LTBench: An Automatic Benchmark for Physically-Based Rendering

Marco Freire[1,2]

[1] Univ Rennes, F-35000 Rennes, France
[2] ENS Rennes, F-35000 Rennes, France
`marco.freire@ens-rennes.fr`

Internship supervised by Jaroslav Křivánek, Associate Professor (doc. Ing., Ph.D.) of Computer Science; carried out in the Computer Graphics Group at the Faculty of Mathematics and Physics of Charles University in Prague, from May 13, 2019 to August 13, 2019; in collaboration with Vojtěch Tázlar.

Abstract. The Light Transport Benchmark *LTBench* is a unified tool for light transport algorithm testing and comparison. It eases this process by providing specific settings for the most common test cases and minimising user involvement. It can be easily extended and adapted to suit the needs of the different actors in the physically-based rendering community. This document is a specification of LTBench, explaining in detail the principles and design decisions that will drive its implementation.

Keywords: Computer graphics · Rendering · Physically-based · Realistic · Light transport · Benchmark · Testing · Comparison.

Introduction

Rendering is the generation of an image from the description of a scene. The purpose of realistic rendering is to make the rendered image look physically plausible by simulating the interactions of light with the scene as described by light transport theory. This process is usually very computationally expensive.

Academic researchers and industry professionals are integral parts of the realistic rendering community. Researchers try to come up with new methods to generate realistic renders of a scene and optimise existing ones, while industry professionals often work on large-scale projects and need to produce satisfying results within tight deadlines. There is a large array of tools at their disposal such as multiple widely-used renderers, many rendering algorithms with their own strengths and weaknesses, and huge collections of scenes and 3D models. All of these renderers support their own scene file formats which are often incompatible with one another, making working with multiple tools at a time inconvenient.

These actors need to test and compare the different resources at their disposal. For example, researchers may need to compare their new algorithm to a state-of-the-art one, and industrial professionals need to run tests to find the resources better suited to the task at hand. Due to the diversity of these resources and of their corresponding standards, they often end up building ad hoc comparison tools from the ground up, which can be very time-consuming.

Very few resources to help or automate this process exist. An automatic scene converter is presented in [5]. Its enables algorithms implemented in different renderers to be tested on the same scene, even if it is only available in a format compatible with only one of the renderers. This tool can convert between the formats used by the *PBRT* [54], *Mitsuba* [53] and *LuxRender* [49] (now *LuxCoreRender*) renderers. To convert one of these formats into another, it first converts it into a *canonical scene representation* which is then converted to the target format. It provides an *Application Programming Interface* (API) to integrate with existing benchmarking tools.

We present *LTBench*, short for *Light Transport Benchmark*. It is an automatic benchmark for physically-based rendering minimising user involvement. It allows users to test and compare different rendering algorithms across renderers on multiple scenes. LTBench is designed to be automated, making time-consuming comparisons much faster and easier to setup. It is also extensible, so the users can modify it to fit their needs and include new resources into the benchmark. Its main purpose is to become a standard benchmark making the use of ad hoc tools unnecessary. Our approach is similar to [5] but broader in scope. LTBench also provides benchmarking functionalities and allows conversions between more formats, targeting both the academic and industrial rendering communities.

This document acts as a specification of LTBench, going from the most abstract aspects of design, the motivation and the general principles, to the most concrete ones, the actual design decisions that will guide its implementation. LTBench is as of the writing of this document in the software design phase.

The document is structured as follows. First, an introduction to the field of realistic and physically-based rendering is given. Then, the purpose and the design principles of the benchmark are explained in detail. Next, the structure of the benchmark is derived. Finally, the different design decisions that have to be taken to build the benchmark are discussed.

1 Rendering Background

1.1 Rendering Principles

Rendering is the process of generating an image from the model of a scene. It has many applications in the video games and film industries or in domains such as architecture, to visualise buildings before their construction. Depending on the application, different techniques are used. For example, rendering for video games must happen in real-time, so the focus is on efficiency at the cost of realism. In architecture and design visualisation, the focus is on realism at the expense of long rendering times. For the rest of the paper, we will focus on realistic rendering.

To render a realistic representation of a scene, one must understand how light travels through space and interacts with the objects in the scene. Physically-based rendering simulates the flow of light based on its physical properties in order to generate photorealistic images. Light transport theory gives a mathematical description of the simulated light interactions.

The simulation of these interactions is computationally expensive. For this reason, researchers are always trying to find new algorithms to speed up this process. Currently, most photorealistic rendering systems are based on the ray-tracing algorithm. A ray tracer follows the path of different light rays as they travel through a scene and interact with its components. The contributions of these light rays are then gathered to construct an image of the scene.

Most techniques focus on the resolution of the light transport equation, first introduced in [6], shown below in its modern form as taken from [7].

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_{S^2} f(p,\omega_o,\omega_i) L_o(t(p,\omega_i), -\omega_i) |cos\theta_i| d\omega_i \tag{1}$$

Where:

- $L_o(p,\omega)$, $L_e(p,\omega)$ are respectively the exitant radiance and the emitted radiance, at a point p along a direction ω;
- $t(p,\omega)$ is the *ray-casting* function that computes the first surface point intersected by a ray originating at p traveling along ω;
- $f(p,\omega_o,\omega_i)$ is the *Bidirectional Scattering Distribution Function* (BSDF) that computes the probability at p that a ray coming from ω_i exits along ω_o;
- S^2 is the unit sphere centered at p.

Radiance indicates how much power emitted, transmitted or received by a surface will be received by an optical system looking at that surface from a specific direction. This equation is derived from the energy conservation principle and describes the equilibrium distribution of radiance in the scene.

From the exitant radiance, we can compute the color values of the pixels forming the resulting image. Since L_o appears on both sides of the equation, it is usually difficult to solve numerically. For this reason, Monte Carlo integration is used to reliably solve the light transport equation. A detailed description of this approach is given in [8].

1.2 Rendering Actors

Rendering is a field of computer graphics with a lot of different applications. The long-standing research community is active; influential film, video game and graphics processing companies such as Pixar, Sony or Nvidia shape the development of the industry; and many artists use rendering tools to create their works.

Based on its application, we can divide rendering in academic rendering and industrial rendering. Academic rendering focuses on finding new techniques to improve rendering quality and rendering times. New light transport algorithms are created and the existing ones are optimised. Industrial rendering applies these techniques to large-scale applications such as films or architectural visualisation. These renders can be quite expensive to compute. For example, DreamWorks Animation's *How to train your dragon 2* took 90 million core-hours to render and the data necessary to achieve it was stored across 398 terabytes [4].

Both actors use different tools and have different goals. This makes rendering a very complex field, where interactions between these actors are quite common.

1.3 Scene structure

For a renderer to produce an image, it needs to know what to draw and how to draw it. The former is supplied by the *scene description* and the latter by the *renderer settings*. We will refer to these elements together as *scene information*.

The scene description is a model of the scene. It contains a description of the geometry of the objects in the scene, the materials applied to those objects and the lights illuminating the scene. The renderer settings tell the renderer how to draw an image of that model. They contain the light transport algorithm settings, the output settings and post-processing options.

The camera is handled differently depending on the renderers. On the one hand, it is a physical object and a part of the scene description, as it is usually handled in *Digital Content Creation* (DCC) tools such as *Autodesk 3ds Max* [2]. Modifying the camera would then mean modifying the scene itself. On the other hand, it can be seen as a renderer setting dictating how the scene should be seen, as it is done in the PBRT renderer. Changing it would then be similar to changing a parameter of the renderer but not the scene itself.

The scene information is often encoded in multiple files. For example, the *Moana* island scene [35] contains files defining the geometry of every object, textures for every element, animation data and data necessary for the definition of the lights, cameras and objects in the scene. These files use a wide array of different file formats depending on the renderer which makes scene compatibility across renderers difficult.

2 Specification

2.1 General Purpose

The Light Transport Benchmark (LTBench) is an automated benchmark and comparison framework for light transport algorithms. Its purpose is to allow users to compare different light transport algorithms across renderers with minimal user involvement, and automatically generate a comparison and performance report. This tool should be useful to the different actors involved in the rendering community: researchers, industry professionals and artists.

Rendering is a computationally expensive process which can take from a few minutes for simple scenes to days for industrial rendering. Render time depends on the choice of algorithm and renderer, the scene complexity and the target quality. Some algorithms fare better than others in scenes containing certain features. Therefore, it is crucial to identify these features for each algorithm to improve rendering performance. Unfortunately, comparing algorithms and renderers is too labour-intensive to be carried out by hand each time.

Computer graphics professionals often develop their own tools to automate this process. These tools have to be tailored to their target renderers, which are all implemented differently. They use different input formats, feature different light transport algorithms and require different settings. An easily extensible benchmark supporting widely used renderers would provide a good alternative to these ad hoc tools.

Moreover, when developing new algorithms, researchers often use them to render scenes tailored to their strengths. This sometimes leads to unintentionally biased comparisons in research articles. Having a transparent benchmark to test algorithms on a wide array of scenes would prevent this from happening.

2.2 General Principles

The purpose of LTBench dictates the following design principles.

Automation LTBench should be able to render scenes on multiple renderers using different algorithms and custom settings with minimal user involvement.

Accessibility The main objective is to facilitate the comparison process. If the entry level is too high or the learning curve is too steep, users will stick to tools they know how to use.

Extensibility The rendering community is very active and new algorithms, renderers and scenes are created continuously. It should be easy to integrate them in the benchmark.

Transparency The inner workings should be clear to the users if they choose to delve into them. The details of the comparison process have to be available in order for the users to be able to assess what is being compared and ensure that the comparison is fair.

2.3 Requirements

The following requirements are consequences of the general principles.

High-level In order to be accessible to everyone, the benchmark must be *high-level*. Frequent use cases should be readily available to the user, without any adjustment to the inner settings of the benchmark. An exception to this may be the extensibility features such as integrating new renderers to the benchmark.

Modular In order to be extensible, the benchmark must be *modular*. The benchmark must be constructed from independent modules each working on independent tasks. Adding functionality to the benchmark should consist in adding modules to the system.

Out-of-core In order to render industrial-grade scenes, the benchmark must be *out-of-core*. As a reference, the data to render the complete *Moana* island scene weighs around 220 GB. This type of scenes cannot fit in live memory and must be handled differently.

Open-source In order to be transparent and extensible, the benchmark should be released under an open-source license allowing at least private use, distribution and modification of the code. This ensures that users can verify the exact details of the testing and comparison process and adapt it to better suit their needs. It should be noted that some of the scenes included with the benchmark might be protected by different copyright licenses.

DCC Integration In order to be accessible, it should be possible to store scenes in a format compatible with modern digital content creation (DCC) tools, so that users can easily modify scenes to fit their needs. We will focus on making our format compatible with *Autodesk 3ds Max*.

Scene Separation Scene descriptions and renderer settings should be handled separately. As explained previously, those two are very different concepts, so the distinction between them must be clear in the structure of the benchmark.

2.4 Envisioned Workflow

Workflow The benchmark features a fully customisable rendering pipeline and comparison framework. The users can select scenes, renderers and algorithms and compare them in whichever way they want by providing custom settings for the rendering algorithms and the report assembly. The benchmark then outputs an HTML page displaying the output of the renderers, documenting their performance and comparing the selected algorithms and renderers on the different scenes with the provided settings.

Use Cases Usually, the performed tests and comparisons will fall in one of the following use cases:
- A comparison of the output of a single algorithm and renderer on multiple scenes, when supplied with different sets of renderer settings to find the best settings for those scenes;
- A comparison of different algorithms on the same renderer, to evaluate their strengths and weaknesses on scenes showcasing different features;
- Regression testing on a renderer, which is a comparison of two versions of the same renderer to see the changes or errors introduced in the newer one.

The user should also be able to choose any combination of scenes, algorithms and renderers and render them without producing any comparison report.

Benchmark Options If a comparison falls in one of the previous use cases, users should not have to set everything manually. Instead, they can choose the type of comparison they want to run, which will limit the number of settings available to them.

The user can choose from the following options:
- Same algorithm, same renderer (SASR);
- Different algorithms, same renderer (DASR);
- Regression testing (RT);
- Custom render (CR).

SASR If the SASR option is chosen, the user has to specify the renderer and algorithm to be used and the scenes that will be rendered. The user can then supply the list of renderer settings to be tested.

DASR If the DASR option is chosen, the user has to specify the renderer to be used, the algorithms to be compared and the scenes that should be rendered. For each renderer and algorithm pair, the user can supply specific rendering settings.

RT If the RT option is chosen, the user has to specify the renderer versions to be compared.

CR This is the most general use case, where the user has to provide all of the settings. If chosen, the user can choose any subset of renderers and for each renderer, any subset of the supported algorithms.

2.5 Examples

The following examples are intended to represent real use cases of LTBench by illustrating how users would use the different tools provided with the benchmark to solve their problems as efficiently as possible.

First Example

Situation A 3D artist has to render a set of scenes made in *Blender* [1], but the *Cycles* [50] simple path tracer algorithm with the default settings is too slow to meet the deadline. They would like to find settings to render these particular scenes more efficiently without compromising image quality.

Steps:

1. **Download and setup LTBench**
   LTBench is hosted in an online repository which needs to be cloned before use. Cycles must already be installed on the system. The artist then provides the location of the renderer to LTBench. The scene importer converts the set of scenes to intermediate format and makes them visible to the benchmark.
2. **Compare settings**
   The artist creates the set of different renderer settings to be tested on the scenes. Then they select the *Same Algorithm, Same Renderer* (SASR) option, choose Cycles as a renderer, the simple path tracer as the algorithm, and provide the custom settings to LTBench which generates a comparison report.
3. **Choose settings**
   The report compares the output and performance of the renderer with each one of the custom settings supplied, giving the artist a solid basis for choosing the right settings for these scenes.

Second Example

Situation A Ph.D. student proposes a variant of an existing algorithm implemented in the *Mitsuba* [53] renderer. The variant could heavily improve performance while maintaining the same level of quality. They would like to implement it as an extension to Mitsuba, verify that it is working properly, compare it to the original algorithm and potentially present it in an article. The new algorithm will be implemented on top of a copy of the original Mitsuba renderer and takes the same parameters as the original algorithm.

Steps:

1. **Download and setup LTBench:**
   LTBench is hosted in an online repository which needs to be cloned before use. The Mitsuba renderer and the copy should be already installed and setup in the student's computer. The student provides the location of the renderers to LTBench.
2. **Develop new algorithm:**
   The student can start developing the variant. When the successive versions of the algorithm are completed, they run an LTBench regression test on the original and modified versions of Mitsuba. This allows them to prevent unwanted bugs introduced by modifications of the code. This step is repeated until the new algorithm is fully implemented and all bugs have been corrected.
3. **Generate a comparison:**
   Once the implementation is finished and verified, the student wants to write a paper on the advantages of their version compared to the original one. Since the two versions of Mitsuba are perceived as different renderers by LTBench, the student

selects the *Custom Render* (CR) option. Then they select the original algorithm and its variant on the corresponding implementations of Mitsuba, provide the renderer settings for each one of them and choose the scenes that should be rendered. The comparison report is then generated.

3 Structure

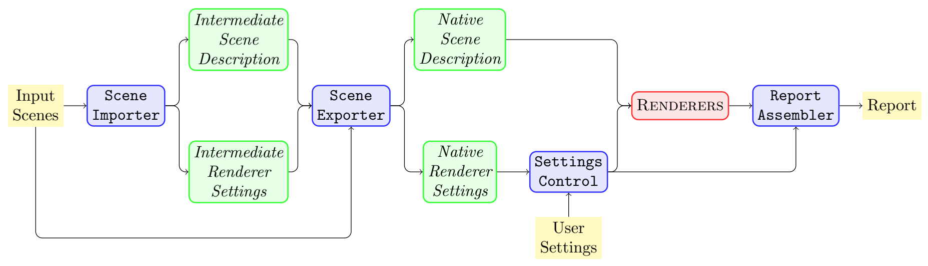


Fig. 1. LTBench general structure

3.1 Component Description

User Interaction

Input Scenes The input scenes can come in the native format of supported renderers or directly in the intermediate format. The benchmark will include popular scenes frequently used in research papers and scenes used in the computer graphics industry. Users will also be able to include their own scenes in the benchmark. The choice of scenes included in the benchmark is discussed in section 4.1.

User Settings The user settings consist of custom renderer settings for each algorithm and renderer pair, and of report assembly settings affecting the shape and details of the final report. These settings can override the ones provided with the scenes.

Report The report consists of automatically generated HTML files featuring different performance tests and comparisons depending on the parameters provided by the user. Different comparison templates are discussed in section 4.3.

Scene Assembly

Scene Importer The scene importer takes input scenes in their native format, converts them into the intermediate file format and splits them into scene description and renderer settings if there are any.

Scene Exporter The scene exporter takes the scene description and the renderer settings in intermediate format and converts them to the target renderer's format.

Settings Control The settings control merges the exported renderer settings with the settings provided by the user. It overrides the exported settings if they are redefined. It also uses the user-defined comparison settings to configure the report template.

Report Assembly

Report Assembler The report assembler takes the output from the renderers, the applied renderer settings and automatically generates an HTML report comparing the outputs and evaluating rendering performance. It can be customised through user-defined settings.

Scene Information

Native Scene Information The native scene representation is the scene information encoded in formats used by renderers supported by the benchmark. It consists of a scene description and renderer settings.

Intermediate Scene Information The intermediate scene representation is used as a transitional file format in which ideally every feature supported by every renderer in the benchmark can be encoded. This allows the benchmark to take a scene in a specific format and render it in renderers using different file formats. It is also broken into scene description and renderer settings. The intermediate renderer settings consist of the most general rendering settings, while algorithm-specific settings are handled by the *Settings Control* and provided by the user. The need for an intermediate format is discussed further in section 4.2.

3.2 Supported Renderers

We have chosen to support the following renderers in LTBench.

Open-source Renderers

PBRT PBRT [54] is heavily used in computer graphics research and teaching. It is designed to be complete and illustrative. The companion book [7] acts as a documentation of the renderer, as a broad introduction to the field of physically-based rendering and as a compendium of advanced techniques used in rendering.

Mitsuba Mitsuba [53] is also heavily used in computer graphics research. It derives from PBRT with a focus on modularity and optimisation. It places a strong emphasis on experimental rendering. New algorithms are often implemented in Mitsuba by researchers.

Cycles Cycles [50] is a production renderer developed by the Blender Project [1]. It is designed for artistic control and to produce pleasing results out-of-the-box. It is integrated with Blender, an open-source digital content creation tool, but a standalone version is currently in development.

Closed-source Renderers

Corona and V-Ray Corona [51] and V-Ray [52] are high-performance photorealistic renderers heavily used for architectural visualisation. They are available for Autodesk 3ds Max and other DCC tools, and are developed by *Chaos Czech a.s.* and *Chaos Group* respectively. They also come as standalone applications usable through the command line interface.

Arnold Arnold [55] is a long-standing production renderer used in the visual effects and movie industries, developed by *Solid Angle*. It is integrated into Autodesk 3ds Max and also comes as a standalone application.

Summary We have chosen Mitsuba and PBRT because of their predominance in the research community. Cycles is one of the few open-source production renderers, used by the Blender community, therefore is utilised by many artists all over the world. Corona and V-Ray are widely used in architectural visualisation, where difficult light transport situations often arise. Finally, since Arnold is available by default in Autodesk 3ds Max, it seems reasonable to include it in the benchmark.

3.3 Extensibility

One of the benchmark's design principles is *extensibility*. New scenes, new renderers and new algorithms should be easy to add to the benchmark.

New Scenes To add scenes encoded in a supported format to the benchmark, the user needs to run the scene importer on them so that they are converted to the intermediate format. The scenes can then be recognised by the benchmark and selected for testing purposes.

New Algorithms To add a new algorithm to an already supported renderer the user has to modify the settings control to provide the algorithm-specific settings to the renderer. This way, the user can configure the new algorithm by providing these settings.

New Renderers Adding a new renderer to LTBench means telling the benchmark how to convert from that renderer's native format to the intermediate format and vice versa. To convert from intermediate to native, the user must specify through a provided *Application Programming Interface* (API) how to translate the tokens of the intermediate format into native format. To convert from native to intermediate, the user must provide a parser for the native format converting a scene file to a stream of intermediate format tokens.

4 Design Decisions

Now that the motivation, purpose, design principles and structure of the benchmark have been established, the next step is to discuss the specific decisions necessary to its implementation. We must answer the following questions:

– *What scenes should be included with the benchmark?*
– *What intermediate format should be used in the benchmark?*
– *How should materials be represented?*
– *How should the results be displayed?*

4.1 Scenes

Criteria Different algorithms fare better or worse depending on the scene. For this reason, researchers use scenes with various features to compare different rendering algorithms. Industrial scenes are usually very complex and feature highly-detailed geometry, while scenes designed to test algorithms are often simpler, focusing on specific light transport situations. The scenes included in the benchmark should represent both types of scene.

Literature Survey We surveyed light transport papers published in *ACM Transactions on Graphics* and *EuroGraphics Symposium on Rendering* between 2012 and 2018. References to the surveyed articles appear in appendix A. The scenes in Figure 3 are all used in multiple papers of the survey and are available at *B. Bitterli's Rendering Resources* [34], except for *Mirror Balls*.

Other online resources provide scenes in different formats, such as the *McGuire Computer Graphics Archive* [36] or the *PBRT-v3 scenes* [33]. These archives include classic scenes such as *Cornell Box* (Cornell University), *Sponza* (M. Dabrovic) or *San Miguel* (Guillermo M. Leal Llaguno).

Licensing While the benchmark is open-source, each scene comes with its own license, ranging from public domain scenes to scenes unsuitable for commercial purposes. Most of them are redistributable, but the users should be aware of the specific licensing of each scene.

4.2 Intermediate File Format

Criteria LTBench must support different renderers used by the different actors of the rendering community, and users should be able to integrate their own into the benchmark. All of these renderers accept different scene file formats. We want to be able to render a scene encoded in any supported format on any supported renderer, even those incompatible with the original scene file format.

To make this possible, we would need to be able to convert any scene file format into any other. The number of different conversions increases with the square of the number of supported formats. Developing a converter for each pair of formats would be unfeasible. Instead, if a format is at first converted to an intermediate format, which is then later converted to the target format, the number of conversions increases only linearly with the number of supported formats. This is the approach taken in [5].

The intermediate file format acts as the backbone of the conversion pipeline in LTBench. Ideally, we want to be able to convert a scene from any format to another, so that every scene can be rendered on any renderer supported by the benchmark. In practice, every renderer supports a unique set of features. Some renderers will support certain features, but others will not. For this reason, conversions between formats will always be lossy. Because of differing renderer implementations, it is also unreasonable to take a scene, render it with two renderers supporting all of its features, and expect the same output.

LTBench targets both academic and industrial computer graphics communities so it should support features necessary to both. For this reason, the intermediate format should be *expressive*, supporting as many features out-of-the-box as possible. The intermediate format should also be *extensible* so that any missing features can be easily added.

Considered Formats Table 1 lists the file formats we considered for the intermediate file format of the benchmark. We considered formats used for the transfer or exchange of 3D scenes.

Table 1. Considered File Formats

Ref.	Name	Developer
[38]	Mitsuba	W. Jakob
[42]	PBRT	Pharr et al.
[43]	Universal Scene Description (USD)	Pixar
[44]	Alembic	Sony Pictures Imageworks, Industrial Light & Magic
[37]	OpenSceneGraph (OSG)	—
[40]	GL Transmission Format (glTF)	Khronos Group
[39]	Collada	Khronos Group
[41]	Open Game Engine Exchange (OpenGEX)	Eric Lengyel

Mitsuba and PBRT The Mitsuba and PBRT file formats are the native formats of their respective renderers. These are both academic renderers, so the formats support similar features that align with our needs. They lack complex material definitions commonly used in the industry such as shading networks. The Mitsuba format is slightly more expressive and has at least limited extensibility features, whereas the PBRT format has none.

USD USD is a scene interchange format originally developed by Pixar for use in the movie industry. It later became open-source and its scope extended to include other rendering applications. Similarly to the Mitsuba and PBRT formats, its specification covers most necessary scene description features such as geometry, light sources and materials. Materials can be defined in multiple ways, through natively supported shading networks, in the *MaterialX* [45] format or with *Open Shading Language* [48]. USD supports custom schemas, making it possible to add any feature to the format. Some digital content creation tools are starting to support USD. It is also used or supported by other influential companies, such as Dreamworks or Foundry. However, as an industrial format with many features, the framework for its manipulation is correspondingly big and complex.

Alembic Alembic was originally designed as a pure geometry interchange format. For this reason, it has limited support for features essential to scene descriptions such as light sources and materials. It can be extended via schemas, but there are no advantages over USD, which supports more scene-related features and can even use Alembic as its geometry file format.

OSG OSG is a graphics file format focused on OpenGL applications. It contains OpenGL-specific constructs, lacks extensibility and documentation is scarce.

glTF, Collada and OpenGEX glTF is a widely adopted scene description format mainly used in real-time rendering. Collada and OpenGEX are interchange formats used in 3D applications for asset sharing. All of these formats share similar properties: they lack support for some geometric primitives and a majority of light sources, and their shading features are extremely limited. They are also difficult to extend.

Summary Out of the considered file formats, OSG, glTF, Collada and OpenGEX lack support for some geometric primitives, light sources and shading features necessary to physically-based rendering. On top of that, they are not designed to be easily extensible. These formats do not fill the criteria for our intermediate representation.

Alembic does not offer any particular advantage over USD, and it is narrower in scope. This leaves us with the Mitsuba, PBRT and USD file formats. While Mitsuba and PBRT support many of the features needed, they are not intended to be used as interchange formats. As a consequence, they are not designed around extensibility.

We have chosen USD as an intermediate format for all of its natively-supported features and its great extensibility.

4.3 Result Comparison

Literature Survey To generate a report displaying, analysing and comparing the rendered images, we need to know how this is usually done. We used the light transport articles in the previous survey as a reference.

Most of the surveyed papers introduce new light transport algorithms or modify existing ones to improve their performance. Rendered images are usually compared side-by-side, with zoom insets showing important parts of the renders.

Papers compare renders obtained by modifying a single parameter. Most of the time, this parameter is the algorithm used to render the scene. This is showcased as a comparison of the equal-time renders produced by the different algorithms on the same scene. The varying parameter can also be a renderer setting, such as the number of samples per pixel.

Occasionally the output is compared to a reference render obtained after a very long rendering time. Some papers compare the outputs by creating pixel-wise difference images between different outputs.

To demonstrate the performance of an algorithm, papers use convergence plots. These plots graph the error between the output and the reference as a function of time or of a certain renderer setting. Convergence plots use different error metrics in the papers, but the most frequent ones are the mean-square error and the root-mean-square error between the output and the reference renders.

HTML Reports The survey articles often link to an interactive HTML report that allows the user to view the different results showcased in the articles. These HTML reports are often structured in a similar fashion. The following variants are seen multiple times in the papers.

Selective View This view is illustrated in Figure 4a.

This is the simplest of the views. When the user clicks on an example in the lower part, it is enlarged and displayed in the upper part.

Crosshair View This view is illustrated in Figure 4b.

The crosshair in the view is bound to the movement of the cursor. On each of the four quadrants of the crosshair is a different example. The user can drag one of the examples from the lower part of the view to one of the quadrants and drop it, and the example will be displayed there.

Hierarchical View This view is illustrated in Figure 4c.

There are two levels of widgets in the figure. In the first level, each widget represents a specific test or a comparison. Once a comparison is chosen, each widget in the second level corresponds to an output image. In the middle part, the image corresponding to the selected second-level widget is displayed. The user can then move the image and zoom in and out on it. In the lower part are zooms on the cursor for each output image on the current comparison.

This view is implemented in the *Javascript Extended-Range Image* (JERI) [3] framework. This framework can also compute image differences based on multiple error metrics. The user can also use hotkeys to navigate between widgets.

5 Future Work

5.1 Material Representation

Materials characterise how light interacts with an object. Objects can reflect, transmit or scatter light in different ways. For example, reflection can be diffuse, glossy or specular. There are infinitely many variants and combinations of these properties. Figure 2 shows a few examples of materials taken from [47].

For this reason, materials are usually hard to describe. Different renderers use different models to represent them. An academic renderer such as PBRT may use simplified descriptions with predefined types of materials such as metal, plastic or glass. On the other hand, industrial renderers often use shading networks to represent materials. A shading network is a collection of connected nodes that defines how colors, textures and lights contribute to the final appearance of surfaces.

These two material representations are very different and translation from one to another may be problematic and lossy. Nonetheless, we need to pick one specific format that will be used for the materials in the intermediate scene representation.

Currently, we are looking into two of the material transfer formats used in the computer graphics industry: *MaterialX* [45], developed by Lucasfilm and *Material Definition Language* (MDL) [46] developed by Nvidia. We need to look at the features offered by each one of these formats and see which one suits our needs the best.

5.2 Minimal Working Product

The next major step is to implement a minimal working product of the benchmark. It should feature a small selection of scenes, support two renderers and be able to generate a simple report. Starting with the Mitsuba and PBRT renderers seems to be a good first step which avoids some of the technical problems presented in this document, because of their similarity.

Once this version of LTBench is implemented and tested, the rest of the features can be added progressively using the information in this document, thanks to the modular structure and extensibility features of the benchmark.

Conclusion

This document explains the motivation behind the *LTBench* project, the purpose of the benchmark itself and the design decisions that will drive its implementation. It documents the research carried out for the design phase of *LTBench* and records all of the useful information gathered during this phase. It should serve as a reference for the future implementation of the benchmark, which will be the next step of this project.

Acknowledgements

I would like to thank Vojtěch Tázlar for the invaluable feedback he provided during the writing of this report and in general for helping me during my internship, which he will continue in the context of his Master's Thesis. I also would like to thank my supervisor, Jaroslav Křivánek, for the time spent overseeing this project and for sharing his insights on the world of physically-based rendering with me.

References

1. Blender project. `https://www.blender.org/`
2. Autodesk: Autodesk 3ds Max. https://www.autodesk.com/products/3ds-max/overview
3. Disney Research: Javascript Extended-Range Image. `https://jeri.io/`
4. Dreamworks Animation: How to train your dragon 2. `https://graphicacy.com/portfolio-item/dreamworks/`
5. Hagemann, L., Oliveira, M.: Scene conversion for physically-based renderers. pp. 226–233 (10 2018). https://doi.org/10.1109/SIBGRAPI.2018.00036
6. Kajiya, J.T.: The rendering equation. In: Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH '86. ACM Press (1986). https://doi.org/10.1145/15922.15902
7. Pharr, M., Jakob, W., Humphreys, G.: Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edn. (2016), `http://www.pbr-book.org/`
8. Veach, E.: Robust Monte Carlo Methods for Light Transport Simulation. Ph.D. thesis, Stanford, CA, USA (1998), aAI9837162

A Additional References

Survey

9. Belcour, L., Yan, L.Q., Ramamoorthi, R., Nowrouzezahrai, D.: Antialiasing complex global illumination effects in path-space. ACM Trans. Graph. **36**(1) (Jan 2017). https://doi.org/10.1145/2990495, `http://doi.acm.org/10.1145/2990495`
10. Bitterli, B., Jakob, W., Novák, J., Jarosz, W.: Reversible jump metropolis light transport using inverse mappings. ACM Transactions on Graphics (Presented at SIGGRAPH) **37**(1) (jan 2018). https://doi.org/10.1145/3132704
11. Bitterli, B., Jarosz, W.: Beyond points and beams: Higher-dimensional photon samples for volumetric light transport. ACM Trans. Graph. **36**(4), 112:1–112:12 (Jul 2017). https://doi.org/10.1145/3072959.3073698, `http://doi.acm.org/10.1145/3072959.3073698`
12. Chaitanya, C.R.A., Belcour, L., Hachisuka, T., Premoze, S., Pantaleoni, J., Nowrouzezahrai, D.: Matrix bidirectional path tracing. In: Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas & Implementations. pp. 23–32. SR '18, Eurographics Association, Goslar Germany, Germany (2018). https://doi.org/10.2312/sre.20181169, `https://doi.org/10.2312/sre.20181169`
13. Georgiev, I., Křivánek, J., Davidovič, T., Slusallek, P.: Light transport simulation with vertex connection and merging. ACM Trans. Graph. **31**(6), 192:1–192:10 (Nov 2012). https://doi.org/10.1145/2366145.2366211, `http://doi.acm.org/10.1145/2366145.2366211`
14. Gruson, A., Hua, B.S., Vibert, N., Nowrouzezahrai, D., Hachisuka, T.: Gradient-domain volumetric photon density estimation. ACM Trans. Graph. **37**(4), 82:1–82:13 (Jul 2018). https://doi.org/10.1145/3197517.3201363, `http://doi.acm.org/10.1145/3197517.3201363`
15. Gruson, A., Ribardière, M., Šik, M., Vorba, J., Cozot, R., Bouatouch, K., Křivánek, J.: A spatial target function for metropolis photon tracing. ACM Trans. Graph. **36**(4) (Nov 2016). https://doi.org/10.1145/3072959.2963097, `http://doi.acm.org/10.1145/3072959.2963097`
16. Guo, J.J., Bauszat, P., Bikker, J., Eisemann, E.: Primary sample space path guiding. In: Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas & Implementations. pp. 73–82. SR '18, Eurographics Association, Goslar Germany, Germany (2018). https://doi.org/10.2312/sre.20181174, `https://doi.org/10.2312/sre.20181174`
17. Hachisuka, T., Kaplanyan, A.S., Dachsbacher, C.: Multiplexed metropolis light transport. ACM Trans. Graph. **33**(4), 100:1–100:10 (Jul 2014). https://doi.org/10.1145/2601097.2601138, `http://doi.acm.org/10.1145/2601097.2601138`
18. Hachisuka, T., Pantaleoni, J., Jensen, H.W.: A path space extension for robust light transport simulation. ACM Trans. Graph. **31**(6), 191:1–191:10 (Nov 2012). https://doi.org/10.1145/2366145.2366210, `http://doi.acm.org/10.1145/2366145.2366210`
19. Jendersie, J., Grosch, T.: An improved multiple importance sampling heuristic for density estimates in light transport simulations. In: Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas & Implementations. pp. 65–72. SR '18, Eurographics Association, Goslar Germany, Germany (2018). https://doi.org/10.2312/sre.20181173, `https://doi.org/10.2312/sre.20181173`

20. Kaplanyan, A.S., Hanika, J., Dachsbacher, C.: The natural-constraint representation of the path space for efficient light transport simulation. ACM Trans. Graph. **33**(4), 102:1–102:13 (Jul 2014). https://doi.org/10.1145/2601097.2601108, `http://doi.acm.org/10.1145/2601097.2601108`
21. Kettunen, M., Manzi, M., Aittala, M., Lehtinen, J., Durand, F., Zwicker, M.: Gradient-domain path tracing. ACM Trans. Graph. **34**(4), 123:1–123:13 (Jul 2015). https://doi.org/10.1145/2766997, `http://doi.acm.org/10.1145/2766997`
22. Kutz, P., Habel, R., Li, Y.K., Novák, J.: Spectral and decomposition tracking for rendering heterogeneous volumes. ACM Trans. Graph. **36**(4), 111:1–111:16 (Jul 2017). https://doi.org/10.1145/3072959.3073665, `http://doi.acm.org/10.1145/3072959.3073665`
23. Křivánek, J., Georgiev, I., Hachisuka, T., Vévoda, P., Šik, M., Nowrouzezahrai, D., Jarosz, W.: Unifying points, beams, and paths in volumetric light transport simulation. ACM Trans. Graph. **33**(4), 103:1–103:13 (Jul 2014). https://doi.org/10.1145/2601097.2601219, `http://doi.acm.org/10.1145/2601097.2601219`
24. Manzi, M., Kettunen, M., Aittala, M., Lehtinen, J., Durand, F., Zwicker, M.: Gradient-Domain Bidirectional Path Tracing. In: Lehtinen, J., Nowrouzezahrai, D. (eds.) Eurographics Symposium on Rendering - Experimental Ideas & Implementations. The Eurographics Association (2015). https://doi.org/10.2312/sre.20151168
25. Moon, B., Iglesias-Guitian, J.A., Yoon, S.E., Mitchell, K.: Adaptive rendering with linear predictions. ACM Trans. Graph. **34**(4), 121:1–121:11 (Jul 2015). https://doi.org/10.1145/2766992, `http://doi.acm.org/10.1145/2766992`
26. Moon, B., McDonagh, S., Mitchell, K., Gross, M.: Adaptive polynomial rendering. ACM Trans. Graph. **35**(4), 40:1–40:10 (Jul 2016). https://doi.org/10.1145/2897824.2925936, `http://doi.acm.org/10.1145/2897824.2925936`
27. Otsu, H., Kaplanyan, A.S., Hanika, J., Dachsbacher, C., Hachisuka, T.: Fusing state spaces for markov chain monte carlo rendering. ACM Trans. Graph. **36**(4), 74:1–74:10 (Jul 2017). https://doi.org/10.1145/3072959.3073691, `http://doi.acm.org/10.1145/3072959.3073691`
28. Pantaleoni, J.: Charted metropolis light transport. ACM Trans. Graph. **36**(4), 75:1–75:14 (Jul 2017). https://doi.org/10.1145/3072959.3073677, `http://doi.acm.org/10.1145/3072959.3073677`
29. Tessari, L., Hanika, J., Dachsbacher, C.: Local quasi-monte carlo exploration. In: Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas & Implementations. pp. 71–81. EGSR '17, Eurographics Association, Goslar Germany, Germany (2017). https://doi.org/10.2312/sre.20171196, `https://doi.org/10.2312/sre.20171196`
30. Vorba, J., Karlík, O., Šik, M., Ritschel, T., Křivánek, J.: On-line learning of parametric mixture models for light transport simulation. ACM Trans. Graph. **33**(4), 101:1–101:11 (Jul 2014). https://doi.org/10.1145/2601097.2601203, `http://doi.acm.org/10.1145/2601097.2601203`
31. Vorba, J., Křivánek, J.: Adjoint-driven russian roulette and splitting in light transport simulation. ACM Trans. Graph. **35**(4), 42:1–42:11 (Jul 2016). https://doi.org/10.1145/2897824.2925912, `http://doi.acm.org/10.1145/2897824.2925912`
32. Van de Woestijne, J., Frederickx, R., Billen, N., Dutré, P.: Temporal coherence for metropolis light transport. In: Proceedings of the Eurographics

Symposium on Rendering: Experimental Ideas & Implementations. pp. 55–63. EGSR '17, Eurographics Association, Goslar Germany, Germany (2017). https://doi.org/10.2312/sre.20171194, https://doi.org/10.2312/sre.20171194

Scenes

33. Scenes for pbrt-v3. https://casual-effects.com/data/
34. Bitterli, B.: Rendering resources. https://benedikt-bitterli.me/resources/ (2016)
35. Disney Enterprises, Inc.: Moana island scene. https://www.technology.disneyanimation.com/islandscene
36. McGuire, M.: Computer graphics archive. https://casual-effects.com/data/ (July 2017)

File formats

37. Open Scene Graph (OSG). http://www.openscenegraph.org/
38. Jakob, W.: Mitsuba 0.5.0 renderer documentation. http://mitsuba-renderer.org/docs.html
39. Khronos Group: Collada. https://www.khronos.org/collada/
40. Khronos Group: GL Transmission Format (glTF). https://www.khronos.org/gltf/
41. Lengyel, E.: Open Game Engine Exchange (OpenGEX). https://opengex.org/
42. Pharr, M., Jakob, W., Humphreys, G.: Pbrt-v3 renderer input file format. https://www.pbrt.org/fileformat-v3.html
43. Pixar: Universal Scene Description (USD). https://graphics.pixar.com/usd/docs/index.html
44. Sony Pictures Imageworks, Industrial Light & Magic: Alembic. https://www.alembic.io/

Materials

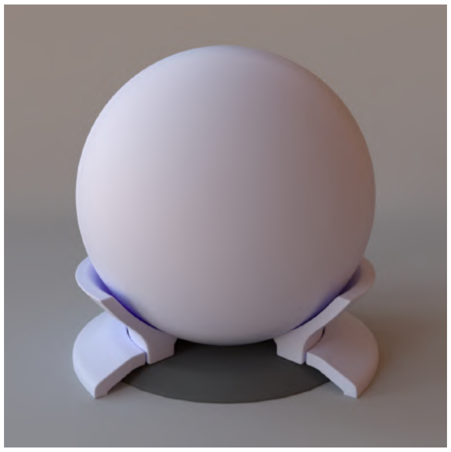
45. Lucasfilm: MaterialX. https://www.materialx.org/
46. Nvidia: Material Definition Lanugage (MDL). https://www.nvidia.com/en-us/design-visualization/technologies/material-definition-language/
47. Realistic Graphics Lab: Material database. https://rgl.epfl.ch/materials
48. Sony Pictures Imageworks: Open Shading Language (OSL). https://github.com/imageworks/OpenShadingLanguage

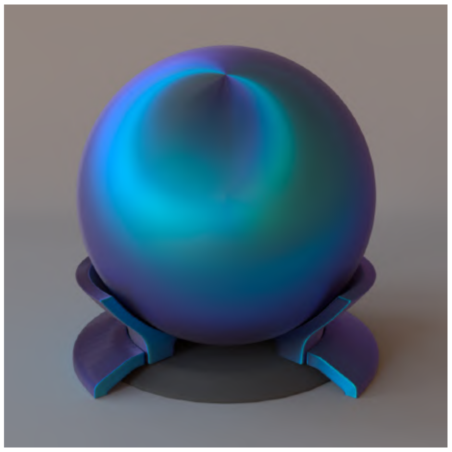
Renderers

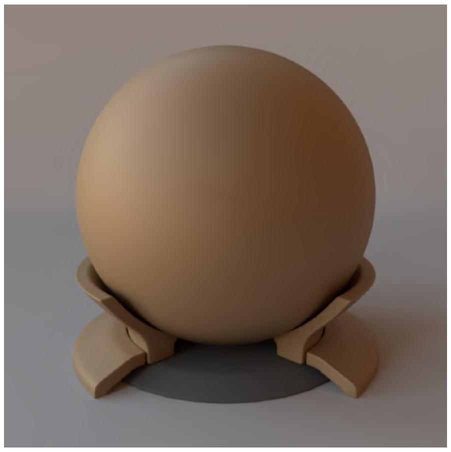
49. LuxCoreRender renderer. https://luxcorerender.org/
50. Blender Project: Cycles renderer. https://docs.blender.org/manual/en/latest/render/cycles/index.html
51. Chaos Czech a.s.: Corona renderer. https://corona-renderer.com/
52. Chaos Group: V-ray for 3ds max. https://www.chaosgroup.com/vray/3ds-max

53. Jakob, W.: Mitsuba 0.6.0 renderer. https://github.com/mitsuba-renderer/mitsuba
54. Pharr, M., Jakob, W., Humphreys, G.: PBRT-v3 renderer. https://github.com/mmp/pbrt-v3
55. Solid Angle: Arnold renderer. https://www.arnoldrenderer.com/

B Figures


(a) White acrylic felt


(b) *Morpho Menelaus* butterfly wing


(c) Cardboard

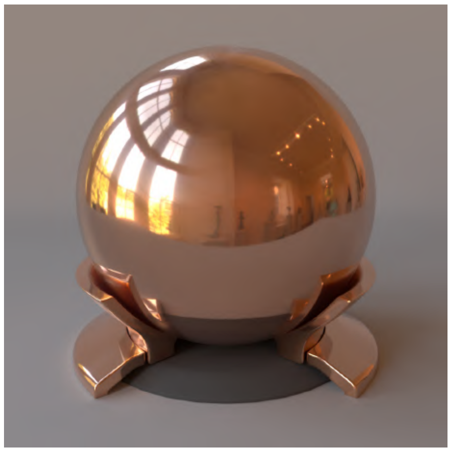

(d) Copper sheet

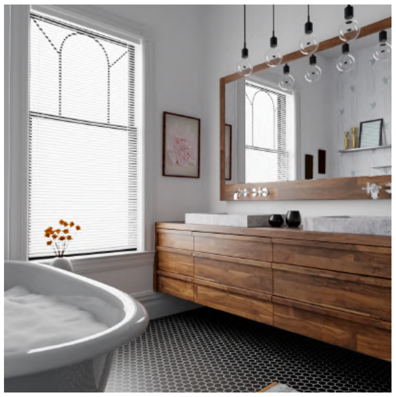
Fig. 2. Different types of materials from the *Realistic Graphics Lab* Material Database

(a) The Wooden Staircase; *Wig42*

(b) Country Kitchen; *Jay-Artist*

(c) Veach, Ajar; *B. Bitterli*

(d) Salle de Bain; *nacimus*


(e) Contemporary Bathroom; *Mareck*

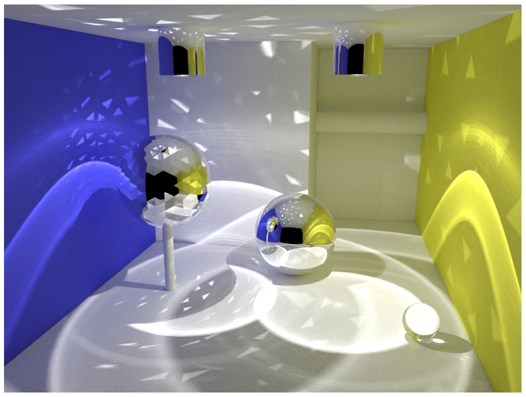
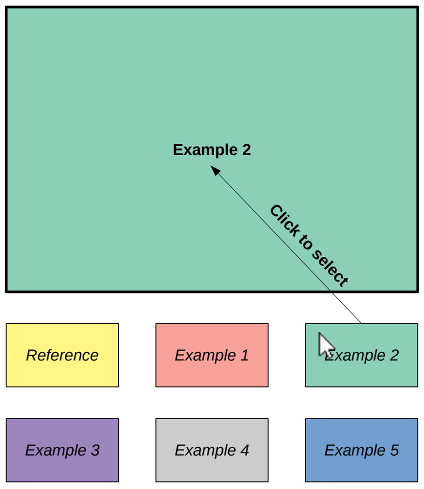
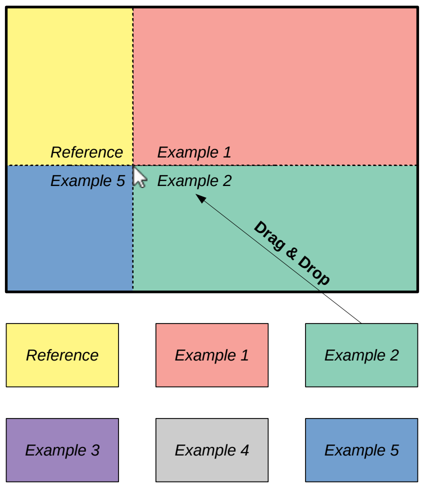

(f) Mirror Balls; *T. Hachisuka*

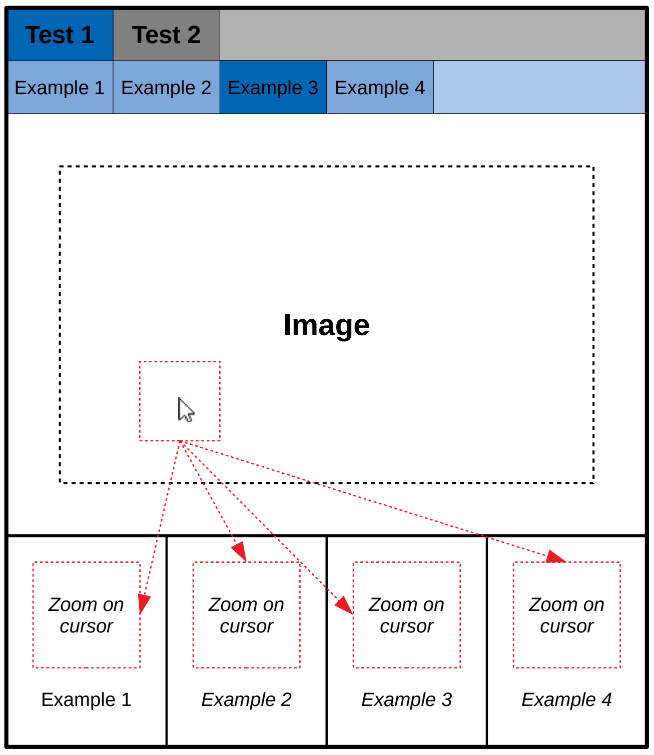
Fig. 3. Frequently used scenes in the articles of the survey.



(a) Selective View



(b) Crosshair View



(c) Hierarchical View

Fig. 4. General structure of the comparison views showcased in the articles of the survey