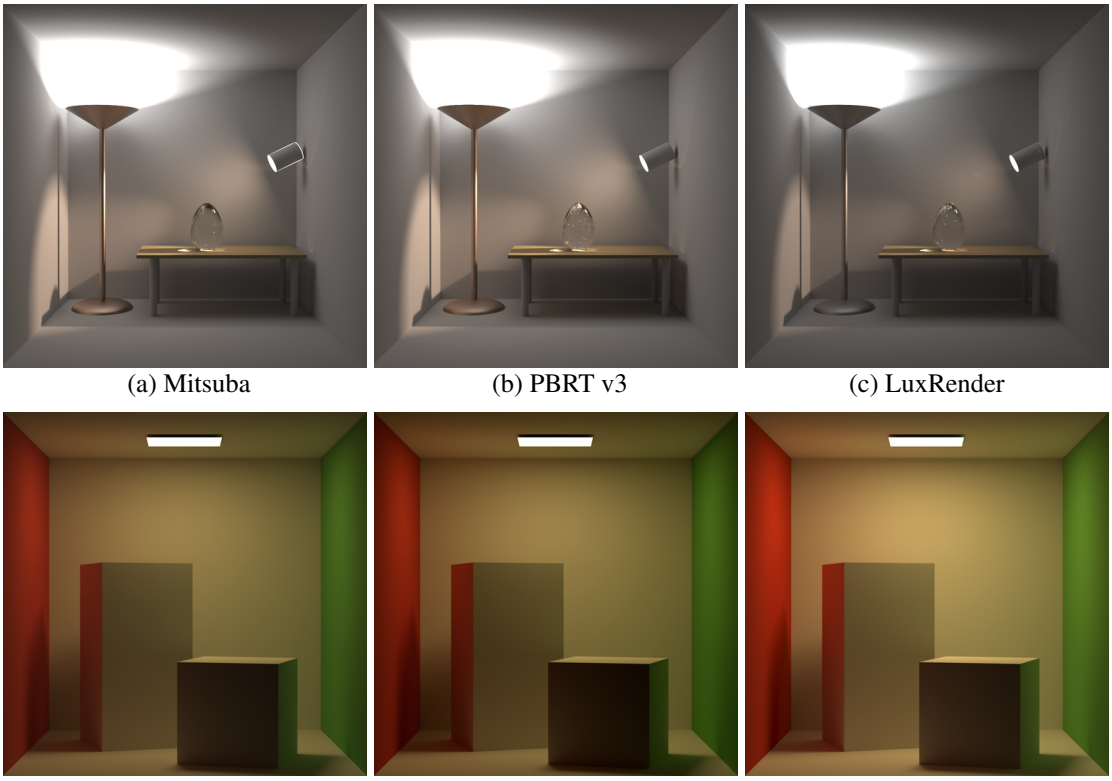
(a) LuxRender



(b) PBRT v3

(c) Mitsuba

Figure 5.3: *Veach, Bidir Room* (top) and *Cornell Box* (bottom). Input scene descriptions for Mitsuba (a). Renderings produced by PBRT v3 (b) and LuxRender (c), from scene descriptions converted by our system.



(a) Mitsuba (b) PBRT v3 (c) LuxRender

presented in the paper plus additional ones. We would like to encourage the reader to explore them, where one can inspect the images at their original resolutions.

5.3 Limitations

Scene-description directives found in one rendering system but without correspondence in the other two renderers are not handled by our system. That is the case, for instance, of Mitsuba-only materials like *phong* and *blendbsdf*.

The current version of our system does not support the conversion of hair or participating media. As discussed in Section 4, LuxRender treats material reflectance differently from PBRT and Mitsuba. Thus, properly converting metal colors to LuxRender is a challenging task, not currently supported by our system. This is illustrated in Figure 5.6, where the rendering of metal obtained from a scene converted to and rendered with LuxRender looks darker.

Figure 5.4: *The Breakfast Room* scene. Input scene description for LuxRender (a). Renderings produced by PBRT v3 (b) and Mitsuba (c), from scene descriptions converted by our system. Mitsuba's sun directive produces a warm-colored lighting.

(a) LuxRender

(b) PBRT v3

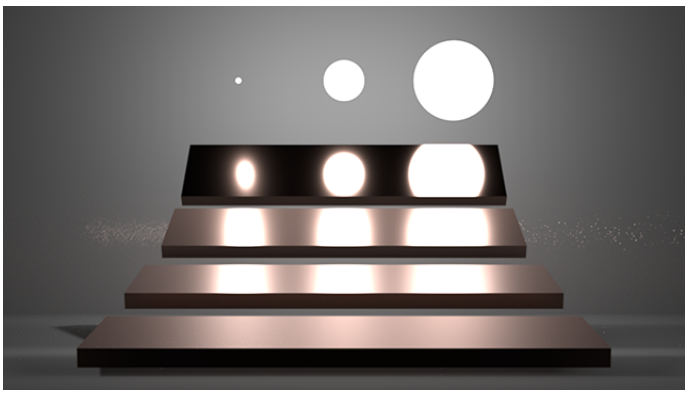
(c) Mitsuba

Figure 5.5: *Little Lamp* scene. Input scene description for Mitsuba (a). Renderings produced by PBRT v3 (b) and LuxRender (c), from scene descriptions converted by our system.

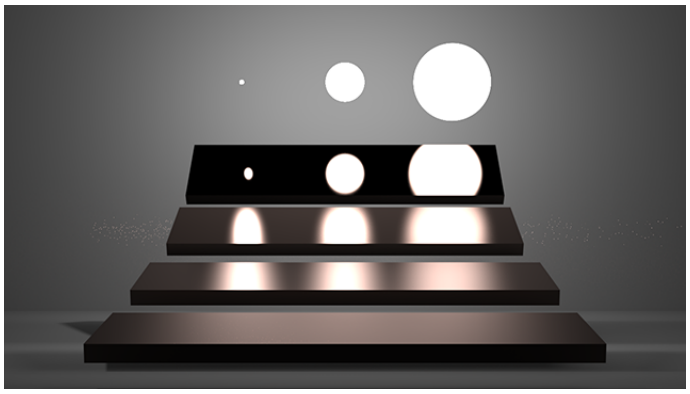
(a) Mitsuba (b) PBRT v3 (c) LuxRender

Figure 5.6: *Veach, MIS*. Renderings by PBRT v3 (a), Mitsuba (b), and LuxRender (c). Converting metal colors to LuxRender is a challenging task, not currently supported by our system.

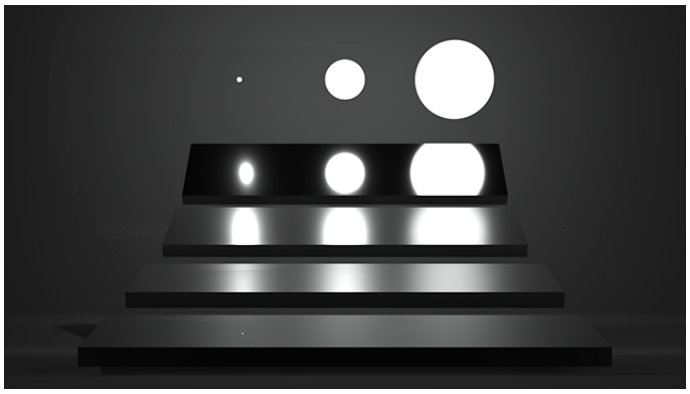
(a) PBRT v3



(b) Mitsuba



(c) LuxRender



6 CONCLUSION

We presented a system for automatic conversion among scene file formats used by Monte Carlo physically-based rendering systems. It enables algorithms implemented using different renderers to be tested on similar scene descriptions, providing better means of assessing the strengths and limitations of MC rendering techniques. Our system can be easily integrated with the API recently introduced by [Santos, Sen e Oliveira 2018], allowing researchers and developers to exploit full orthogonality among MC algorithms, rendering systems, and scene files.

We have demonstrated the effectiveness of our system by converting scene description among three of the most popular MC rendering systems: PBRT v3, Mitsuba, and LuxRender. Providing support to additional renderers only requires specializing the import and export modules described in Section 4 for the given renderers. Our system is freely available and we encourage developers to provide support for other renderers.

In the future, we would like to add support for the conversion of hair and participating media, as well as for other rendering systems. By documenting limitations and incompatibilities found among different renderers, our work might stimulate efforts to reduce these differences.

REFERENCES

ARNAUD, R.; BARNES, M. C. **Collada: Sailing the Gulf of 3D Digital Content Creation**. [S.l.]: AK Peters Ltd, 2006. ISBN 1568812876.

BAKER, S. et al. **Middlebury Flow Accuracy and Interpolation Evaluation**. 2011. Disponível em: <http://vision.middlebury.edu/flow/eval/>.

BEAZLEY, D. **PLY (Python Lex-Yacc)**. 2001. Disponível em: <http://www.dabeaz.com/ply/>.

BITTERLI, B. **The Tungsten Renderer**. 2014. <https://benedikt-bitterli.me/tungsten.html>.

BITTERLI, B. **Rendering resources**. 2016. <https://benedikt-bitterli.me/resources/>.

BITTERLI, B. et al. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. **Computer Graphics Forum**, v. 35, n. 4, p. 107–117, jul 2016. ISSN 01677055.

Blender Online Community. **Blender - a 3D modelling and rendering package**. Blender Institute, Amsterdam, 2018. Disponível em: <http://www.blender.org>.

BRAINARD, D. H. The psychophysics toolbox. **Spatial Vision**, v. 10, p. 433–436, 1997.

BUCK, I. et al. Brook for gpus: Stream computing on graphics hardware. **ACM Trans. Graph.**, v. 23, n. 3, p. 777–786, ago. 2004. ISSN 0730-0301.

EROFEEV, M. et al. **VideoMatting**. 2014. Disponível em: <http://videomatting.com/>.

GRIMALDI, J.-P.; VERGAUWEN, T. **LuxRender v1.6**. 2008. <http://www.luxrender.net/>.

HEASLY, B. S. et al. Rendertoolbox3: Matlab tools that facilitate physically based stimulus rendering for vision research. **Journal of Vision**, v. 14, n. 2, p. 6, 2014.

HECK, D.; SCHLöMER, T.; DEUSSEN, O. Blue noise sampling with controlled aliasing. **ACM Trans. Graph.**, v. 32, n. 3, p. 25:1–25:12, 2013. ISSN 0730-0301.

JAKOB, W. **Mitsuba: Physically Based Renderer**. 2014. <http://www.mitsuba-renderer.org/>.

KALANTARI, N. K.; BAKO, S.; SEN, P. A machine learning approach for filtering Monte Carlo noise. **ACM Trans. Graph.**, v. 34, n. 4, p. 122:1–122:12, jul 2015. ISSN 07300301.

KELLER, A.; HEINRICH, S.; NIEDERREITER, H. **Monte Carlo and Quasi-Monte Carlo Methods 2006**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2007. ISBN 3540744959, 9783540744955.

LAFORTUNE, E. **Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering**. [S.l.], 1996.

LI, T.-M. et al. Differentiable programming for image processing and deep learning in halide. **ACM Transactions on Graphics**, v. 37, n. 4, 2018.

MARK, W. R. et al. Cg: A system for programming graphics hardware in a c-like language. **ACM Trans. Graph.**, v. 22, n. 3, p. 896–907, jul. 2003. ISSN 0730-0301.

NICKOLLS, J. et al. Scalable parallel programming with cuda. In: **ACM SIGGRAPH 2008 Classes**. New York, NY, USA: ACM, 2008. (SIGGRAPH '08), p. 16:1–16:14.

O'CONNOR, R. **Autodesk 3Ds Max 2016 - Modeling and Shading Essentials**. 1st. ed. USA: CreateSpace Independent Publishing Platform, 2015. ISBN 1517185815, 9781517185817.

OLIVEIRA, M. M.; BRAUWERS, M. Real-time refraction through deformable objects. In: **Proc. ACM Symposium on Interactive 3D Graphics and Games**. [S.l.: s.n.], 2007. p. 89–96. ISBN 978-1-59593-628-8.

PALAMAR, T. **Mastering Autodesk Maya 2016: Autodesk Official Press**. 1st. ed. Alameda, CA, USA: SYBEX Inc., 2015. ISBN 1119059828, 9781119059820.

PBR Scene Converter. <https://github.com/lahagemann/pbr_scene_converter>.

PHARR, M.; HUMPHREYS, G. **PBRT, Version 3**. 2015. Disponível em: <https://github.com/mmp/pbrt-v3>.

PHARR, M.; JAKOB, W.; HUMPHREYS, G. **Physically Based Rendering, from Theory to Implementation**. 3. ed. [S.l.]: Morgan Kaufmann, 2016.

PHONG, B. T. Illumination for computer generated pictures. **Commun. ACM**, ACM, New York, NY, USA, v. 18, n. 6, p. 311–317, jun. 1975. ISSN 0001-0782.

PILLEBOUE, A. et al. Variance analysis for monte carlo integration. **ACM Trans. Graph.**, v. 34, n. 4, p. 124:1–124:14, 2015. ISSN 0730-0301.

POLICARPO, F.; OLIVEIRA, M. M. Relief mapping of non-height-field surface details. In: **Proc. ACM Symposium on Interactive 3D Graphics and Games**. [S.l.: s.n.], 2006. p. 55–62. ISBN 1-59593-295-X.

POLICARPO, F.; OLIVEIRA, M. M.; COMBA, J. a. L. D. Real-time relief mapping on arbitrary polygonal surfaces. In: **Proc. the ACM Symposium on Interactive 3D Graphics and Games**. [S.l.: s.n.], 2005. p. 155–162. ISBN 1-59593-013-2.

PREETHAM, A. J.; SHIRLEY, P.; SMITS, B. A practical analytic model for daylight. In: **Proc. SIGGRAPH '99**. [S.l.: s.n.], 1999. p. 91–100. ISBN 0-201-48560-5.

RAGAN-KELLEY, J. et al. Decoupling algorithms from schedules for easy optimization of image processing pipelines. **ACM Trans. Graph.**, ACM, v. 31, n. 4, p. 32:1–32:12, jul. 2012. ISSN 0730-0301.

RHEMANN, C. et al. **Alpha Matting Evaluation Website**. 2009. Disponível em: <http://www.alphamatting.com/eval_25.php>.

ROUSSELLE, F.; MANZI, M.; ZWICKER, M. Robust denoising using feature and color information. **Comput. Graph. Forum**, v. 32, n. 7, p. 121–130, 2013.

SANTOS, J. D. B.; SEN, P.; OLIVEIRA, M. M. A framework for developing and benchmarking sampling and denoising algorithms for monte carlo rendering. **The Visual Computer**, v. 34, p. 765–778, 2018.

SCHARSTEIN, D.; SZELISKI, R.; HIRSCHMüLLER, H. **Middlebury Stereo Vision Page**. 2002. Disponível em: <http://vision.middlebury.edu/stereo/>.

SEN, P.; DARABI, S. On filtering the noise from the random parameters in Monte Carlo rendering. **ACM Trans. Graph.**, v. 31, n. 3, p. 1–15, 2012. ISSN 07300301.

SHREINER, D.; GROUP, T. K. O. A. W. **OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1**. 7th. ed. [S.l.]: Addison-Wesley Professional, 2009. ISBN 0321552628, 9780321552624.

THE ElementTree XML API. <https://docs.python.org/3/library/xml.etree.elementtree.html>.

UPSTILL, S. **RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201508680.

WHITTED, T. An improved illumination model for shaded display. **Commun. ACM**, ACM, New York, NY, USA, v. 23, n. 6, p. 343–349, jun. 1980. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/358876.358882>.

WIKIPEDIA. **Phong reflection model — Wikipedia, The Free Encyclopedia**. 2018. <http://en.wikipedia.org/w/index.php?title=Phong%20reflection%20model&oldid=846717164>. [Online; accessed 27-June-2018].

WIKIPEDIA. **Ray tracing (graphics) — Wikipedia, The Free Encyclopedia**. 2018. <http://en.wikipedia.org/w/index.php?title=Ray%20tracing%20(graphics)&oldid=840257222>. [Online; accessed 27-June-2018].

WYMAN, C. An approximate image-space approach for interactive refraction. **ACM Trans. Graph.**, v. 24, n. 3, p. 1050–1053, 2005. ISSN 0730-0301.