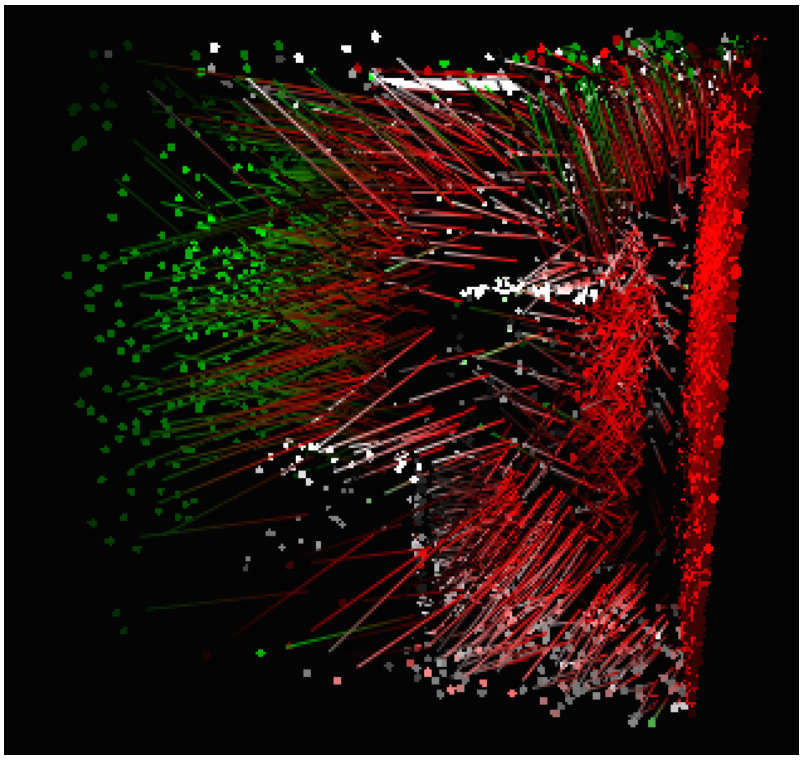
A Comparative Information Visualization Approach to Physically-Based Rendering

Master's Thesis



Gerard Simons

A Comparative Information Visualization Approach to Physically-Based Rendering

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Gerard Simons
born in Weert, the Netherlands

Computer Graphics and Visualization Group
Department of Intelligent Systems
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Karlsruhe Institute of Technology
Hermann-von-Helmholtz-Platz 1
76344 Eggenstein-Leopoldshafen
Karlsruhe, Germany
www.kit.edu

A Comparative Information Visualization Approach to Physically-Based Rendering

Author: Gerard Simons
Student id: 1358413
Email: g.j.c.simons@student.tudelft.nl

Abstract

In this work we present a novel information visualization framework to gain insight into the light transport in a physically-based rendering setting. The framework consists of a sampling-based data reduction technique, an extended interactive parallel coordinates plot that provides an overview of the attributes linked to each light sample, 2D and 3D heat maps to represent different aspects of the rendering process, as well as a three-dimensional view to display and animate the light path transportation throughout the scene. Furthermore the interactivity of the visualizations enables users to guide the rendering process. We show several applications that make use of the presented framework, including differential light transport visualization to optimize lighting in a scene, thereby improving the rendering times when changing scene properties and finding and resolving rendering bottlenecks.

Thesis Committee:

Chair:	Prof. Dr. E. Eisemann, Faculty EEMCS, TU Delft
University supervisor:	Prof. Dr. M. Eisemann, Cologne University of Applied Sciences
Committee Member:	Dr. K.V. Hindriks, Faculty EEMCS, TU Delft

Preface

This work before you today is the culmination of my research done over the past year. It would not have been possible without the participation of several key individuals.

First I want to thank Elmar Eisemann for making this thesis and my trip to Karlsruhe possible. This brings me to Carsten Dachsbacher, who was very welcoming and kind in receiving me for my stay in Karlsruhe and in getting me started. I also want to thank Marco Ament and David Koerner, who helped me out in orienting on the problem at the start. David, together with Sebastian Herholz were also great as reviewers of my thesis and in generating new exciting ideas.

Undoubtedly, most of my thanks should go to Martin Eisemann, who was able to take the time to meet with me through Skype almost every week and provide valuable feedback on almost every aspect of the work involved. I am incredibly grateful for his effort, and will surely miss our weekly meetings.

Gerard Simons
Delft, the Netherlands
June 5, 2015

Contents

List of figures

Chapter 1

Introduction

Physically-based rendering algorithms are able to synthesize realistic images of astonishing beauty that are difficult to distinguish from actual photographs. While the theory of light transport is well understood, and despite recent advancements, the complexity of a rendering task may lead to computation times ranging from minutes to hours or even days for a single image. This makes it especially difficult for designers, architects and engineers to quickly observe how changes to the scene, such as adding or removing objects, or changes in the lighting or materials affect the light transport and how this eventually influences the final appearance of the image and the computation time of the rendering algorithm. In other scientific fields such as biology, visualizing light distribution has some concrete practical applications, for example in determining how light is distributed between different lighting settings so as to optimize the lights received by plants, which is important for photosynthesis.

Although various visualization approaches to light transport have been developed, we do not know of any that incorporate general information visualization techniques which also allows a user to interact with the rendering process.

Recently, it has been shown how visualization techniques can increase scene understanding and the productivity of users faced with global illumination tasks [21]. Specialized visualization tools, mostly based on light probes, provide local insight into the light transport, how certain caustics are created and how much light is reached in a certain position. The use of general and established visualization tools for rendering tasks is a research area still in its infancy, but with promising initial results for rendering tasks such as parameter optimization in photon mapping relaxation algorithms [1] or for debugging of rendering systems [14].

Specifically, challenges faced by engineers such as finding out how light is distributed, what the effect of certain objects is on the light distribution are questions we would like to address. Other challenges include finding parts of the scene that require an inordinate amount of computation time, called rendering bottlenecks or finding sources that cause a significant amount of noise in the final image. Enhancing a user's control over the rendering process in order to improve the convergence is another challenge we want to address. We feel that all these issues essentially stem from a lack of control and feedback in current renderers. We aim to provide both of these qualities by augmenting a state-of-the-art renderer

with interactive visualization tools.

To summarize, the main research questions this thesis addresses are:

- Can information visualization techniques improve the understanding of light transport?
- Can these visualization techniques be used to guide and improve the rendering process?

The main contribution to be made by this work is a comparative light path visualization framework intended to improve the users understanding of the light transport within a single scene and between different scenes. Our framework is also able to give a rapid and comprehensive feedback to the user with only a fraction of the samples usually required for a full render. These samples, which do not only contain the usual attribute of color, but also additional relevant data, may be used for visualization purposes, but may also be used to guide the rendering process. We make use of several general information visualization techniques, such as parallel coordinate plots [8], with established interaction metaphors like brushing and subset selection [25]. The parallel coordinates plot is used to summarize data and enable quick user interaction. We also offer various 2D and 3D heat map visualizations that displays how information such as radiance, throughput and depth are distributed throughout a scene. Furthermore a 3D view was conceived that allows a user to brush certain path trajectories and animate them to gain insight into how light is propagated. We have placed additional emphasis on visualizing the differences of the data generated by two render instances. Finally, we will see how these various interaction techniques may guide the rendering and we demonstrate an especially powerful rendering technique called incremental rendering, where rendering is focused according to the difference in radiance distribution between two scenes.

Chapter 2

Related Work

Our work combines visualization techniques with state-of-the-art Monte Carlo rendering techniques, which means that the related work falls within these two categories. Because both fields are vast, we will only discuss the most relevant work done in each field in this section.

2.1 Physically-Based Rendering

Simulating real-world light transport involves solving the rendering equation [10] which is a comprehensive and very elegant description of the visually most important light-matter interactions:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega f_r(x,\omega_i,\omega_o) L_i(x,w_i)(\omega_i \cdot \vec{n}) d\omega_i \tag{2.1}$$

This is a slightly simplified version (omitting the wavelength and time parameter), which describes the outgoing radiance L_o at position x into direction ω_o, consisting of the emitted light L_e at position x and the reflected incoming light at x, which is described as the integral over all sampling directions. Here, f_r is the bidirectional reflectance distribution function (BRDF) and L_i is the incoming light from direction ω_i scaled by the cosine of the angle between ω_i and the surface normal n at x.

Unfortunately, solving this equation analytically is considered impossible for non-trivial scenes, which is why sampling techniques such as Monte Carlo (MC) [5] are used to approximate the integral by taking point samples based on an appropriate probability distribution function. One popular way of doing so is by using path-tracing algorithms, where rays are shot from the eye to compute the final color value of a pixel. Another important improvement called bi-directional path-tracing was made where rays were not only shot from the eye, but also from the light source, which could increase the rate of convergence [12].

Despite the slow convergence of Monte Carlo rendering techniques, they proved to be more or less efficient for most scenes. Nevertheless, alternatives to solving the rendering equations were proposed, such as the photon mapping algorithm [9]. Photon mapping simulates the light transport by imitating real photons within the scene, which is beneficial for

effects such as caustics, caused by light reflected or refracted at curved surfaces. Algorithms based on Metropolis Light Transport [28] apply local mutations to the light paths to improve the convergence in complex lighting situations.

The complexity of the light path simulations requires high amounts of computational power to synthesize the final image. Since publishing their seminal work [29], the according renderers have become highly efficient [30] but are still far from real-time for complex scenes. This poses a problem to designers who want to optimize their scene *before* it is fully rendered. Whenever changes to the scene or viewpoint are applied a costly re-rendering is required. Our solution can visualize the main differences within a scene with a fraction of the required samples, providing a much faster feedback mechanism, which also contains more information than would be usually visible from a rendered result.

EMBREE [30] is a renderer employing various Intel-specific optimizations in order to significantly reduce the computational cost per ray traced. Other advancements seek to reduce the number of required paths for a suitable result. Adaptive importance sampling [19] was an important work in trying to allocate sampling according to the regions of the image which maintained the greatest variability. Another interesting approach proposed a path re-using algorithms where paths do not contribute to a single pixel but rather to multiple pixels [2]. Constructing polynomial filters able to construct suitable results using a much lower sampling rate, was another approach [20]. See [31] for a comprehensive overview of recent advancements in Monte Carlo rendering.

Other, more recent work, aims to improve the rendering times of specific effects such as soft shadows [16] or depth of field [27]. Coherence exploiting solutions, such as [22] exploit frame-to-frame coherence to improve interactivity by replacing objects by pseudo-objects to improve rendering time. Research by [18] and [23] explained how coherence between frames in animations is usually very high and devised a way of accelerating computation by caching and re-using computation of certain scene objects. Similarly, our incremental rendering technique can be thought of as exploiting frame coherence on a very general level, without looking specifically at spatial or temporal coherence but rather at the radiance recorded in two frames.

2.2 Information Visualization

The goal of information visualization is to convey as much information as effectively as possible to humans. Humans, unlike computers, are not well equipped for parsing large volumes of data. We are however much more adept at distinguishing patterns in graphical primitives. Information visualization exploits this by encoding information using colors and shapes.

Scatterplots [4] are a popular way of visualizing data, by plotting data points as glyphs (typically circles or crosses) along two or three orthogonal axes. Each coordinate is therefore able to encode information about one data dimension. Furthermore we can add colors and different glyph sizes to encode additional data dimensions. Eventually, however, every useful encoding, which can still be effectively observed by humans, is exhausted and it will be very difficult to display datasets of arbitrary dimensionality using scatter plots.

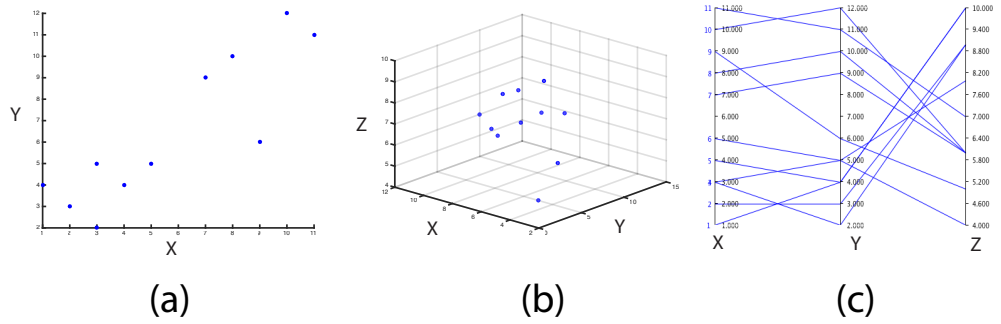


Figure 2.1: (a) shows a 2D scatterplot, (b) a 3D scatterplot and finally (c) shows the same data depicted in a parallel coordinates plot.

This is why parallel coordinates plots [8] are useful; a parallel coordinates plot depicts N-dimensional data by displaying axes in parallel rather than orthogonally, allowing for an arbitrary number of data dimensions. A data point is then plotted as a poly-line by connecting the points on each axis.

Mapping scalar values to colors is another important aspect of information visualization worth discussing. By applying colors to scalar fields we can distinguish differences in values much more easily than we would by simply looking at numbers alone. A popular color map often seen in scientific visualization tools is the rainbow color map. This color map is based on the ordering of colors in the spectrum of visible light, ranging from blue to red. Unfortunately this color map is considered ineffective as the the color range does not constitute a natural ordering, and because the perceptual resolution is not uniformly distributed across its range. A more suitable color map according to [17], is the divergent color map which is especially effective to show differences for a data dimension with a natural center, and the hot body color map, which unlike the rainbow colors, has a natural ordering whose perceptive resolution is also more uniformly distributed. See 2.2 for a comparison of the three color maps.

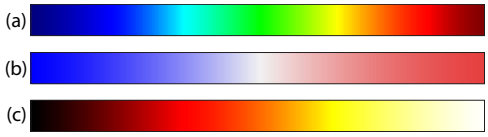


Figure 2.2: (a) shows the popular - but ineffective - rainbow color map. (b) shows a divergent color map, and (c) shows the hot body color map

2.3 Visualization of Light Transport

The visualization of light transport is gaining popularity in the field of computer graphics and visualization. Earlier approaches used a signal processing framework to visualize the

frequency content of light and how it changes as it interacts with different materials [6], or made use of visualization techniques to optimize the parameter space of photon mapping algorithms [1]. A more general and more related work is the visual dynamic analysis framework [14], that is able to visualize light paths, which is mainly intended for debugging and academic purposes. Similarly, rtVTK [7], a collection of programming and layered visualization tools, is aimed to help users explore the computational elements of a rendering algorithm.

Other research in the field provides visualization tools together with editing techniques. Light path visualization, clustering and modification [24] was employed to offer artists more freedom in creating suitable lighting effects. By allowing for a brushing mechanism, users could select and cluster light paths, as well as better comprehend them through visualization. Users could also manipulate so-called proxy objects which existed only to influence light transport, so that users could change shadows using manipulation mechanisms they were already familiar with. Similarly BendyLights [11] is a light-editing tool that allows progressive bending of light paths in free space, empowering artists with more artistic freedom, while sacrificing physically based correctness.

To our knowledge, Reiner et al.[21] were the first to primarily focus on the visualization aspect of light transport. They created a suite of visualization tools aimed to help a general audience of computer graphics consumers to understand light transport. By enriching photons in their photon mapping algorithm with additional information such as radiance and object interactions, they made it possible to create meaningful visualizations of the light transport within a scene. Other than false-color rendering, which colors surfaces according to the received brightness, the visualization tools make use of light probes to collect data about photons passing through it, which means that local data of the light transport is collected, such as directional distribution, the overall path trajectory, specific photon paths and volumetric properties. They also conducted a formal user study showing the usefulness of visualizations as a way to help users solve different global illumination tasks.

In a way, our work is similar to theirs, but with a larger focus on using general information visualization techniques to display comparative properties of two scenes whilst also providing an immediate interaction with the renderer which may guide and improve the rendering process.

Chapter 3

Overview

In this next section we will discuss the various aspects of our light path visualization approach. We will discuss the specifics of the relevant data in section 3.1 and how we deal with its volume in section 3.2, after which we will explain the various techniques employed to display this data: the parallel coordinates plot in section 3.3, the render view in section 3.4 and finally, the scene view in section 3.5. We will also exhibit our user-guided rendering technique and the incremental rendering technique in section 3.6.

An important property of these visualizations tools is that they adhere to the the brushing-and-linking paradigm [3], popular in information visualization, where the brushing of data in one view is automatically reflected in other views. Our visualizations are also interactive with respect to the renderer, which helps the user to guide the rendering process. The comparative nature of our tool stems from the possibility to compare two different scenes. The visualizations for these scenes are then color-coded per renderer, or given a side-by-side view of the data. This allows one to quickly observe the differences in data between two scenes. An overview of our application with all its views can be seen in 3.5

3.1 Data

In this section we will discuss the data we decided to collect to make the proposed visualization and interaction techniques possible. Our data is generated using the EMBREE [30] renderer, which is a unidirectional path tracer that shoots rays from the eye into the scene and recursively extends it in order to refine the final color associated with it. Averaging over multiple paths gives us the final color value of the pixel. We sample and maintain several of these light paths, its intersections with the scene and information about light collected at each intersection. Below is an overview of the data which we maintain.

Figure 3.1 shows a schematic overview of unidirectional path-tracing and the data involved. The solid line constitutes a light path, which is shot from the eye and is reflected and refracted by various elements in the scene. The solid line denotes the entire path trajectory, a black dot represents an intersection and light-related data is encoded by the dashed line.

Each path originates from the lens of the camera and intersects the image plane at a certain position which defines the pixel to which it will contribute. The radiance of a path

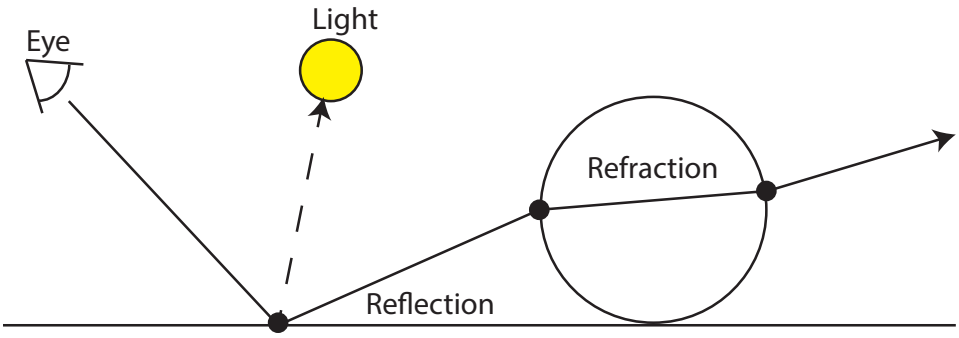


Figure 3.1: A schematic overview of path-tracing showing how a path starting from the eye is reflected and refracted through the scene.

is determined by the accumulation of energy received along its path. Depending on various properties such as the angle at which a light path intersects various surfaces and the material it intersects with, the degree to which it contributes to the final pixel value is determined. This attribute is called the throughput of a light path.

The number of intersection points a path contains is denoted by its depth attribute. Each intersection has information associated with it such as its order in the light path, called the bounce number, its position, the radiance it received, the kind of material interaction it underwent, such as a specular reflection or refraction, and an object ID that indicates the object it interacts with. Data about light contains information about how much energy it received from a particular light source. It also contains information about a possible object that prevented it from reaching a light source, marked by the occluder ID. A complete overview of the data is given below.

- Light Path
  - Pixel Position
  - Radiance
  - Throughput
  - Depth
- Intersection
  - Bounce Number
  - Position
  - Radiance
  - Object ID
  - Interaction Type
- Light

- Radiance
- Occluder ID

At this point, it is important to note that other data attributes are available, such as the time when a ray was shot in terms of the camera's shutter time, and two lens parameters. These attributes might prove useful for other applications, but were not necessary for our applications and were therefore omitted

3.2 Data Reduction

Ray-tracing techniques are known for the large volume of rays required to obtain a high fidelity image, where often hundreds or even thousands of samples per pixel are necessary. This also means that the volume of data about the light paths would become too excessive very quickly, making our application unresponsive.Furthermore, too many samples would clutter our visualizations, in particular the parallel coordinates plot, rendering it ineffective. In figure 3.2 we see how sampling can reduce clutter. (a) shows a parallel coordinates plot with 500,000 data points making individual lines impossible to see. By sampling 5,000 data points in (b), individual lines are more clearly visible, showing their angular information as well as giving the user a better sense of the density and distribution of the data.

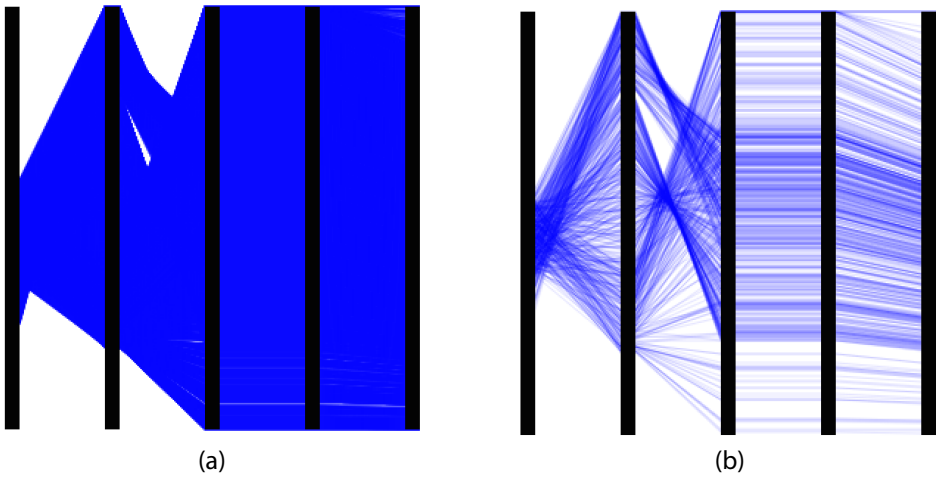


Figure 3.2: How parallel coordinates plots suffer from overdrawing when too many elements are added. (a) is a parallel coordinates plot with 500,000 data points and (b) a parallel coordinates plot with 5,000 data points.

3.2.1 Sampling

To obtain these samples, we use two possible sampling strategies. One method is by sampling based on a certain property, for example by taking N paths that have the most radiance.

This sampling strategy is used in section 4.7 to gather possible fireflies. Another way is by keeping a sample set of the data that is able to represent the whole corpus in the best possible way. To do this, we maintain a set of histograms, one for each data dimension, which we update as the rendering processes generate new data. We then randomly sample (with replacement) from a set of data accumulated from the rendering process and keep the set whose histogram matches the true distribution most accurately. This strategy was inspired by bootstrapping used in statistics [26], although we do not use it to infer any additional statistics.

A histogram can be defined mathematically as having a function m_i that counts the number of observations that fall into each of its disjoint categories (called the bins). The histogram m_i, with n observations and k bins, meets the following condition:

$$n = \sum_{i=1}^{k} m_i \tag{3.1}$$

A parameter N determines how many paths we actually sample and use for our visualizations. This value is typically set in the range of 5,000 to 10,000 so that the application maintains responsiveness whilst also providing enough detail to the data. We maintain two separate sets of data for each renderer, one which is used for the actual visualization and one which acts as a buffer for new data. When the buffer reaches its capacity, we join the two datasets and sample N new samples from it and determine how well it fits the true distribution by comparing it to the full histogram using a distance metric:

$$D = \sum_{i=1} |m_i/M - n_i/N| \tag{3.2}$$

Where m_i and n_i is the number of occurrences for bin i and M and N denote the total number of elements in the two histograms. By summing up these distances for every data dimension, we get an estimate for the goodness of fit for the sampled data. If the value is lower than before we found a better candidate, if not, we keep the previous set and continue. See a simple example of sampling and histogram matching in figure 3.3, where (a) shows the true distribution, (b) and (c) show two different samples, where (c) more closely matches (a) than (b).

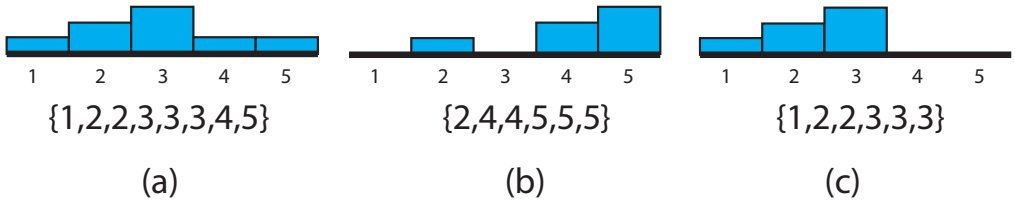


Figure 3.3: (a) shows the true distribution and its histogram, (b) and (c) are two independent samples from the dataset in (a), where the data in (c) matches the distribution in (a) most closely.

3.2.2 Distributions

In addition to a sampled set of light paths that most accurately represent the data's distribution we also maintain several higher dimensional distributions. First we maintain several 2D distributions in the image domain which describes data such as radiance, depth and throughput. We will discuss these distributions in section 3.4. Another 3D distribution is maintained that keeps track of the energy distribution within the scene. We will discuss this further in 3.5.

3.3 Parallel Coordinates Plot

As mentioned above, parallel coordinates plots are very adept at displaying a high-dimensional set of data in 2D space, as they do not suffer from a limitation on the number of axes that can be used. In the previous sections we described our data and showed that it is indeed high-dimensional, especially due to the radiance and positional triples. In order to still allow the user an overall overview of the data we use the parallel coordinates plotting technique [8]. Additionally brushing data is very conveniently done in a parallel coordinates plot by clicking and dragging across an axis (see 3.4).

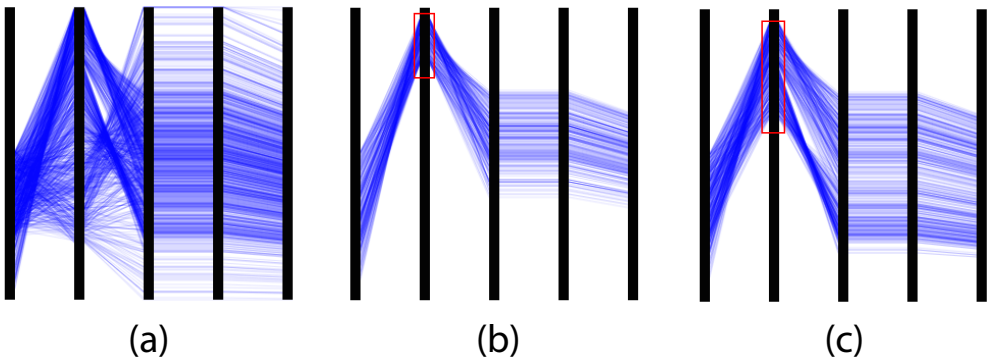


Figure 3.4: How brushing works in a parallel coordinates plot. In (a) we see all the data displayed. in (b) we have clicked and dragged on an axis to brush some of the data whose lines pass through the area indicated by red. (c) increases the size of the brushed area, and as such the number of data points to be displayed.

From section 3.1 we recall that the data on our sampled light paths consists of multiple intersections which may again have data associated with it from multiple light sources. Because of these N-ary relationships, we use three separate parallel coordinates plots, one for data on paths, one for its constituent intersections and one for the energy received by each light source, which are then linked together. Poly-lines are color-coded according to the renderer that generated the data. See figure 3.5 for an example, where the data about paths, intersections and lights are displayed. Here the lines are color-coded according to the renderer the data belongs to.

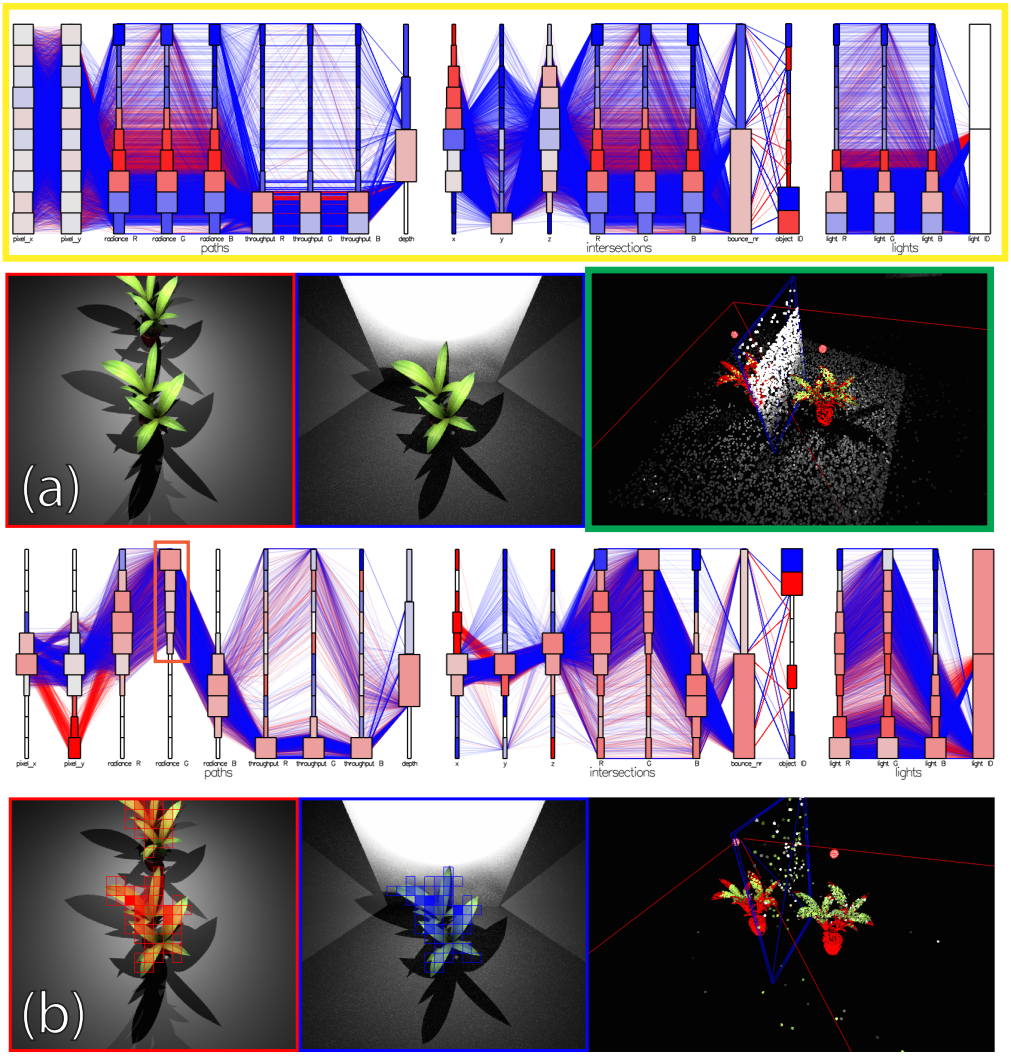


Figure 3.5: The parallel coordinates plot (marked in yellow) along with the other views : the render views, marked in red and blue (the markings correspond to the colors used in the parallel coordinates plot), and the scene view (highlighted in green). (a) shows all the data associated, where in (b) we have brushed light paths whose energy has a strong green component (indicated by the red rectangle on the axis).

As previously mentioned, over-drawing is a significant problem in parallel coordinates plots, as too many poly-lines may clutter the visualization technique, making it ineffective. Although this problem is already diminished by the sampling strategy, discussed in section 3.2, it is still possible that too many lines overlap, depending on the size of the sample set and the nature of the data. Also, another problem arises when too many lines overlap, as an incorrect impression may be given that fewer data points are present than is actually the case. A possible improvement over traditional parallel coordinates plots which we have

employed in our current work, uses the process of binning [15] to show how many data points lie within discrete regions of each axis.

Another reason for using this binning approach is that we often want to show data from two renderers and compare them. This is why we use the bins to highlight the color of the data most points in the bin belong to. This means, that the bins on each axis are color-coded according to the ratio of membership of the data points in the bin. Bin membership is defined as $M_1(b,x)$, or the number of data points in bin b for data dimension x belonging to dataset one and $M_2(b,x)$ as the bin membership value for dataset two. The ratio of these two bin membership values determines the color of the bin: $M_{ratio}(b,x) = M_1(b,x) - M_2(b,x)$ where $M_{ratio}(b,x) \in [-1,1]$.

See figure 3.5 for an example, where we use a divergent color map (see figure 2.2) to color code each bin according to the ratio of membership of the data points inside the bin. A completely red bin for example, indicates that all data points belong to the renderer corresponding to the red color, and blue corresponding to the complementary renderer. A gray color is used to indicate a bin whose ratio is perfectly balanced between the two. Additionally we encode the size of each bin to indicate how many data points reside in the bin, regardless of membership.

The parallel coordinates plot is especially useful in our application to give an overview of the data: Angular information may display certain correlational patterns and color highlights in the axis bins show subspaces of the regions most unique to a specific renderer. Furthermore it allows for an intuitive way of brushing the data. By interacting with the axes the visualizations automatically update to reflect information about the brushed data.

In 3.5 we see an example of the parallel coordinates plot (marked in yellow) along with the other views in two scenes of two plants where in one scene we have added a highly reflective wall. The parallel coordinates plots provides a convenient and quick overview of the data in (a). In one scene a wall is placed between the two plants. in (b) we have brushed the data of light paths that have a lot of green energy (highlighted in orange). We see how the render views now highlights those parts of the image that match the selection as well as the updated intersection points in the 3D view. The bins in the parallel coordinates plots are also updated, and because in the red scene another plant is visible most bins then appear as red.

3.4 Render View

The render views (highlighted in red and blue in figure 3.5) display the current result of the rendering processes. As the rendering process progresses these views are kept up-to-date with the latest state of the rendering process.

3.4.1 Heat Maps

We also mentioned the various 2D distributions that we maintain in addition to the sampled set of data. The render view is able to display the distributions of various data dimensions as heat maps on top of the image. Furthermore, the render view is able to display a heat map visualization of the currently brushed light paths' pixel positions. This can either be

visualized using the hot body color map or as a monochromatic heat map with interpolated opacity so as not to obstruct too much of the image. We also allow the user to smooth the distributions, in order to get rid of noise. This can be especially useful when using the distributions to guide the rendering (see sections 3.6, 4.5 and 4.4.)

In figure 3.6 we see the various heat map visualizations of distributions obtained by rendering an instance of the Cornell box scene. The various distributions that may be visualized as heat maps on top of the render preview. (a) shows the throughput distribution, note the high throughput around the light source and the slightly higher throughput on the boxes. (b) shows the depth distribution, note the lower depth distribution on the boxes and the very low depth distribution at the light source, which is due to the lack of actual geometrical data associated with the light source (paths simply pass through and terminate as they leave the scene). Sub-figure (c) visualizes the average radiance distribution and (d) uses the overlaid heat map to show a selection of paths whose energy is high in the green spectrum. Note how the green wall is selected, but also other regions whose paths may at one point intersect with the green wall. Later on we will revisit these heat maps to show how they can also be used to guide the rendering process.

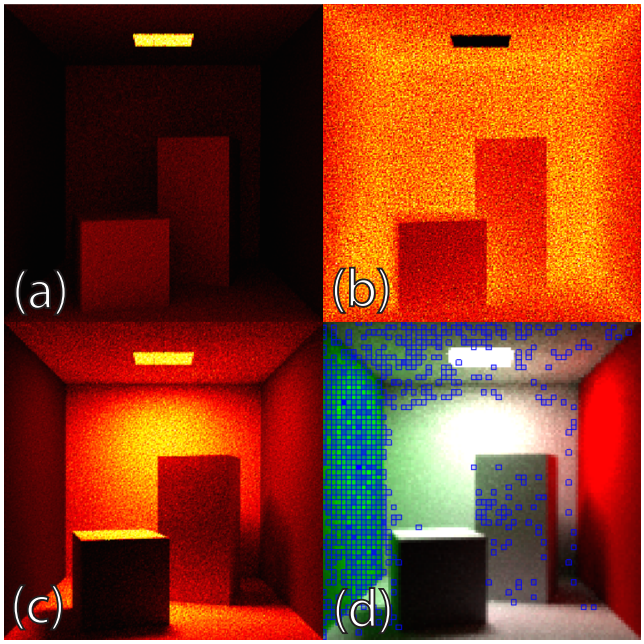


Figure 3.6: Heat map visualizations of the image-based distributions. Here the throughput, depth, average radiance and selection are visualized in (a) - (d).

3.4.2 Interactivity

The render view also provides an interactive aspect. A user might often be interested in a distinct region in the image, such as a specular highlight, the render view also allows for a user to scribble on the image. The area enclosed by this scribble is then used to find paths

that originate inside this region. A user might also brush a rectangular shaped part of the image using the parallel coordinates plot, but by scribbling we can also use non-regularly shaped regions. Also, because the data of the final image is present at the render view, brushing data is often more intuitive than having to brush the separate pixel axes in the parallel coordinates plot.

3.5 Scene View

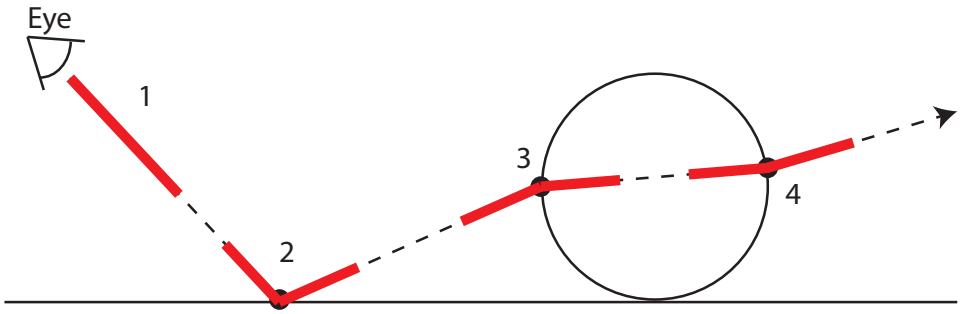


Figure 3.7: A diagram demonstrating the light path animation. Each red line segment indicates a specific point in time as it is animated across its entire trajectory (indicated by the dashed line).

The scene view (marked in green in figure 3.5) aims to place data in its spatial context by rendering it in three dimensions. Path-tracing consists of rays recursively hitting geometry in the scene which makes displaying these paths along with its intersection points and the geometrical data an intuitive way of giving the user insight into how light is transported throughout a scene. Intersection points are colored according to the energy received at that particular point, whereas paths are colored according to the gross accumulated energy. Alternatively when we are more interested in how light is transported at each bounce, we can color the points according to the number of bounces it is distant from the light source.

An obvious way of visualizing the light paths would be to render them as poly-lines connecting the different intersection points. Unfortunately this would quickly cause too much clutter. It also would not convey any information about the direction of light. We have chosen to let the user animate the trajectory of the light path. A line segment is drawn along the entire light path which may be animated along all intersection points. See 3.7 for a schematic illustration of the animation and figure 3.8 for an example of the animation in our application.

3.5.1 Selection Mechanisms

The scene view also allows for an intuitive way of brushing data in the object space. An object ID in a scene description file is often associated with a group of vertices and faces that have a certain semantic meaning, such as a teapot or a chair. By clicking one of these

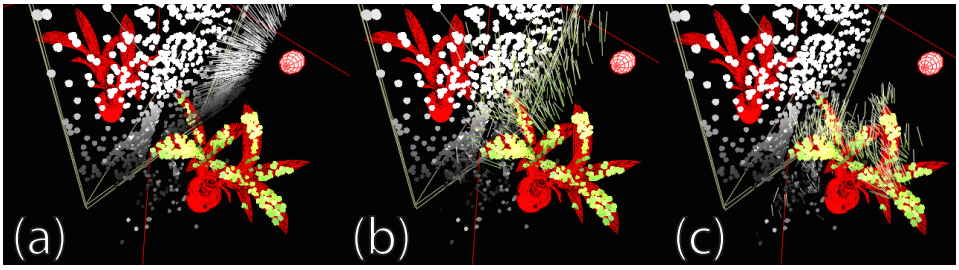


Figure 3.8: The animation of light paths starting from a light source. (a)-(c) shows three still frames of the light paths as they reflect of the wall unto the plant.

objects in the scene view the user may brush data regarding the selected object. We have identified four different mechanisms of selection in the scene view:

1. Object
2. Path
3. Interaction
4. Shadow

The object selection mode simply selects all intersection points residing on the surface on the selected object. The path selection mode is slightly more complex; it allows a user to select different objects light paths should interact with along the way. For example when a user is interested in light paths hitting object A and then hitting object B, the user selects object A and B consecutively. Interaction selection mode is used to display paths that interact with an object when the order of intersection is not important. For example, when selecting object A, all paths that eventually intersect with this object are selected. This distinguishes it from the object selection mode, because that mode only selects those intersection points on the object, whereas the interaction mode allows the entire path to be selected as long as at least one of its intersections lie on the object of interest. Lastly, we have the shadow selection mode which selects rays that hit the shadow of an object. At each intersection a shadow ray is cast to each light source to determine if the intersection point can be reached from the light source. When it cannot be reached the point is said to be occluded, and the energy for this light point is not taken into account. We will see in section 4.4 how this particular piece of information can help guide the rendering.

3.5.2 3D Heat Map

Displaying the intersection points and animating the trajectory between these points is useful in visualizing how light is transported in a scene. Something that is less obvious when looking at the scene in this manner, is how energy in general is distributed. Knowing where most of the energy of the scene ends up might be useful for various reasons, for example when we want to place an object that should receive a lot of light in order to increase its

luminance. Furthermore determining the energy distribution in a scene might also be useful to users involved with additional engineering or scientific challenges involved with light.

We create a 3D heat map of the energy distribution by using a sparse voxel octree [13]. Octrees may be used to store three-dimensional data in an efficient hierarchical fashion using a tree structure, where each node in the tree may consist of eight children. Voxels are rectangular blocks, often used as an alternative to triangular meshes to store geometrical data, which can be seen as a 3D version of pixels. Combining octrees and voxels allows for a hierarchical 3D model where the resolution of the models varies with the information it contains. An octree is often created by starting of with a single root node and recursively splitting the node in eight new nodes as more data is added. Recursion halts when a certain maximum depth is reached.

In our case we initialize the octree with a single node that encompasses the entire scene. We then compute the energy for each new intersection as the average of the R,G,B channels. We use the average rather than each channel separately so that we can easily observe how energy is distributed in a single view. Energy is therefore computed as such:

$$E = \frac{R+G+B}{3} \tag{3.3}$$

When new intersection data is received from the renderer, the correct node is found by finding a leaf-node in the tree which contains the point. This makes the tree useful as we can traverse the tree hierarchically starting from the top and only continue the search in branches that contain the point.

We render the octree by using voxels, where the size of the node determines the size of the voxel and the color is interpolated using a hot body color map. Alternatively, when we are interested in differences in the energy between two renderers, we can visualize the differences by subtracting the energies of the voxels of the two octrees and color coding this value using the red-blue divergent color map we also use in the parallel coordinates plot.

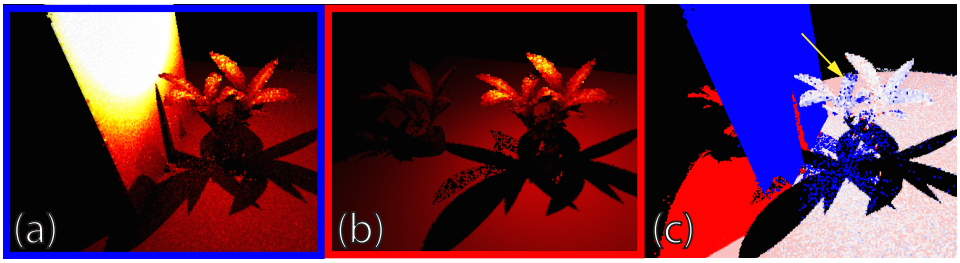


Figure 3.9: 3D heat map visualization of the energy distribution in two scenes. (a) shows the scene with a reflective wall, (b) without this wall and $c)$ visualizes the difference in energy for these two scenes. The colored borders around (a) and (b) correspond to the colors used in (c).

In figure 3.9 we see an example of the 3D heat map used to demonstrate the difference in energy distribution. In (a) we have placed a reflective wall. In (b) we see the energy distribution when there is no such reflective wall. In (c) the difference in energy is explicitly

rendered using the divergent color map (see 2.2, where blue indicates energy a region where the scene with the wall contains more energy and red areas indicate regions that receive more energy when there is no wall configured. We can clearly see some of the leaves that appear more blueish, especially the third from the left (marked by the yellow arrow). We also see some energy in the blue scene reaching the shadow of the leaves, indicated by the blue speckles in the shadow.

3.5.3 Gizmo

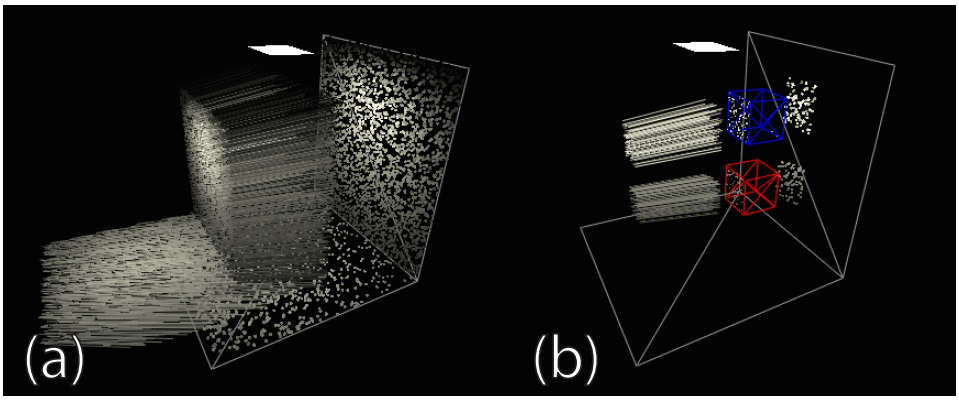


Figure 3.10: A simple example demonstrating how gizmos are able to select paths intersecting with it. (a) displays all light paths, whereas (b) displays only those light paths that intersect with the gizmos (the red and blue cubes).)

Sometimes information is not confined to a specific object but is confined to a certain region in the scene, a so-called region of interest (ROI). Reiner et al. [21] introduced a gizmo that could be placed in the scene to collect localized information about a specific subspace of the scene. We extend the use of the gizmo into our comparative visualization framework. By placing two gizmos, one for each renderer, and selecting only those paths that intersect with the gizmo, we can compare data about two distinct regions of interest. See figure 3.10 where (a) shows all sampled light paths and (b) shows only those light paths interacting with the gizmo.

3.6 User-guided Rendering

By default our renderers use a uniform sampling strategy when spawning new rays. That is to say, all parts of image receive a roughly equal amount of attention. When a user interacts with the visualization, different pixel regions in the render images may become highlighted. We have added the functionality that this distribution may be used to sample where new paths originate from. As such the user is able to guide the rendering according to the interest of the user, making the rendering process interactive. The reason why we think this is useful is threefold:

Firstly, we believe that by guiding the rendering through interaction with the visualization gives the user intuitive control over the rendering process. As we will see in the results section, it is able to improve the rendering by reducing noise locally, as well as give the user valuable information through the resulting change in the rendered result.

Secondly, by guiding the rendering the user may obtain more light path samples for a specific region, giving him or her more detailed information about this region.

Lastly, as we recall from section 3.4.1, we have several 2D distributions that can be visualized as heat maps. We can use these distributions as the sampling distribution for new rays. Using the radiance difference distribution can be especially effective. We call this technique incremental rendering.

3.6.1 Incremental Rendering

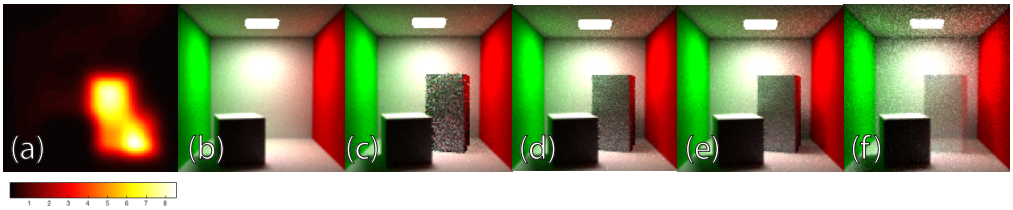


Figure 3.11: The incremental rendering technique applied to the Cornell box scene where the tall box is added in the second scene. (a) displays the radiance difference distribution which acts as a sampling distribution for new rays, (b) shows the input image, a rendered image of the scene without the tall box. (c)-(e) shows the result of our incremental rendering technique at 2,4 and 8 samples per pixel respectively. (f) shows a naive blending of the two images with 8 samples per pixel, as a comparison.

Incremental rendering can be seen as an especially useful instance of user-guided rendering. It is effective when we have two scenes that are very similar to each other, except for one or two objects that are either changed, added or removed, which causes the final image to change. Normally we would have to re-render the entire scene, but incremental rendering is designed to reuse samples for areas that were not affected by the change and resample regions that *were* changed. We can use the information of the sampled paths, and the information contained in the radiance distributions of the two scenes to compute a new sampling strategy whose distribution is based on the difference in radiance of the two images which may be deduced from only a few samples per pixel.

We see the steps involved with incremental rendering in figure 3.11, where it is applied to the Cornell box scene, in which we have a base version without the tall box, which is then introduced. We input the difference in radiance $((a))$ and the high fidelity base version (b) rendered with 128 samples per pixel. By continuing rendering we see the tall box slowly appearing in (c)-(e) with 2,4 and 8 more samples per pixel, respectively. In (f) we have added a naive blending of the base image with a uniform render of 8 samples per pixel where the box appears opaque and more noise is introduced. We will see a more detailed analytic comparison of this technique in the results section.

An important weighting parameter, the swapchain weight, determines how quickly the old values of the base image should be blended with new values from the incremental rendering. A low weight means the new values will mean results from the new scene will become more quickly apparent, but might also introduce more noise. This parameter may be set by the user depending on his or her desired outcome: a quick, but possibly noisy result or a slower but higher fidelity result. The optimal setting for this value may also depend on scene specific parameters and how large the influence of the object is on the result. As an additional improvement we use the data collected about our sampled paths to determine which paths hit the new object on the first bounce so that we may set the swapchain weight to zero for this region, to further increase convergence.

Chapter 4

Results

We have combined the previously proposed visualization techniques into a custom-built C++ OpenGL application. All visualization tools are created in a single OpenGL window where each sub-window displays a visualization view coupled with two instances of an adapted EMBREE [30] renderer. In this section we will discuss various example applications of our tool and the results which we obtained.

4.1 Light Distribution

Due to the complex nature of global illumination, light paths may, through various interactions with the scene, end up almost anywhere. Understanding light distribution is useful for artists, engineers and scientists, but is often very difficult to understand without a useful visualization tool.

In modern greenhouse environments, where artificial lighting is often employed to mimic sun light, it is important that plants receive a roughly equal distribution of light energy, so that they grow at roughly equal rates. Placement of these artificial light sources is therefore very important. Below we have created an example scenario containing a row of five plants that simulates a row of greenhouse plants. We compare two different versions of a greenhouse setting; one where we use two intense light sources and the other where we use five smaller light sources. We have taken care to ensure that the total sum of the energy of the light sources is the same.

See the result of using our application in figure 4.1 where we investigate the difference in energy distributions. The top row shows the results for two light sources, the bottom the result for five. (a) and (b) show the rendered result. (c) and (d) show the energy distribution using a hot body color map. The spheres on top are the light sources. Note the strong highlight in the center of the plants in (c). (d) shows a more uniform distribution. (e) and (f) show a close up of the three plants in the middle. The leaves of the plants in (f) show a more uniform distribution.

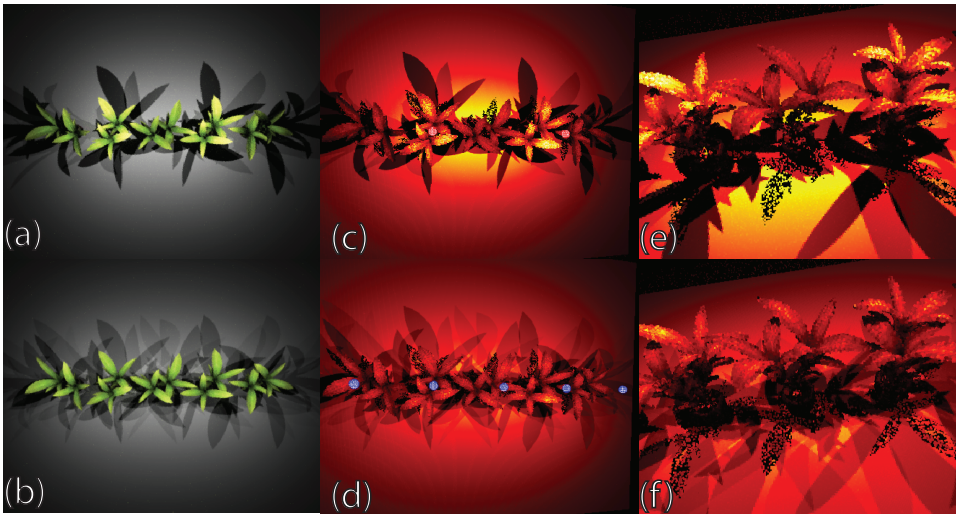


Figure 4.1: A 3D heat map visualization of two different lighting settings. The top row shows a scene with two large light sources, whereas the bottom row shows the result for five smaller light sources. (a) and (b) shows the rendered result, and (c)-(f) the visualization of the energy distribution.

4.2 Light Transportation

When dealing with light, it is not only interesting to know how much light reaches certain regions of a scene, it is also interesting to understand how light reaches these regions. In addition to our previous greenhouse example, we now look at another greenhouse scenario where we want to improve the light transportation. In greenhouse design, reflectors are often placed at key locations in order to increase the light reflected onto plants.

In figure 4.2 we have applied our visualization tools to a simple model of a greenhouse, which has a slanted roof with a reflector, where we want to place another reflector at a optimal location. The front of the model is made of glass which also helps to catch light, but it is not the light that we are interested in. In (a) we use the parallel coordinates to brush data about secondary bounces, that have a lot of energy (see the brushed markings on the "R" and "bounce_nr" axes). Furthermore we want to exclude paths that go through the glass window at the front, so we only brush bounces that are reflective rather than transmissive (see the axis "interaction_type"). (b) shows the intersection points that match the criterium set in (a). $(c)-(f)$ show a comparative path animation, where the red paths show the old scenario without a reflector, and blue where we have added a reflector in order to reflect the high energy bounces found in (b). The animation shows how red paths spread from the back wall diffusely to the rest of the greenhouse whereas blue paths are focused more towards the plant.

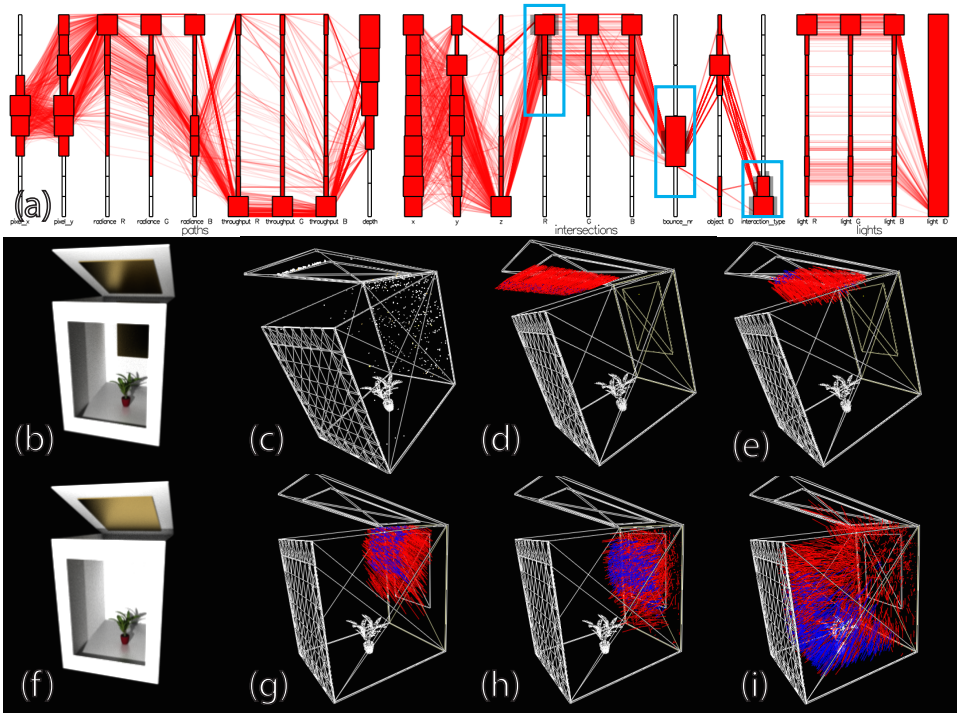


Figure 4.2: The greenhouse scenario where we use our tool to find the best placement for an additional reflector. (a) displays the parallel coordinates plot with blue markings indicating the data brushing that was used. (b) and (f) shows the two different scenarios, one with the additional reflector. (c) shows the intersection points of the paths selected by brushing by the parallel coordinates plot and (d),(e),(g),(h),(i) shows the animation of the light paths with (blue) and without (red) the reflector

4.3 Gizmo

In addition to the previous example, where we showed how adding a reflector focuses the light toward the ground, we want to show how the gizmo can be used to investigate two regions of interest. When we have no plant present in the greenhouse yet, but we want to find the most suitable location, we can use the gizmos to compare two distinct regions. We can see how using the gizmos finds the most suitable location for a candidate plant in figure 4.3. One possible position is closer to the reflector (marked by the red gizmo), the other is a bit further away (marked by the blue gizmo). By brushing the paths that intersect with these gizmos we can observe in the parallel coordinates plots how the radiance of the paths is greater by looking at the colors of the bins (highlighted in green). In (b) and (c) we see the path animation for the red gizmo and for the blue gizmo in (d) and (e). Note how the number of bright paths intersecting the blue gizmo is much lower than those intersecting the red gizmo. From this we can conclude that a plant located at the region of the red gizmo would receive more light from the reflectors.

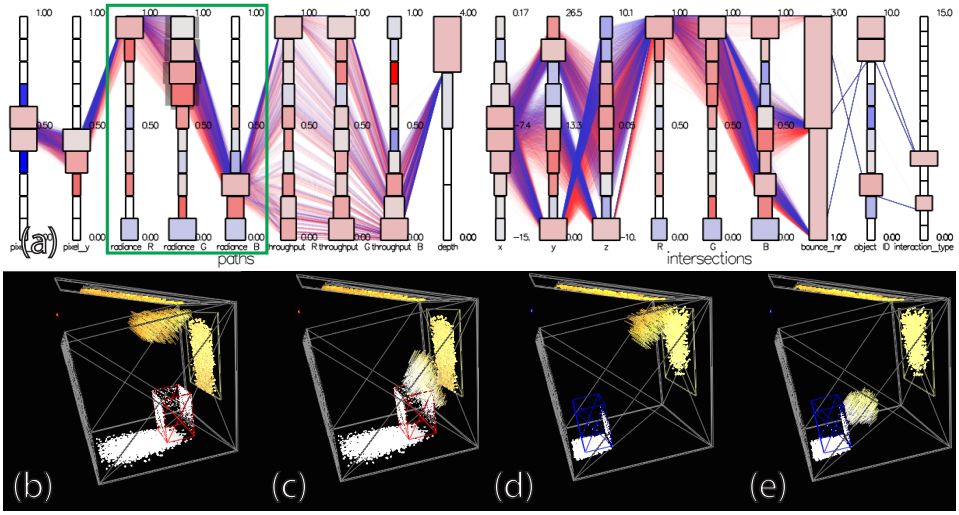


Figure 4.3: The use of gizmos to compare two possible candidate positions for a plant. (a) displays the data associated with each gizmo. (b) - (e) shows the animation of the paths interacting with the gizmo.

4.4 User-guided Rendering

We mentioned earlier in section 3.6, that our application could be used to guide the rendering in a specific way. Techniques such as this often exist in 3D content creation software, but are often limited to guiding the rendering to a rectangularly shaped region of the image space. In reality, effects are rarely confined to such a regularly shaped area of the image. Our visualization framework makes it possible to guide the rendering according to one of its features, present in the samples we take.

In figure 4.4 we see the result of applying this technique to a caustic that is cast on a bunny and on a textured wall. Most parts of the image, such as the grey walls, the floor and most of the textured wall do not require many samples, but other effects like caustics are more difficult to render and require more samples. This is also the case with the caustic projected by the glass sphere unto the bunny and the textured wall.

By gathering light path samples and selecting those that intersect with the shadow of the glass sphere (using the shadow selection described in section 3.5) we can use the distribution of these paths (seen in (a)) to sample new paths that interact with the shadow of the glass sphere. By combining uniform sampling with this guided sampling distribution we can get a significantly better result, as more samples are directed towards this region. (c) shows the result of using this distribution after 1024 uniform samples for an additional 1024 samples, whereas (b) shows the result of uniform sampling with 2048 samples and (d) is the reference image: a uniformly rendered image with 16,384 samples per pixel. We have also included various insets in the rows below. Overall the user-guided rendering performs significantly better than the uniformly sampled one, as can be seen from the SSIM and MSE metrics. Especially areas affected by the caustic perform much better (see blue and green

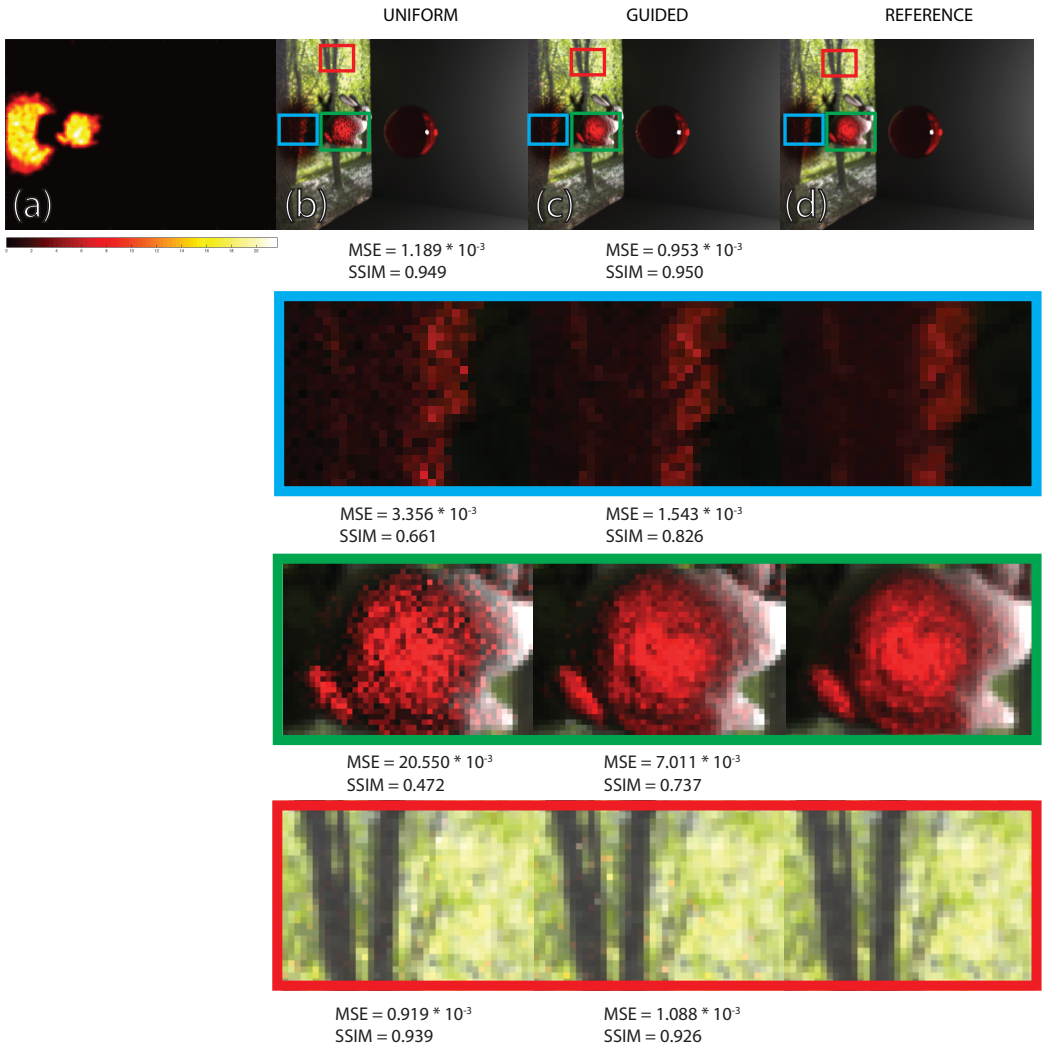


Figure 4.4: An example application of our user-guided rendering technique. (a) shows the sampling distribution used to guide the rendering based on light paths whose shadow rays intersect the glass sphere. (b) shows the result using a uniform sampling strategy with 2048 samples per pixel, whereas (c) uses our user-guided rendering technique, where 1024 samples were generated uniformly and 1024 samples using the distribution in (a). (c) shows a reference image using uniform sampling with 16,834 samples per pixel. Several color-coded insets are added and annotated with their MSE and SSIM results.

inset), while areas unaffected perform only slightly worse (red inset).

4.5 Incremental Rendering

The reader may recall from section 3.6.1 our incremental rendering technique, where by obtaining a coarse estimation of the difference in radiance distribution we can guide the rendering towards those parts of the images where radiance differs most. As such we only need one base image that we can then easily reuse to render different scene configurations more rapidly. This exploits the information we have recorded in our distributions and our sampled set of light paths.

We showed how this works in a Cornell Box scene but we want to show the results for a more complex scene. For this we have created a scene where we have a green and blue teapot with a glass sphere causing refractions and a distorted mirror that causes various reflections of the scene. The alternate scene contains a new red teapot which causes various additional reflections and refractions, as well as casting a new shadow.

See figure 4.5 for the results. (a) shows the input image rendered with 128 samples per pixel. (b) shows our alternate version rendered at 1 sample per pixel, which is just enough to get information about the radiance distribution (depicted in (c), smoothed five times with a 3x3 rectangular uniform filter). (d) shows the result after rendering with 32 additional samples using a swapchain weight of 64, using (c) as a sampling distribution. Note how not only the teapot but also the various reflections, the refraction and its shadow have appeared.

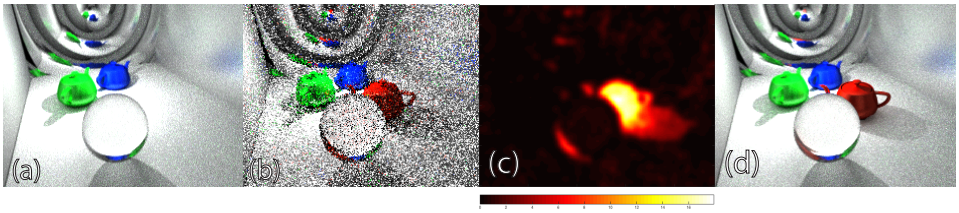


Figure 4.5: The steps involved when applying the incremental rendering technique to a more complex scene. (a) shows the input image, where the red teapot is not present. (b) is the rendered result of the new scene using only one sample per pixel. (c) is a visualization of the (smoothed) difference in radiance when the teapot is added. (d) is the result when sampling another 32 samples per pixel according the radiance difference distribution.

As mentioned before, different swapchain weights offer different results. A higher weight often means less noise but also means it takes longer for the incremental rendering to have its full effect. This effect manifests itself often in a less pronounced appearance of secondary effects such as shadows and reflections. Figure 4.6 shows a detailed comparison between different swapchain weights for this scene and a uniformly sampled rendering. The last column is added as reference, showing an image rendered with 2048 samples per pixel. The first row shows the overall image, with four additional rows showing insets of the original image. For each image we have also added the MSE and SSIM metric used to compare each image with the reference image. As can be seen, A weight of 64 performs

best in all of our test cases. The result derived from using a weight of 8 seems to reintroduce noise, whereas a weight of 512 is too high and does not allow the effects to fully appear. This is especially visible in the reflections and the refraction in the glass sphere. By using a weight of 64 the effects are much more opaque while also largely avoiding the introduction of new noise.

The MSE graph in (a) shows us that all weighted instances have a MSE value that is much lower than the uniform example. Note also how the uniform instance slowly overtakes the 8 and 512 weighted instance. The dimple in the blue line is also interesting, this can be explained by the teapot which rapidly appears in this instance, but the resulting increase in convergence is then quickly lost again as new noise accumulates. According to the SSIM metric, which we see in (b), the 512-weighted instance and the 64-weighted instance are to be regarded as roughly equal. Here the difference between these instances and the uniform instance remains significant as the uniformly sampled instance gains fidelity more slowly. The uniformly sampled instance also seems to gain on these two instances much more slowly, and is only able to overtake the 8-weighted instance. The 8-weighted instance has a much stronger decay when compared to the MSE metric and also recovers much more slowly.

4.6 Finding Rendering Bottlenecks

Differences in scene geometry and materials have different effects on the computational effort in creating a suitable rendered image. Some materials such as glass or metal are more complex and may require more light paths to account for reflections and refractions. Furthermore they may also require more depth, or number of rays, per light path. Scene properties that have a seemingly small visual effect may therefore have a considerable impact on the overall computation time. We call these elements rendering bottlenecks.

Usually, paths in ray-tracing algorithms are naturally terminated by a mechanism called Russian roulette, where low throughput paths have more chance to be terminated than high throughput paths. This allows for a variable depth in the scene which often increases performance. In some cases however, such as in the case of GPU ray tracers it is often preferable to know beforehand how much depth we will need rather than relying on this randomized style of termination. Furthermore if we can assign a fixed depth to all parts of the image, we can more easily pre-allocate memory and possibly apply other optimizations.

In practice this amounts to finding parts of the image that still have a lot of throughput after a certain depth. In figure 4.7 we have such an example. In (a) we see the parallel coordinates plot being used to find high throughput paths with a lot of depth. The ceiling lamp seems to require a lot of depth to render. (d) and (e) show still shots of the animation of the these high throughput paths, where we see it interacting with the lamp. When we change the material of the lamp from glass to a diffuse surface (see (b) and (c)) the depth of the rays hitting the lamps is much lower (see (f) and (g)) as well as the throughput ((h) and (i)).

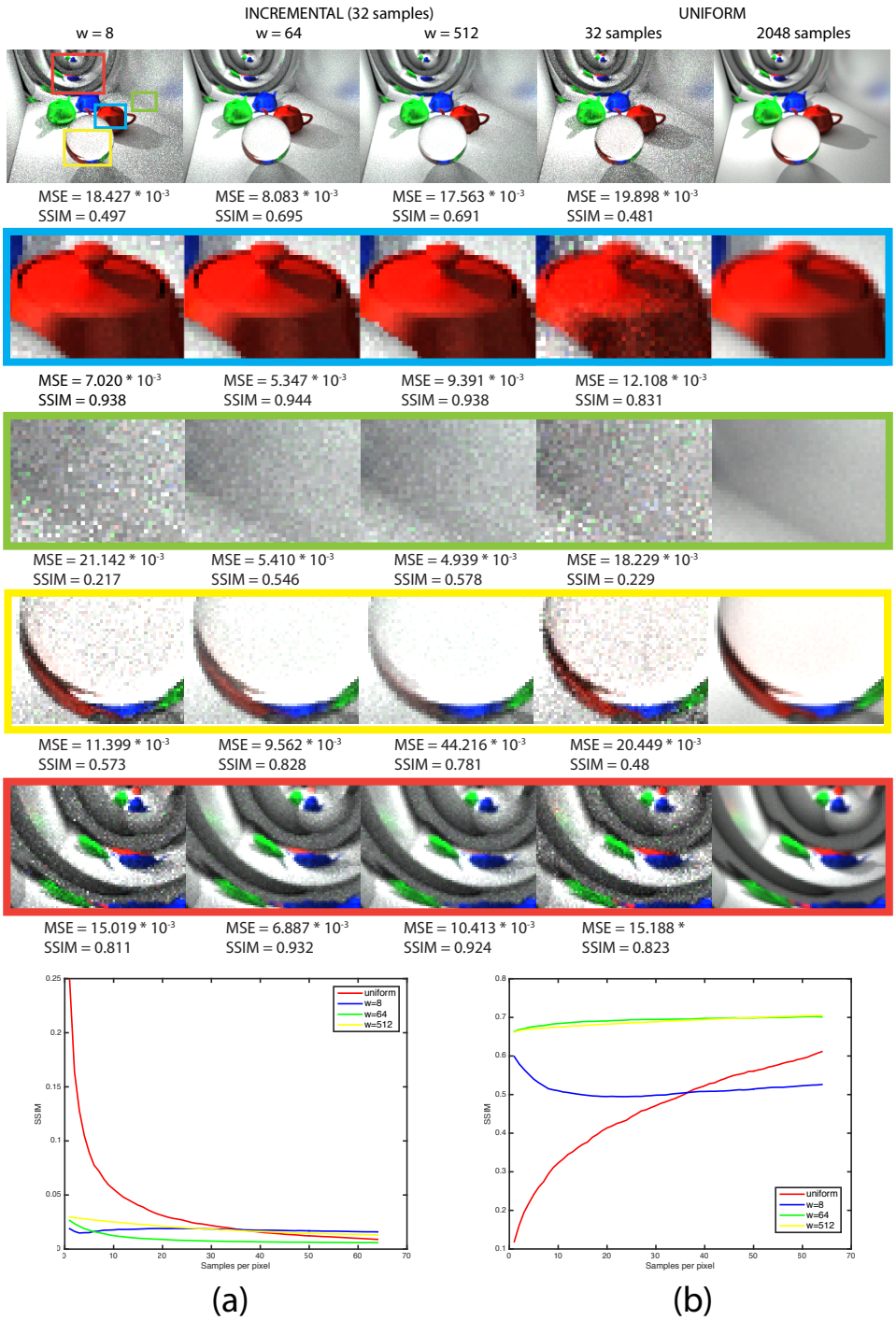


Figure 4.6: A comparison between different parameterizations of the incremental rendering technique with uniformly sampled results. The first three columns show the result of our technique with weights of 8,64 and 512 respectively after 32 samples, whereas the last two columns display the results for a uniformly sampled rendering with 32 and 2048 samples respectively. Two graphs showing how the MSE (a) and SSIM (b) metrics develop according to the number of samples used per pixel are given an the bottom of the figure.

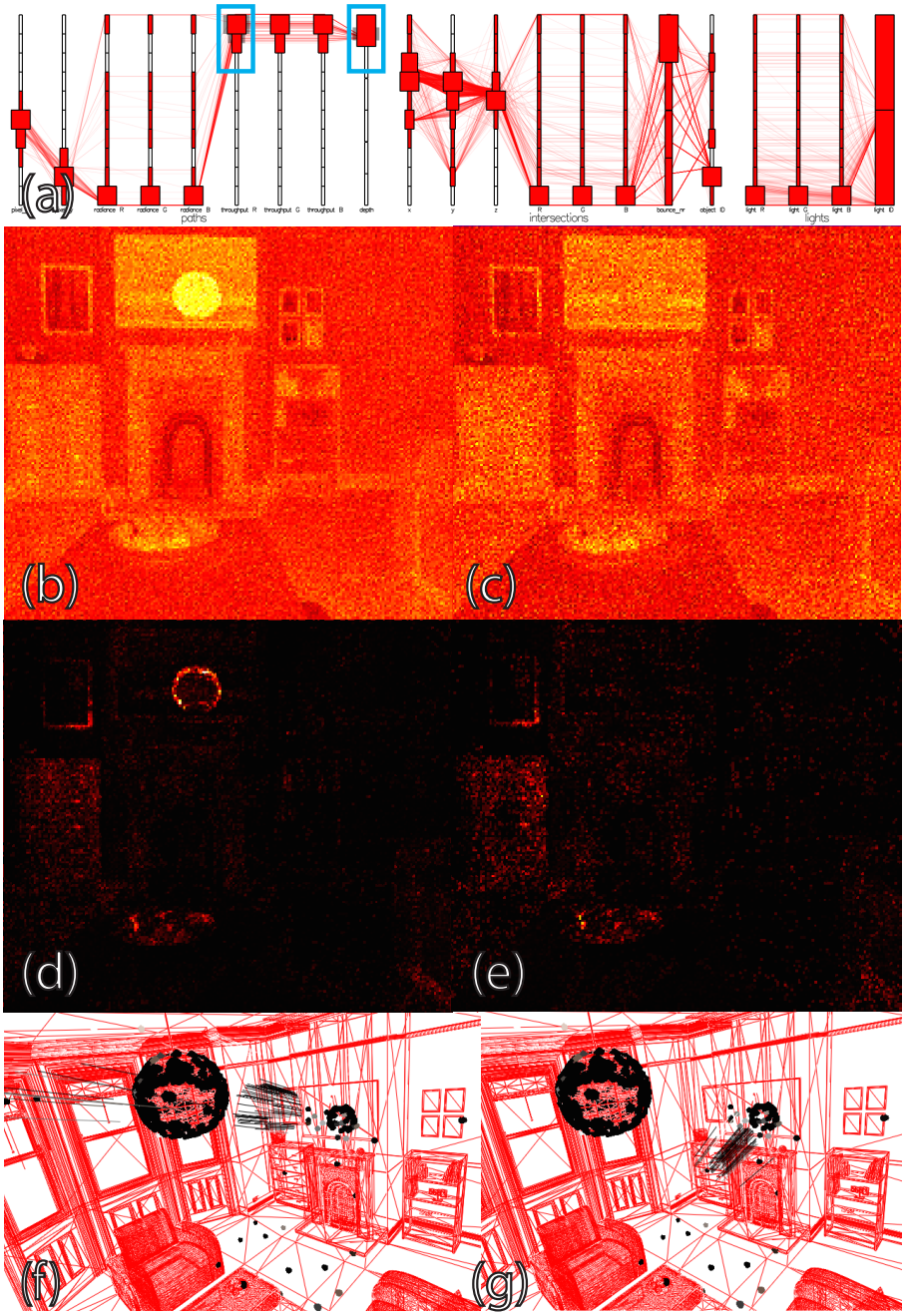


Figure 4.7: Using our heat map visualizations and path animation to find rendering bottlenecks. (a) shows how the parallel coordinates plot was used to brush the data about light paths that have a considerable amount of throughput and depth. (f) and (g) shows the animation of these paths. (b) and (c) shows the average depth of the scene with a glass lamp and a diffuse lamp, respectively. (d) and (e) similarly shows the throughput of these two scenes.

4.7 Firefly Detection

In the previous section we found that the material properties of the lamp shade caused a rendering bottleneck when using a glass material. In the next scenario we show how we can find a different kind of rendering issue, called fireflies. Fireflies are caused by objects that, when hit by a light path, have a very small probability of sampling a large amount of energy from a light source. Because the inverse of this probability is used in Monte Carlo as a product to the energy received, this often results in a very bright pixels. Because these effects are so bright, often 20-100 times brighter than other paths, they are very hard to get rid off. Sometimes the cause of these effects are obvious, large glass or metallic surfaces with high specular coefficients for example, but sometimes the cause is more subtle and may not even be visible from the rendered image.

In figure 4.8 we show the results of such a scenario where we have created a gallery containing various objects such as a teapot, a bunny, a dragon, a buddha and a spherical object. Rather than sampling light paths uniformly, we employ the sampling strategy that prefers high energy paths previously discussed in the main section. These high energy paths are possible candidates for fireflies. After gathering enough of these samples we can observe how these paths interact with the objects in the scene; these objects are likely to be responsible for causing the fireflies. (a) shows the parallel coordinates plot selection for high energy paths. (b) shows the selection in the image space, note how many of the firefly pixels are selected. (e)-(g) shows the animation of the light paths of the fireflies. (e) shows the first bounce, where paths primarily hit the blob on the left and its pedestal (see green marking). In (f) and (g) we see the paths also interacting with the buddha statue in the back (see blue marking). The full animation can also be seen in the video accompanying this paper. By changing the material properties of the spherical object, its pedestal and the buddha we greatly reduce the number of fireflies, as seen in (d), when compared to (c) where we have used the original materials.

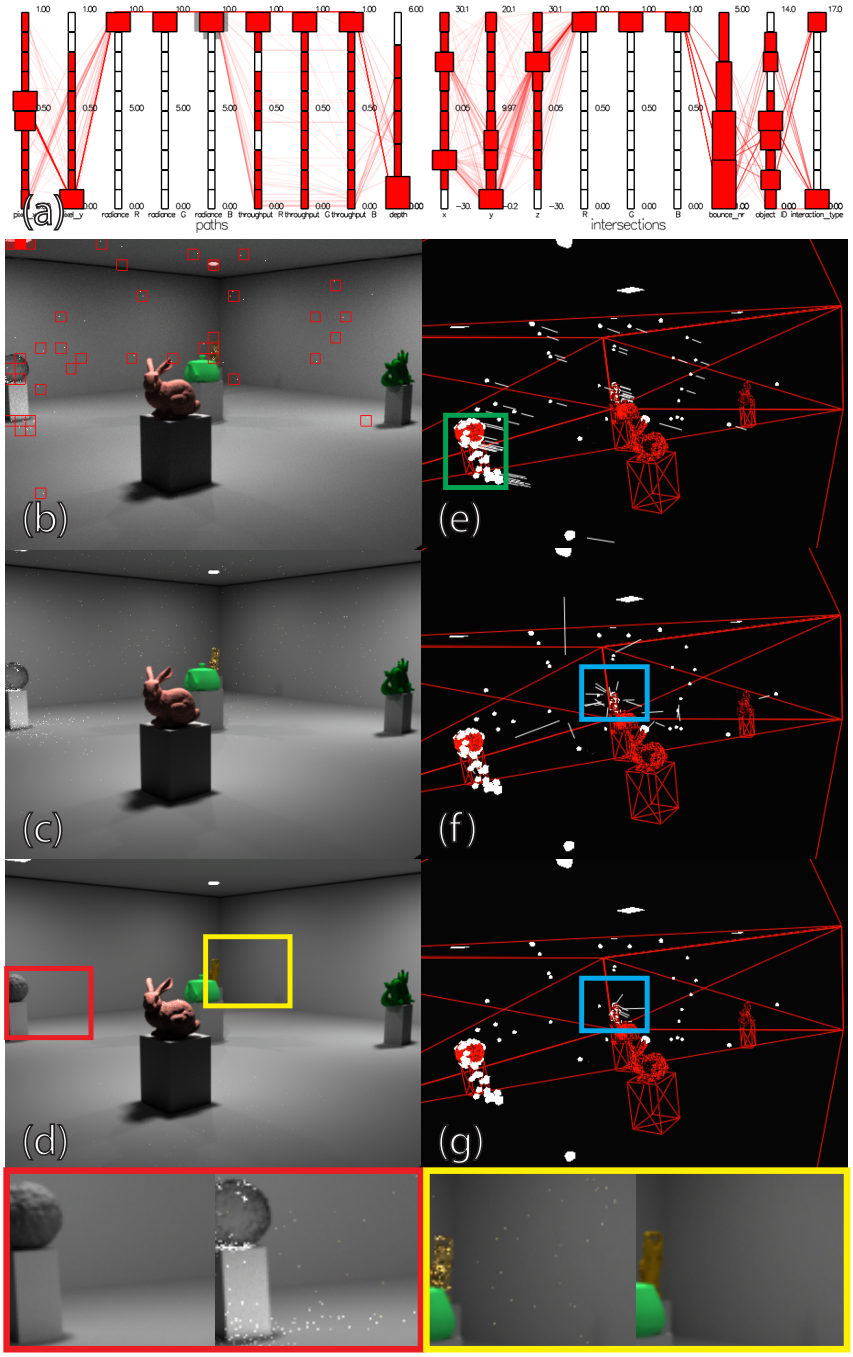


Figure 4.8: Collecting and animating high-energy paths to determine possible causes of fireflies. (a) shows the result of brushing light paths that are very bright. (b) shows the selection distribution in the image view of these paths. (f)-$(h$ shows the animation of these paths (c) and (d) show the rendered result using 1024 samples per pixel with and without the changes in material, with two color-coded insets.

Chapter 5

Conclusion and Future Work

In this work we have shown how information visualization techniques can be of added value to users involved with light transport. Engineers may discover how light is distributed in a scene and how it gets there. Artists may obtain suitable results more effectively by using our user-guided rendering technique so that they may may select parts of the image according to its features. Furthermore significant efficiency gains may be obtained when rendering small differences in scenes using our incremental rendering technique, where we utilize information about radiance and light paths to improve the sampling strategy.

There are various limitations to the current work, some of which may constitute interesting future challenges. We have tried to show how rendering may benefit from information visualization techniques and have purposefully tried to maintain the most general version of each visualization technique, with only a few specific optimizations. This was done to retain a wide applicability. Nevertheless creating more customized visualizations may prove useful for different applications.

There is also plenty of room for completely new visualization techniques. Techniques that show information about light transport at a more abstract level might be useful. A possible direction in this research would be to add visualization on interactions between objects to the scene. What are the relationships between two objects in terms of energy exchange and how is this affected by changing for example the material of these objects? Although these changes may become apparent in our visualization tool, a more direct visualization of these relations might prove useful.

The user-guided rendering showed how convenient it can be to interact with the rendering process through interaction with the visualization tools. An interesting extension to this idea would be to include further rendering manipulation constructs. For example, if it were possible to change material PDFs by brushing light paths, we could increase convergence of the scene even further.

The incremental rendering technique is an especially powerful instance of user-guided rendering, and clearly shows the merit of exploiting the data associated with light paths. We therefore feel that it warrants more research into its effectiveness in a wider range of scenarios, its limitations, the influence of the different parameters and how it compares to other adaptive rendering techniques. Furthermore, it might be possible to deduce the optimal swap chain weight from the given scene parameters.

Bibliography

[1] M.W. Jones B. Spencer and I.S Lim. A visualization tool used to develop new photon mapping techniques. 2014.

[2] Philippe Bekaert, Mateu Sbert, and John Halton. Accelerating path tracing by re-using paths. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 125–134. Eurographics Association, 2002.

[3] Andreas Buja, John Alan McDonald, John Michalak, and Werner Stuetzle. Interactive data visualization using focusing and linking. In *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on*, pages 156–163. IEEE, 1991.

[4] William S Cleveland and Robert McGill. The many faces of a scatterplot. *Journal of the American Statistical Association*, 79(388):807–822, 1984.

[5] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 137–145. ACM, 1984.

[6] Frédo Durand, Nicolas Holzschuch, Cyril Soler, Eric Chan, and François X Sillion. A frequency analysis of light transport. *ACM Transactions on Graphics (TOG)*, 24(3): 1115–1126, 2005.

[7] Christiaan Gribble, Jeremy Fisher, Daniel Eby, Ed Quigley, and Gideon Ludwig. Ray tracing visualization toolkit. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 71–78. ACM, 2012.

[8] Alfred Inselberg and Bernard Dimsdale. Parallel coordinates. In *Human-Machine Interactive Systems*, pages 199–233. Springer, 1991.

[9] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. AK Peters, Ltd., 2001.

[10] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[11] William B Kerr and Fabio Pellacini. Toward evaluating lighting design interface paradigms for novice users. In *ACM Transactions on Graphics (TOG)*, volume 28, page 26. ACM, 2009.

[12] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics 93)*, pages 145–153, 1993.

[13] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. *Visualization and Computer Graphics, IEEE Transactions on*, 17(8):1048–1059, 2011.

[14] Hristo Lesev and Alexander Penev. A framework for visual dynamic analysis of ray tracing algorithms. *Cybernetics and Information Technologies*, 14(2):38–49, 2014.

[15] Stephen M Longshaw, Martin J Turner, and W Terry Hewitt. Interactive grid based binning for information visualization. In *TPCG*, pages 35–42. Citeseer, 2008.

[16] Soham Uday Mehta, Brandon Wang, Ravi Ramamoorthi, and Fredo Durand. Axis-aligned filtering for interactive physically-based diffuse indirect lighting. *ACM Transactions on Graphics (TOG)*, 32(4):96, 2013.

[17] Kenneth Moreland. Diverging color maps for scientific visualization. In *Advances in Visual Computing*, pages 92–103. Springer, 2009.

[18] Diego Nehab, Pedro V Sander, Jason Lawrence, Natalya Tatarchuk, and John R Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware*, volume 41, pages 61–62, 2007.

[19] Man-Suk Oh and James O Berger. Adaptive importance sampling in monte carlo integration. *Journal of Statistical Computation and Simulation*, 41(3-4):143–168, 1992.

[20] James Painter and Kenneth Sloan. *Antialiased ray tracing by adaptive progressive refinement*, volume 23. ACM, 1989.

[21] Tim Reiner, Anton Kaplanyan, Marcel Reinhard, and Carsten Dachsbacher. Selective inspection and interactive visualization of light transport in virtual scenes. In *Computer Graphics Forum*, volume 31, pages 711–718. Wiley Online Library, 2012.

[22] Gernot Schaufler. Exploiting frame-to-frame coherence in a virtual reality system. In *Virtual Reality Annual International Symposium, 1996., Proceedings of the IEEE 1996*, pages 95–102. IEEE, 1996.

[23] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 45–50. Eurographics Association, 2007.

[24] Thorsten-Walther Schmidt, Jan Novak, Johannes Meng, Anton S Kaplanyan, Tim Reiner, Derek Nowrouzezahrai, and Carsten Dachsbacher. Path-space manipulation of physically-based light transport. *ACM Transactions On Graphics (TOG)*, 32(4): 129, 2013.

[25] Harri Siirtola. Direct manipulation of parallel coordinates. In *Information Visualization, 2000. Proceedings. IEEE International Conference on*, pages 373–378. IEEE, 2000.

[26] Kesar Singh and Minge Xie. Bootstrap: a statistical method. *Unpublished manuscript, Rutgers University, USA. Retrieved from http://www. stat. rutgers. edu/home/mxie/RCPapers/bootstrap. pdf*, 2008.

[27] Cyril Soler, Kartic Subr, Frédo Durand, Nicolas Holzschuch, and François Sillion. Fourier depth of field. *ACM Transactions on Graphics (TOG)*, 28(2):18, 2009.

[28] Eric Veach and Leonidas J Guibas. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76. ACM Press/Addison-Wesley Publishing Co., 1997.

[29] Ingo Wald, Timothy J Purcell, Jörg Schmittler, Carsten Benthin, and Philipp Slusallek. Realtime ray tracing and its use for interactive global illumination. *Eurographics State of the Art Reports*, 1(3):5, 2003.

[30] Sven Woop, Louis Feng, Ingo Wald, and Carsten Benthin. Embree ray tracing kernels for cpus and the xeon phi architecture. In *ACM SIGGRAPH 2013 Talks*, page 44. ACM, 2013.

[31] Lehtinen Moon Ramamoorthi Rouselle Sen Soler Zwicker, Jarosz and Yoon. Recent advances in adaptive sampling and reconstruction for monte carlo rendering. 2015.

Appendix 6

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

The Rendering Equation The rendering equation is a mathematical description of the light transport. It defines how the light at a particular position in time and space should be evaluated. A slightly abbreviated version is listed below (omitting wavelength and time).

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega f_r(x,\omega_i,\omega_o) L_i(x,w_i)(\omega_i \cdot \vec{n}) d\omega_i \tag{6.1}$$

BRDF The bi-directional reflectance distribution function defines how light is reflected at a certain position on an opaque surface.

Monte Carlo Monte Carlo sampling is a sampling technique employed by rendering techniques to approximate the rendering equation. It is especially useful in physically based rendering because its performance does not depend on the dimensionality of the integrals.

ROI Region of interest is a distinct region in space where a particularly interesting phenomenon is thought to take place. This is often a reason to investigate it using gathering and visualization techniques such as the placement of a gizmo.

Russian Roulette Russian roulette is used to stochastically terminate paths to prevent them from continuing forever. The probability of a path being terminated depends on its throughput, where paths with lower throughput are more likely to be terminated.

PDF The probability distribution function is used in physically based rendering to approximate the reflectance of materials and distribution of light sources through sampling.

Brushing and Linking A common interacting paradigm employed in visualization software. By combining several views and allowing a interaction, the brushing of data, in one view to be immediately updated in the other views of the data.

Appendix 7

Process & Implementation

In this chapter we will discuss the technical decisions we made regarding our software solution. We will not only discuss the implementation details, but also the reasoning behind it and the various avenues researched that did not make the final implementation, these include libraries, the language choice and visualization techniques proposed.

First and foremost is the choice of programming language: In our case it was immediately clear that performance would be important as rendering techniques are known to be computationally intensive, so if we were to combine them with visualization techniques, which are often resource intensive themselves we would require a language that is indeed very efficient and fast. When thinking of a fast language, C and C++ often comes to mind. C++ has a speed comparable to C, while also offering various higher level data structures and algorithms (STL). Indeed when looking at popular rendering implementations, they are often implemented in C++. A downside of C++ is its lack of portability, where each platform has its own set of compilers with their own specific quirks. This was not thought to be a big issue for this project as it was meant more as a proof of concept, than a piece of production ready software. Another downside of C++ is its low productivity as the programmer has to deal with many low-level memory management issues as well as type safety. The first is still a big problem in C++, but can be alleviated by using new pointer classes introduced in C++11. In order to deal with type safety efficiently in C++, one can use templates. Taking these issues in regard, it still seemed C++ was the right tool for the job, most notably due to its performance and its use in available rendering software.

Having chosen C++, we set out investigating what would be the best choice for window management as well as candidates for 2D and 3D manipulation and display software. A popular combination in these two is the OpenGL-GLUT combination. GLUT is a low-level cross-plaftorm window management solution, and OpenGL a multi-platform API for rendering 2D and 3D graphics. Although other options exist, such as GLFW which also allows for built-in texture loading we thought these additional features were probably not going to be used, so we decided on GLUT. OpenGL was the obvious choice as it seemed the only viable candidate for cross-platform 2D and 3D rendering.

Although in the final version of our solution we incorporated EMBREE[30] as our renderer, we started out using PBRT, which is a very extensive and comprehensive physically based rendering framework containing a variety of rendering techniques. It is also accom-

panied by a book, serving as its documentation, which makes it excellent as a first renderer to work with. It implements various rendering techniques such as path-tracers and photon mapping. We chose path tracer as our rendering technique as it is the most popular technique used and also because photon mapping was already investigated by Reiner et al. [21]. In order to quickly get some results, we used PBRT to generate datasets, files containing all the data about light paths and its intersections. These files would then be read into our visualization software.

There are also various other libraries whose usage we considered, such as VTK, a popular visualization framework. From various sources we learned however that it can often be very costly in terms of time and effort to change it in such a way as to create completely customized visualisations. This is why we decided not to use VTK, but rather create our own visualizaton techniques in OpenGL. As our solution grew more complex it was also contemplated whether we should use QT, which is a popular and extensive application framework. It offers various abstractions of the GUI and includes many UI elements that can be incorporated, which are lacking in OpenGL and GLUT. This would be useful if we were ever to make the software more useable by adding buttons and other UI elements, but for now it seemed adding a framework this large would not be very useful and cost too much overhead for too little a gain. It would be better to implement the few missing features ourselves when we really needed them, or find some stand-alone solutions we could incorporate.

Another library that was tried was the SQLite framework, which is a lighter version of the well-known SQL database management software. It allows for great expressiveness through its SQL queries. This might be useful to allow for example to query certain light paths with various complex properties. Furthermore it would also provide us with various general data structures that could store floats, ints and other data types we would need. We also thought that the SQLite version would also perform well, but after comparison we found that it did slow down our system considerably when iterating over result sets from SQL queries. So this called for a tough decision; revert to our old system and improve it considerably, but have it run fast, or deal with the loss of performance while gaining expressiveness. We choose the former, and by refactoring our solution using templates we created general data structures such as datasets, tables and records. This greatly improved the code and allowed for easy extension of other datatypes. The issue of expressiveness was resolved by adding various filtering classes, that are able to filter datasets according to some specific data dimension and some operator. More complex group-based filtering operators were not made generic, but rather left as custom filtering operations in parts of the software that so required them.

The idea driving this thesis, was to extend the work done by [21] with general information visualization techniques. A popular information visualization technique often used is the parallel coordinates plot. Because our data was high-dimensional the use of the parallel coordinates plot was especially useful. Further on the scene view that would display intersection points and animate paths, was added. A simple heat map that displayed the selection of paths in the image was also added quickly. Soon we discovered that a heat map according to the sampled paths was too sparse to accurately display data such as throughput, so we implemented various distributions that would always be updated as the rendering process

progressed. This would allow for greater detail in the heat maps.

We also made the visualization more focused on comparative visualizations. For this we designed and developed the membership view. This was intended as an extension to the parallel coordinates plot with which the user could quickly see the greatest differences between two datasets. By letting the user choose various data dimensions to compare, it would sum over all the bins a row was part of and determine how unique this bin was. The sum of these bins would then be the uniqueness value. The uniqueness values would then be displayed as a sort of bar-chart which could be brushed by the user. Unfortunately we could not really find a specific application for this visualization technique, and we also found that often the same could be achieved by simply interacting with the parallel coordinates plot.

As our solution matured, its limitations became more and more apparent. Especially the reliance on PBRT and its static data exchange was a problem. The data files were huge, as a 256 x 256 image with a 500 samples would already generate almost 3 GB worth of data in binary format. Furthermore, although PBRT was a nice introduction to the rendering techniques, it was more geared towards academic use and was therefore not very optimized for performance. At the time, EMBREE [30] was a renderer that was fairly new and gaining considerable attention from the graphics community due to its high performance gains made possible by various Intel-specific optimizations.We also wanted it to be dynamic, so we modified it so that it would return data about light paths in real-time. The visualisation software would then decide which data to actually use for the visualization by way of sampling, keeping those paths that fits the data in the best possible way.