Huber Loss Reconstruction in Gradient-Domain Path Tracing

Henrik Dahlberg

School of Science

London 19.12.2018

Supervisor

Jaakko Lehtinen

Advisor

Markus Kettunen

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Author Henrik Dahlberg

Title Huber Loss Reconstruction in Gradient-Domain Path Tracing

Supervisor Jaakko Lehtinen

Advisor Markus Kettunen

Date 19.12.2018 **Number of pages** 108+21 **Language** English

Abstract
The focus of this thesis is to improve aspects related to the computational synthesis of photo-realistic images. Physically accurate images are generated by simulating the transportation of light between an observer and the light sources in a virtual environment. Path tracing is an algorithm that uses Monte Carlo methods to solve problems in the domain of light transport simulation, generating images by sampling light paths through the virtual scene.

In this thesis we focus on the recently introduced *gradient-domain path tracing* algorithm. In addition to estimating the ordinary primal image, gradient-domain light transport algorithms also sample the horizontal and vertical gradients and solve a screened Poisson problem to reconstruct the final image. Using L_2 loss for reconstruction produces an unbiased final image, but the results can often be visually unpleasing due to its sensitivity to extreme-value outliers in the sampled primal and gradient images. L_1 loss can be used to suppress this sensitivity at the cost of introducing bias.

We investigate the use of the *Huber loss* function in the reconstruction step of the gradient-domain path tracing algorithm. We show that using the Huber loss function for the gradient in the Poisson solver with a good choice of cut-off parameter δ can result in reduced sensitivity to outliers and consequently lower relative mean squared error than L_1 or L_2 when compared to ground-truth images.

The main contribution of this thesis is a predictive multiplicative model for the cut-off parameter δ. The model takes as input pixel statistics which can be computed on-line during sampling and predicts reconstruction parameters that on average outperforms reconstruction using L_1 and L_2.

Keywords computer graphics, rendering, path tracing, gradient-domain

Preface

First of all, I would like to express my deepest appreciation and sincerest thanks to my supervisor and advisor for their guidance and encouragement during this thesis; Jaakko Lehtinen, who invited me to conduct my work as an exchange student in a great team of rendering researchers, offering me the best support and supervision I could ask for; and Markus Kettunen, who continuously has devoted his time and made himself available around the clock, often at a moment's notice to answer my uneducated questions by delivering explanations and derivations and running render tasks in the computer cluster.

I want to thank Miika Aittala for fruitful discussions in the early stages of the project around the reconstruction problem and the solver implementation.

My thanks also go to Aalto University for their financial support and for allowing the use of the Triton supercomputer.

Finally, I would like to express my heart-felt gratitude for my friends and loved-ones, and most of all for my brother Christian, my sister Liv, my mother Vivian and my father Urban and thank them for their unwavering love and continuous support.

London, 19.12.2018

Henrik Dahlberg

Contents

List of Figures

1 Introduction

1.1 Background

Rendering is a sub-field of computer graphics concerned with synthesizing images of virtual scenes. Depending on the area of application, different methods with varying degrees of physical correctness and computational intensity are employed in order to reach the ultimate goal of visual realism and fidelity in the resulting images. Images are formed using a sensor to measure the interaction of light emitted from light sources with other matter. Images are represented as a regular grid of pixels each with an associated color, and we need to compute these colors.

Computer games, mobile applications and other interactive media typically demand real-time performance at high frame rates, and commonly have to sacrifice some degree of realism as a result of the time limitation when producing images. Applications in less interactive areas such as film, animation and architectural visualization are often less time-constrained while expecting a higher level of realism. Regardless of application domain, the demand for results of increasing quality delivered in the shortest frame of time possible remains a constant motivation behind research in rendering.

1.1.1 Light Transport and Global Illumination

A core part of creating images in a movie production pipeline is to generate the virtual scene. Modellers, texture painters and groom artists generate geometry of objects in the scene and layout artists and animators dictate the movement and placement of said objects. FX artists generate fluid and smoke simulations. Look development artists design material networks and assign shaders to the different objects and virtual characters. Lighting artists design the lighting setup of the shots to achieve the look and feel requested by the director and visual effects supervisor. Light transport algorithms are needed in order to generate the image that results from the virtual scene with camera and light sources that has been set up by the artists - the scene acts as an input to the renderer, and the renderer simulates the resulting light transport and generates an image.

The field of *light transport* studies the interactions of light with objects and materials as it travels from an emitting light source towards an observer in a virtual setting. Rendering algorithms aimed at solving the light transport problem are known as *global illumination* techniques. This collection of algorithms captures many of the important lighting effects needed to produce realistic images, such as shadows and diffuse inter-reflection (Figure 1.1) and volumetric and caustic effects (Figure 1.2).

Historically, artists have had to spend disproportionate amounts of time and effort in devising tricks and workarounds to successfully achieve these effects, since the renderers of days past did not capture realistic light transport automatically. Modern rendering

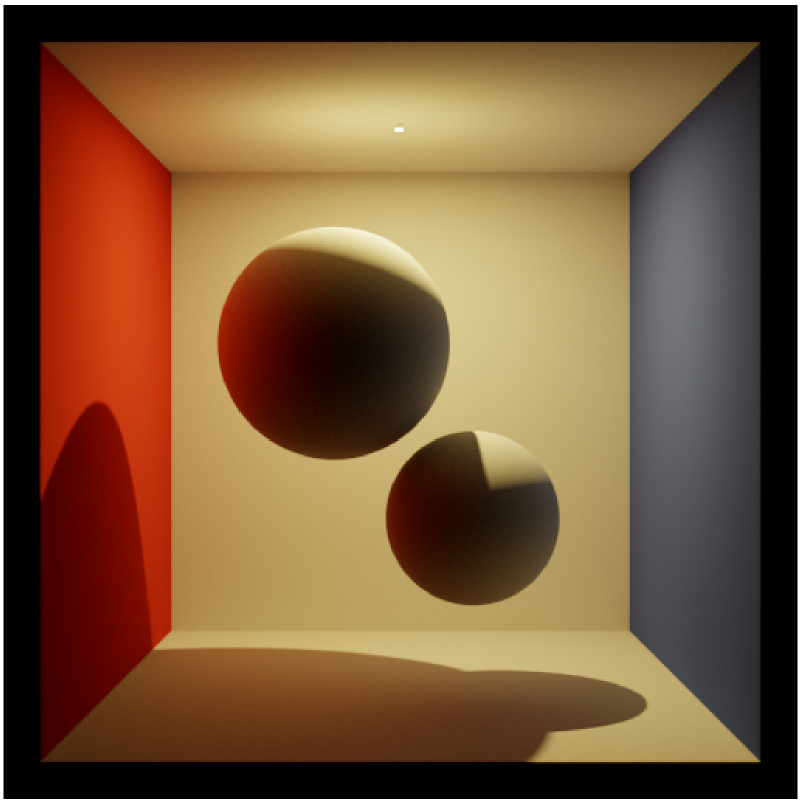


Figure 1.1: Shadows and diffuse inter-reflection. The color of the walls are reflected onto the white diffuse spheres.

techniques are moving towards exclusive usage of models based on physical laws however which capture many of these effects inherently, giving industry professionals more artistic freedom (Christensen, Jarosz, et al. 2016). These physically based methods sample how the light travels through a virtual scene and depend on computationally intensive Monte Carlo simulations. Sampling light paths in a virtual environment using Monte Carlo simulation is known as *path tracing*, and is a core component of physically-based rendering.

The difficulty in solving global illumination lies in the fact that many virtual scenes represent complicated lighting scenarios where out of all the possible ways for the light to propagate, only a small portion actually carries light that contributes to the image. Figure 1.3 illustrates such a scenario. Not seldom a scene can be lit solely from light sources that are obscured by windows or glass objects or seen only indirectly via mirrors or other metallic or glossy materials, which means that the contributing light paths often are very narrow and their contribution to the image difficult to sample.

Physically-based methods have only in recent years been made available for feasible use in production environments thanks to the steady increase in computational resources and algorithm development over the last decades. There is currently a wealth of different rendering algorithms, but despite being capable of solving many problems in the field and seeing widespread use in industry, no method stands out as being capable of solving all problems at higher efficiency than any other method. It is therefore important to invent new algorithms and frameworks as well as improving and extending existing ones in

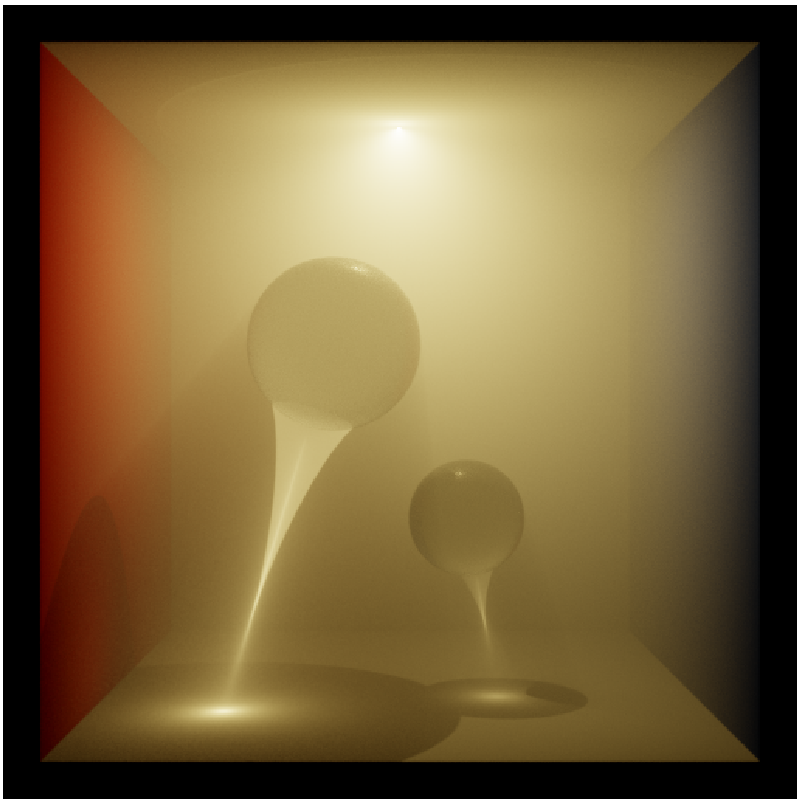


Figure 1.2: Volumetric and caustic effects. The volumetric medium and glass spheres scatter and bend the light rays.

order to produce images faster and simulate more complex lighting scenarios.

1.1.2 Gradient-Domain Rendering

In this work we focus our efforts on an aspect of a recently introduced family of global illumination methods, namely *gradient-domain light transport* algorithms. In addition to sampling the image itself, henceforth referred to as the *primal image*, these algorithms obtain an estimate of the gradient of the image directly during sampling. The rationale behind this approach comes from the observation that natural images tend to be piecewise smooth (Gunturk and X. Li 2012). It is well known that derivatives help in the reconstruction of smooth functions; hence, by estimating how pixel intensities change from pixel to pixel by measuring finite differences, we hope to produce high-quality images faster than the direct approach. Estimates of the changes in pixel intensities are generated by sampling regular base paths, shifting them deterministically and taking finite differences, illustrated in Figure 1.4. The details of how paths are sampled and offset will be discussed in Chapters 2 and 3.

Normally, the pixel intensities I of the final image are estimated directly. Instead of doing only this, we also estimate the horizontal and vertical gradients dI/dx and dI/dy, and combine these estimates to recover a better estimate of I. It turns out that the procedure of combining these estimates involves integrating the gradient estimates by

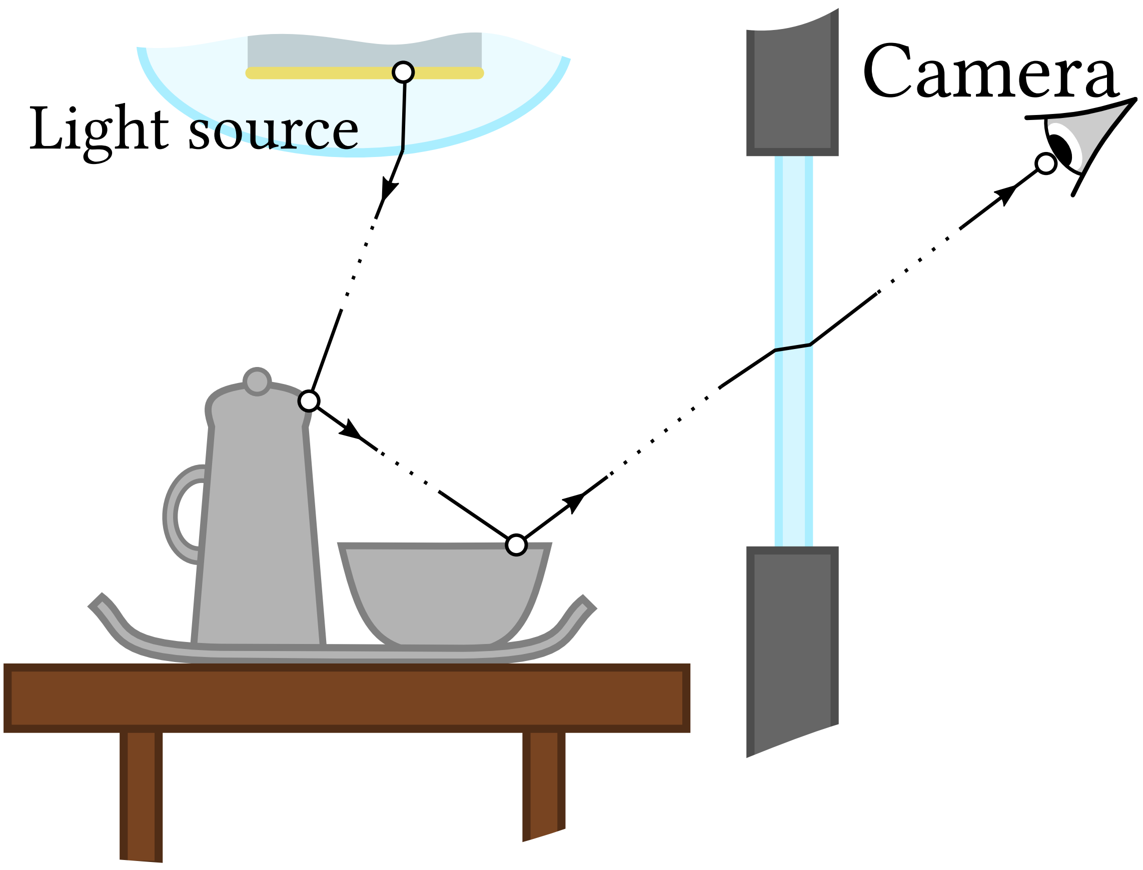


Figure 1.3: A difficult lighting scenario that is representative of common production settings. The camera is obscured by a glass window, and is facing a table with glossy and metallic objects that exhibit a large portion of specular scattering properties. The light source illuminating the scene is covered by glass. In order to produce an image, the light has to be transmitted through the glass cover, scattered from the complicated objects and then transmitted through a window before reaching the camera. Light paths similar to this are difficult to sample for rendering algorithms.

solving a screened Poisson partial differential equation. If we denote p as the estimate of the image intensities I and g as the estimate of the image gradients (dI/dx, dI/dy), finding the solution to the screened Poisson equation and thereby recovering the sought after final image estimate can be formulated as the following optimization problem

$$I = \operatorname*{argmin}_{\phi} L_p(\alpha(\phi - p)) + L_g(\nabla\phi - g). \tag{1.1}$$

Here, α is a scale parameter and L_p and L_g are loss functions. The choice of the loss functions influence the characteristics of the final image I; the image I is selected as the candidate image ϕ that minimizes the error metric related to the sampled p and g in the right hand side of (1.1). Finding the final image using this procedure is henceforth referred to as the *reconstruction*, and our main goal is to improve the reconstruction step so that an image of a given quality can be produced with less computational effort.

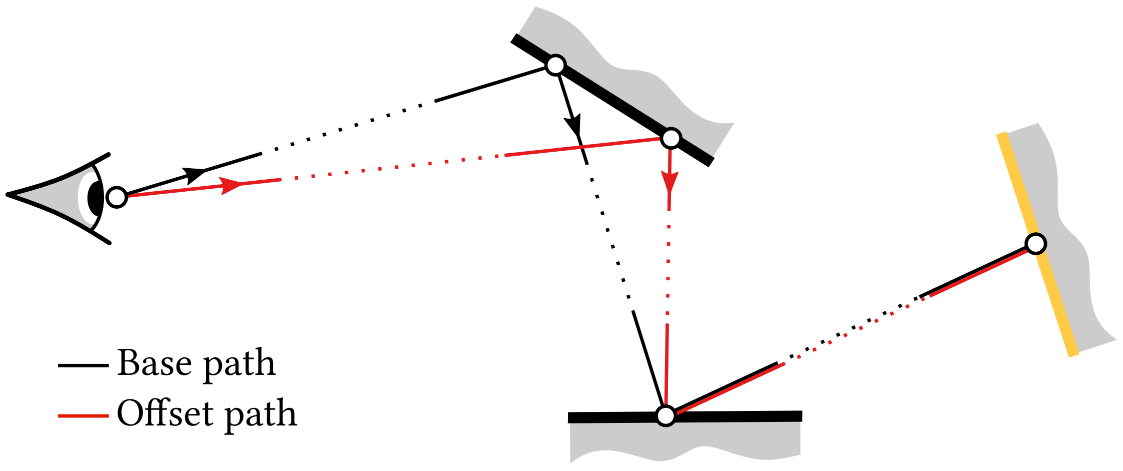


Figure 1.4: A gradient estimate sample is formed by sampling a base path, offsetting the base path and taking the finite difference. The base path is sampled using a Monte Carlo estimator such as path tracing and the base path is then offset using a deterministic shift mapping.

1.1.3 Choice of Reconstruction Loss

The optimization of the reconstruction is driven by measuring the loss of the candidate image compared to the measured primal and gradient images. One loss function L_p is used to measure the candidate image's goodness of fit to the measured primal image, and another loss function L_g to the measured gradient image. Current implementations typically use L_2 loss for the primal part and either L_1 loss or L_2 loss for the gradient part. Using the L_2 loss as error metric for the gradient part produces an unbiased image, but this choice is sensitive to extreme-value outliers in the gradient image pixels caused by the intrinsic variance in the sampling process. This sensitivity manifests itself as undesired dipole-shaped artifacts in the resulting image. Choosing L_1 loss for the gradient part is effective in alleviating this sensitivity issue since it gives less punishment to outliers, but this comes at the cost of introducing statistical bias in the reconstructed image, visible sometimes as darkening. These drawbacks are especially noticeable for low sample counts, one such case being shown in Figure **??**.

1.1.4 Main Goal of the Thesis

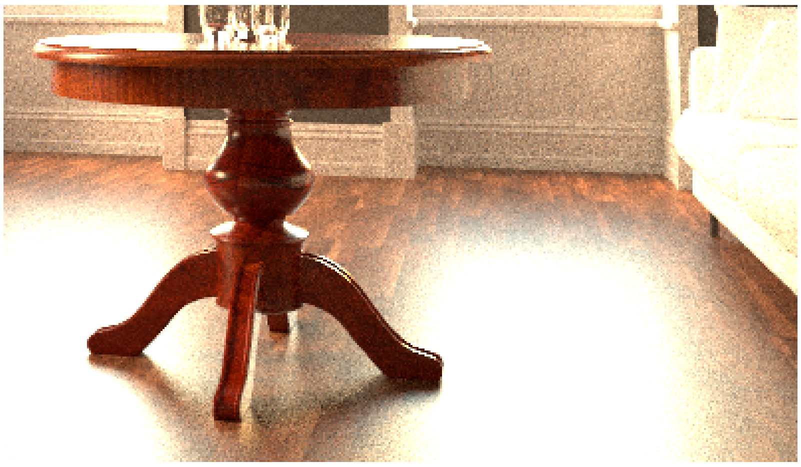
We investigate the use of *Huber loss* as error metric for the gradient part when reconstructing the final image in gradient-domain path tracing. Huber loss combines L_1 loss and L_2 loss by smoothly joining a parabolic function with a linear function at a position specified by a cut-off parameter. We denote this parameter δ. The shape of the Huber loss function varies depending on the choice of this parameter as illustrated in Figure 1.7.

The value chosen for the δ-parameter during Poisson reconstruction influences the resulting final reconstructed image from the optimization procedure and it is therefore important to choose δ wisely. The influence of the δ-parameter on the Huber loss function is illustrated in Figure ??. Larger values of δ bring the characteristics of the Huber loss function closer to those of the L_2 loss, while a value of δ approaching zero suppresses the gradient's importance in the optimization, often leading to poor performance. Somewhere in between the case of a too small or too large value for δ there is an optimal value δ^* that leads to the best reconstruction for a particular scenario. This optimal value varies between different reconstruction cases and it is in general not possible to know what this value is without having a reference render at hand to compare the reconstructed image against.

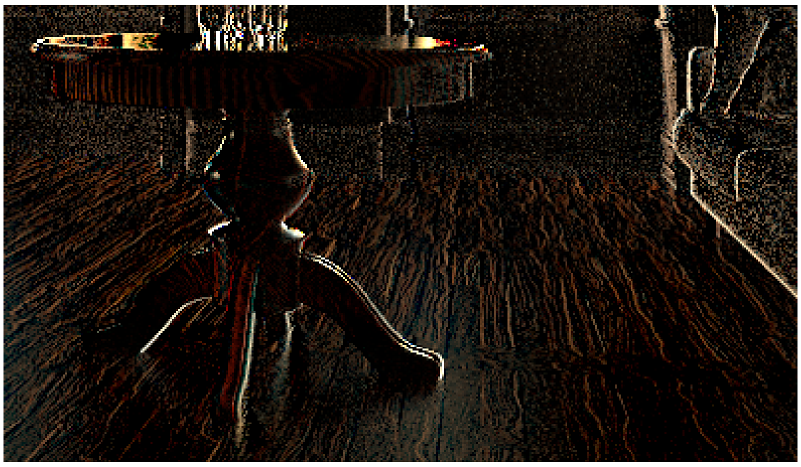
This thesis aims to develop a method for choosing the δ-parameter used in Huber loss reconstruction that produces final images of quality as close as possible to the quality of the final images produced when performing reconstruction using the optimal value δ^*. Our main contribution is a multiplicative model for the δ-parameter, capable of predicting a desirable value of δ to be used in the reconstruction. The model parameters are trained using statistics computed on-line in the gradient-domain path tracing algorithm. We show that reconstruction using Huber loss with δ produced by our model outperforms the use of L_1 loss and L_2 loss in a wide range of scenarios. Additionally, we show that reconstruction using Huber loss introduces considerably less bias in the final image compared to L_1 loss.

1.2 Thesis Overview

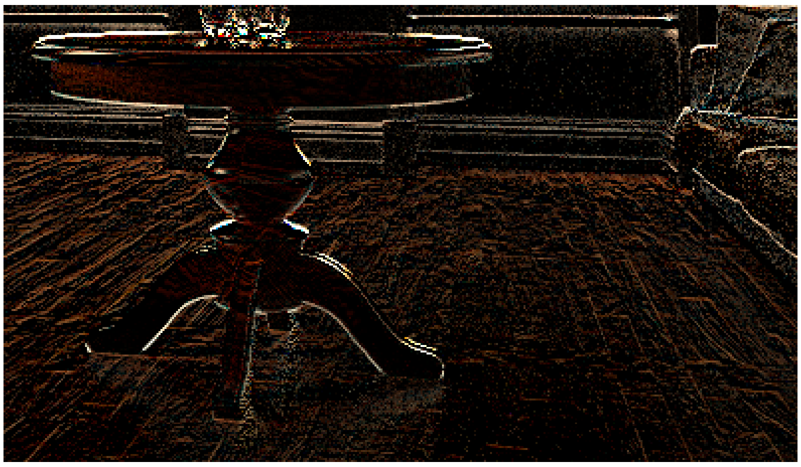
For the reader who is not familiar with the field of light transport simulation, Chapter 2 gives an introduction to the fundamentals of light transport theory and defines the problem to be solved in order to produce photo-realistic images. The mathematical tools and algorithmic frameworks used to solve the light transport problem is also introduced. We present the area of gradient-domain light transport in Chapter 3, giving an account of why and when these methods can solve the light transport problem at a higher efficiency than traditional methods and an overview of algorithms utilizing gradient sampling and integration. The reader who is well-versed in the field of computer graphics applied to photo-realistic imaging may comfortably skip ahead to Chapter 4, where we describe the methods employed in this thesis to predict values for the Huber loss parameter δ. The results of our work are given and discussed in Chapter 5. We reflect upon the implications of our results and findings in Chapter 6, where we also discuss potential venues of future work.



(a) Regular pixel colors.

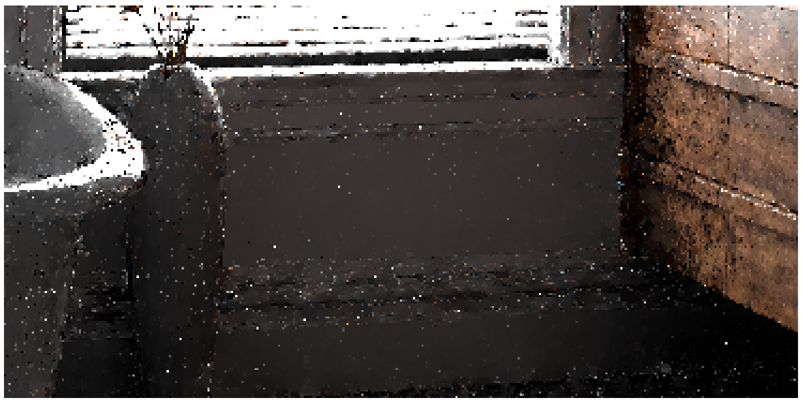


(b) Horizontal finite differences.

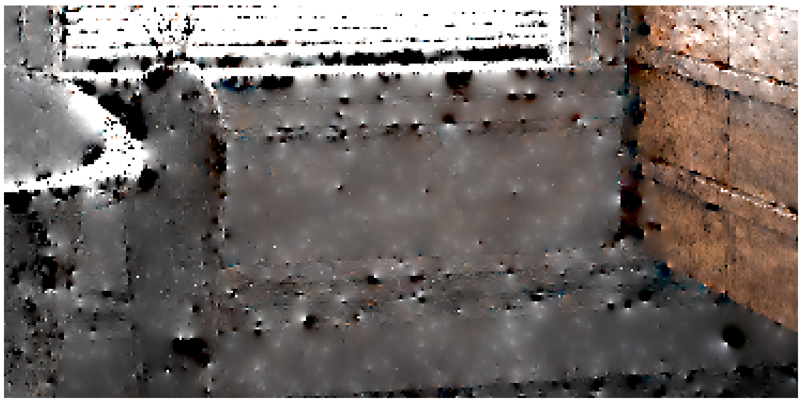


(c) Vertical finite differences.

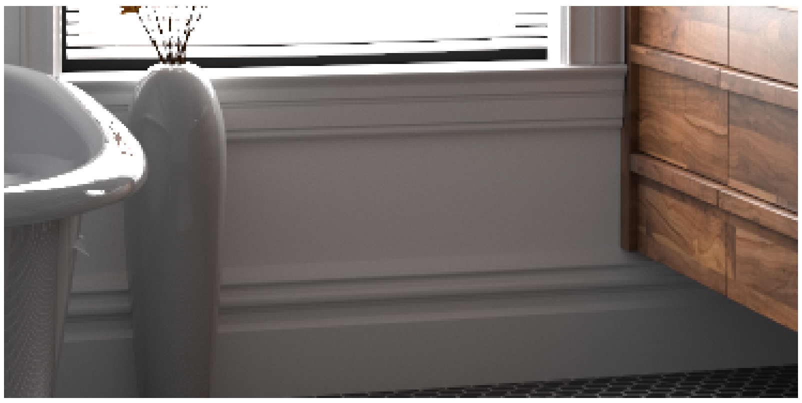
Figure 1.5: Buffers generated through gradient-domain rendering. The horizontal **(b)** and vertical **(c)** finite difference images give information about how the regular pixel colors **(a)** vary across the image plane.



(a) L_1 reconstruction darkens the image.



(b) L_2 reconstruction introduces dipole artifacts.



(c) Reference image.

Figure 1.6: Drawbacks of current reconstruction methods. L_1 reconstruction generates a smooth image that is biased to be too dark. The brightness of the image generated through L_2 reconstruction is closer to the reference image, but instead introduces undesirable dipole shaped artifacts.

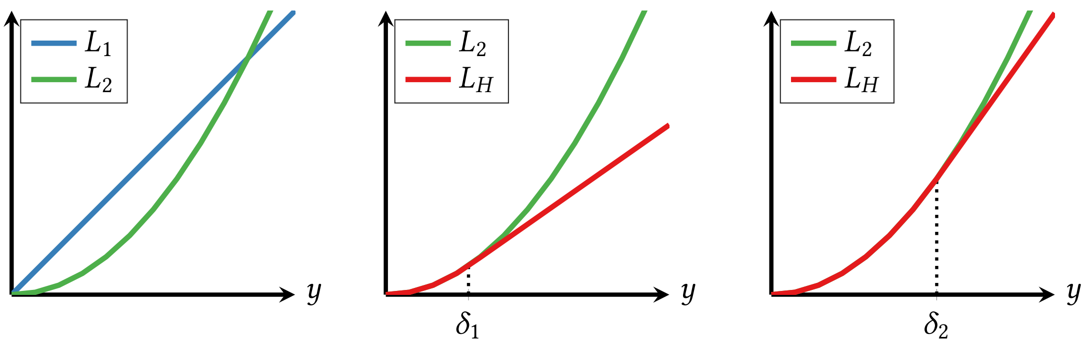


Figure 1.7: Illustration of L_1, L_2 and Huber loss at different values for the cut-off parameters. The value chosen for δ influences the loss a candidate is given. The smaller the value the more candidates will be treated as outliers and be given less consideration in the optimization procedure.

2 Light Transport Theory

The synthesis of photo-realistic images through computer simulation, known as *rendering*, has its foundations in light transport theory. The main concern is to generate images in various applications that contain effects that would be difficult, prohibitively expensive or impossible to achieve with other methods. With a description of the scene environment and its virtual camera viewpoint, geometric objects and the scattering properties of materials and other entities such as fluids and smoke, light transport algorithms simulate the physics of this environment that are relevant to generate accurate images.

There are multiple areas of application that benefit from advances in light transport simulation. In areas such as product and architectural design, accurate light transport simulation functions as a pre-visualization tool that can predict how an object looks in a variety of different lighting conditions prior to manufacturing. In applications such as film production where the generation of images with realistic lighting is the main focus, there has historically been a heavy artistic burden to carefully position the light emitters in the virtual scene environment. Lights had to be positioned to account for refractions and reflections in the scene, since it was difficult to simulate indirect lighting correctly before global illumination algorithms became computationally viable. The study of light transport has therefore been and continues to be important in order to inherently and correctly capture the visual aspects of digital objects.

Correctly simulating the virtual environment as in the real world would require a physically accurate description of the scene and a complete description of the physical laws that govern the world. To the best of our knowledge this is impossible, since the physical laws of our world is not completely known and the digital storage and computational power required would be astronomical. A complete simulation is not required to capture the visuals of a scene however, and we can make many simplifications to both the scene geometry and the physical laws at virtually no cost.

Light transport simulation for the described purposes has its roots in neutron transport theory, radiative transfer theory and the geometric optics model (Chandrasekhar 1950). Geometric optics is a simplified idealistic model of light interaction but accurate to a high degree and adequate for realistic image synthesis. Under the adoption of this model, phenomena such as diffraction, iridescence and fluorescence that can be described by a more complex quantum or wave optics model are not captured automatically (Grimaldi 1665; Guilbault 1990; Kinoshita, Yoshioka, and Miyazaki 2008). These effects are often left to be added through various hacks and manual model augmentations. To simulate iridescence for instance, it is common to manually tint the highlights of a material in various ways or add the effect using digital compositing (Porter and Duff 1984).

Before we define the fundamental physical quantity for rendering, we here introduce the concept of solid angle. Solid angle, denoted σ, is a measure of the amount of the field of view from some particular point $\mathbf{p}$ that a given object covers, and is defined as a

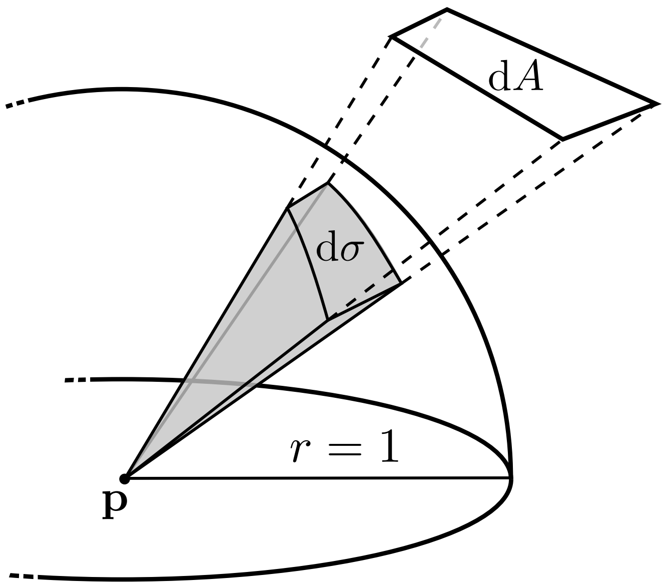


Figure 2.1: Illustration of the solid angle. An element with differential area dA subtends a differential solid angle dσ on a unit sphere centered at the apex point $\mathbf{p}$.

pyramid or a cone. The point $\mathbf{p}$ from which the object is viewed is called the apex of the solid angle. The size of the solid angle is determined by the area subtended by the object on a unit sphere centered at $\mathbf{p}$. The differential solid angle dσ is thus the differential cone or pyramid subtended by a differential surface dA from the apex point $\mathbf{p}$. An illustration of a the differential solid angle is shown in Figure 2.1. The unit for solid angle is steradian [sr].

The physical quantity of interest in the presentations to come is *radiance*. Radiance is defined as the amount of energy arriving per second from a direction ω at a differential surface area dA at a surface point $\mathbf{x}$ aligned perpenticularly to ω, and is written as $L(\mathbf{x}, \omega)$. Expressed differently, it is defined as the flux per unit solid angle and per unit projected area. Its unit is [W/m^{2}sr]. We will also use the incident and exitant radiance $L_i(\mathbf{x}, \omega)$ and $L_o(\mathbf{x}, \omega)$, the former being the radiance arriving at a surface point $\mathbf{x}$ from direction ω and the latter being the radiance leaving it. We refer the interested reader to the work by Arvo (1995) for a more rigorous treatment of radiance and radiometry in the context of light transport.

2.1 Scene Representation

Computer generated images are created by capturing a digital environment using a virtual camera analogous to how physical cameras capture real environments, the renderer essentially simulates the photography process in a digital setting. The virtual environment is commonly refered to as the *scene* and a typical scene with its basic components is illustrated in Figure 2.2. In addition to a sensor in the form of a virtual camera, its position and viewpoint, the scene description contains information about scene geometry such

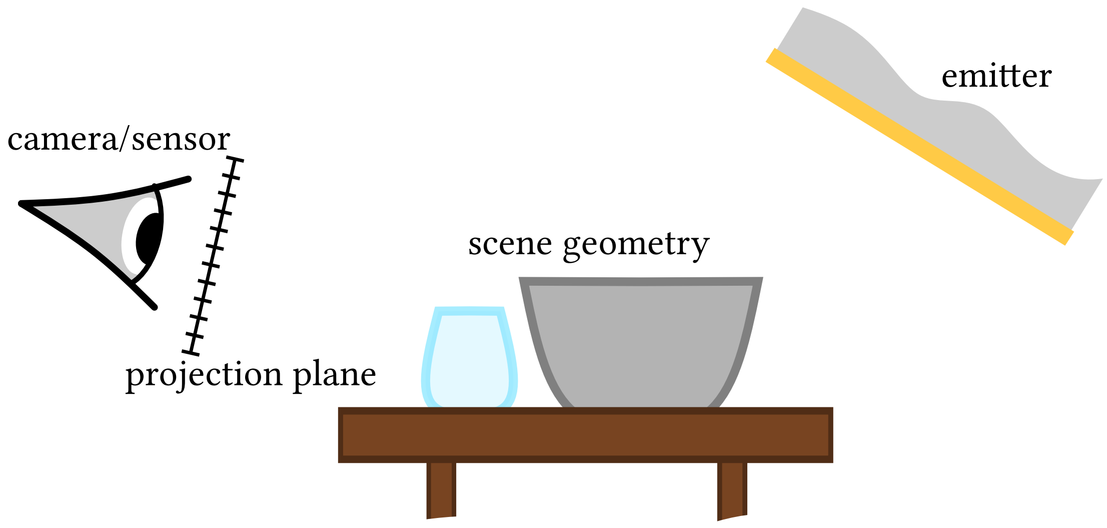


Figure 2.2: Illustration of a virtual scene in light transport simulation. Light from the emitters interact with the scene geometry and the radiance arriving into the camera aperture is measured in the camera sensor to generate an image.

as floors, walls, furniture, foliage or any other objects that constitutes the scene and the materials of the objects. It also describes which objects emit light, such as a lamp or a distant sun-like object, and the properties of the light they emit. The renderer is fed the scene description and simulates the transportation of light from the emitters in the scene, computing the interaction of light with the scene geometry on its way to the camera.

For the intents and purposes of this work, a thin lens, pinhole camera model is sufficient, and this thesis will therefore not delve into the area of modelling lenses or other internal camera components. The interested reader is referred to the works by Kolb, D. Mitchell, and Hanrahan (1995) and Wu et al. (2011).

2.1.1 Geometry

Geometry in virtual environments need to be defined in such a way that the renderer can determine the location at which light interacts with the object. Analytic shapes such as spheres, ellipsoids, cones and splines are often used for simple objects and to model thin details such as hair. Objects that can not be described analytically, which are the most common, are described using polygon meshes. These meshes, such as the quadrilateral mesh in Figure 2.3, are point clouds of vertices where each vertex defines a coordinate in space.

The shape of the geometry meshes are often modified further through the use of normal and displacement mapping to achieve higher detail. Normal mapping modifies the geometry by changing the direction of the surface normal of the triangles in the mesh, and displacement mapping changes the position of the triangle surfaces. Both these techniques change how the light interacts with the object, at what position and

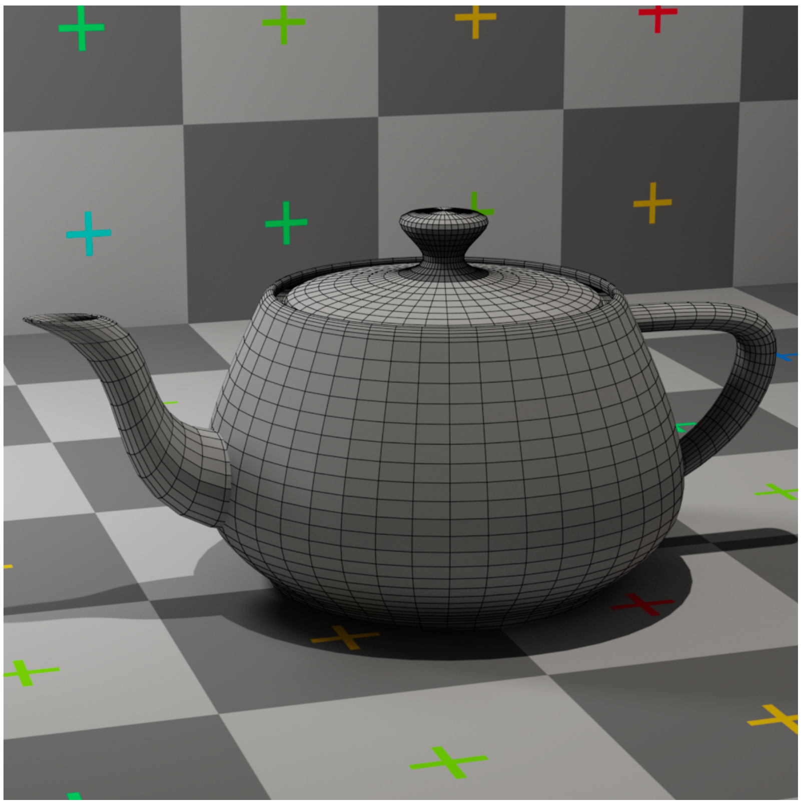


Figure 2.3: Polygon mesh of the Utah teapot. The vertices of the model are collected into quadrilateral polygons. Triangle meshes are also common.

angle the light strikes the object's surface, ultimately changing how it looks. Texture mapping is used to change various material parameters across the surfaces, such as the albedo color, the amount of normal and displacement mapping or the surface roughness, and to control the variation of other material parameters such as roughness.

2.1.2 Material Appearance

The appearance of a virtual object is decided entirely by how it is defined to interact with light through its scattering properties. Materials can exhibit diffuse, specular, glossy and transmissive properties to name a few.

The scattering properties of a material is described by a function known as the *bidirectional scattering distribution function*, abbreviated as the BSDF (Nicodemus 1965). It is defined as ratio of the differential reflected radiance leaving the surface point $\mathbf{x}$ in a direction ω_o and the radiance arriving at the surface point from direction ω_i through a differential solid angle $\mathrm{d}\sigma(\omega_i)$. Explicitly, it is defined as

$$f_r(\mathbf{x}, \omega_i \to \omega_o) = \frac{\mathrm{d}L_o(\mathbf{x}, \omega_o)}{L_i(\mathbf{x}, \omega_i) \cos \vartheta_{\mathbf{x},\omega_i} \mathrm{d}\sigma(\omega_i)} \tag{2.1}$$

where $\vartheta_{\mathbf{x},\omega_i}$ is the angle between the incident direction ω_i and the surface normal at the point $\mathbf{x}$ and $\mathrm{d}\sigma(\omega_i)$ is the solid angle measure.

The BSDF describes how light arriving at a surface point should be scattered from the material and together with surface position, normals and textures captures entirely the properties that dictate its visual appearance.

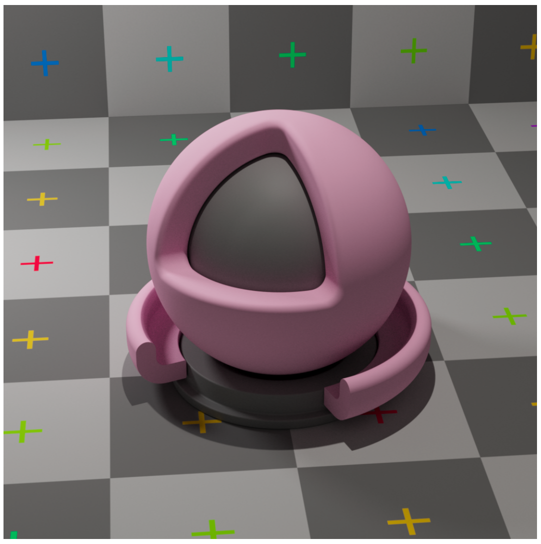
2.1.3 Participating Media

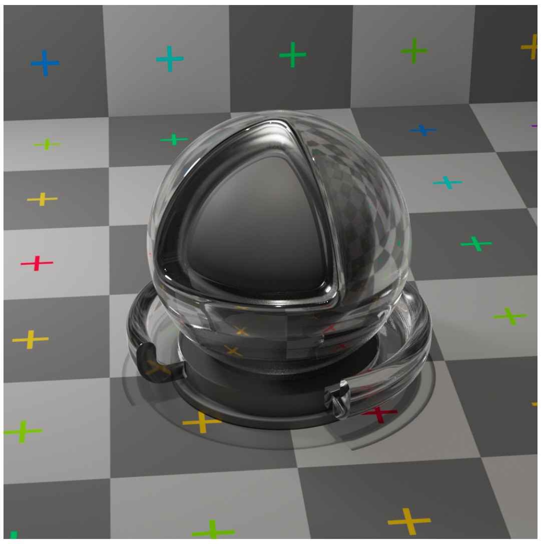
So far we have only described how surface components of the scene is modeled and how light interacts with these entities, but volumetric effects such as smoke and fire are naturally a central component of producing accurate images. Such effects are commonly called *participating media* in the context of rendering. Simulations of these components in light transport for realistic image synthesis are based on the work in radiative transfer by Chandrasekhar (1950). Volumetric rendering is also of key importance when simulating light interactions with skin and fluids.

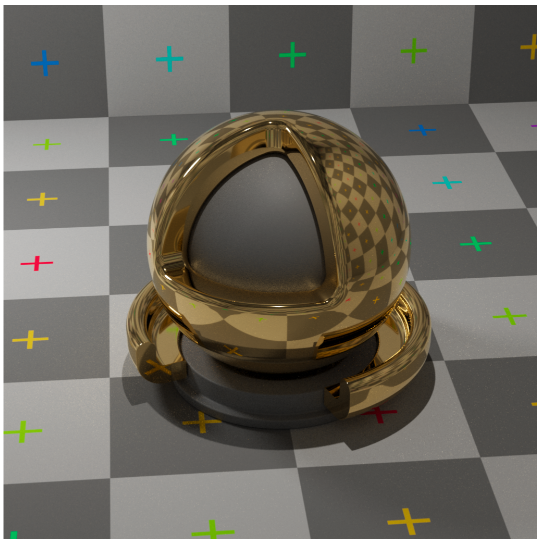
We will not cover the interaction of light with volumetric components here and instead refer to work by Jarosz (2008) for a thorough overview of participating media rendering.

2.2 The Rendering Equation

To render images we first need a mathematical framework through which we describe the transportation of light from one place to another in the scene. We here give a cursory overview of the rendering equation, the measurement equation and its path integral formulation that constitute the foundation of recent advances in the field of light transport. For a rigorous derivation of this machinery we direct the reader to the seminal work by Veach (1998). The interested reader may also review extensions of the path integral formulation by Jakob (2013) to the context of volumetric rendering.


(a) Diffuse.


(b) Dielectric.


(c) Smooth conductor.

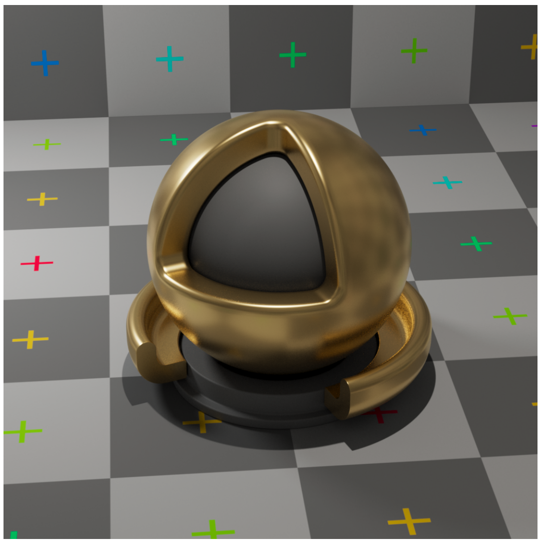

(d) Rough conductor.

Figure 2.4: Examples of various material properties. **(a)** shows a perfectly diffuse object, scattering light rays equally in all directions. The dielectric material in **(b)** acts like glass, refracting and transmitting rays through the object. **(c)** and **(d)** show metallic conductor materials of different roughness, giving the objects varying levels of glossiness.

Let $\mathcal{M}$ denote the union of all the surfaces in the scene. $\mathcal{M}$ can be thought of as the set of all points that lie on any surface in the scene geometry. Note that while the surfaces of the scene lie in a three dimensional space, $\mathcal{M}$ is a two dimensional manifold. We begin by defining the outgoing radiance at a surface point $\mathbf{x} \in \mathcal{M}$ as the sum of the inherently emitted radiance at the point, and radiance that has been scattered away after arriving at the point from other parts of the scene.

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + L_s(\mathbf{x}, \omega_o) \tag{2.2}$$

The emitted radiance $L_e(\mathbf{x}, \omega_o)$ describes the emissive properties at the surface point $\mathbf{x}$ and allows for the modelling of emitters such as lamps and other light sources. If the object at $\mathbf{x}$ is not emissive, this term is zero.

The scattered radiance $L_s(\mathbf{x}, \omega_o)$ at a point $\mathbf{x}$ in the outgoing direction ω_o is found by integrating the differential outgoing radiance over all incident directions on the hemisphere $\mathcal{S}^2$

$$L_s(\mathbf{x}, \omega_o) = \int_{\mathcal{S}^2} \mathrm{d}L_o(\mathbf{x}, \omega_o). \tag{2.3}$$

From one point of view, the emitted radiance $L_e(\mathbf{x}, \omega_o)$ can be seen as the integration constant to the above integral. Note that the differential outgoing radiance in the integrand above is the numerator of Equation 2.1 for the BSDF f_r. After rearranging and substituting we can write the scattered radiance as

$$L_s(\mathbf{x}, \omega_o) = \int_{\mathcal{S}^2} L_i(\mathbf{x}, \omega_i) f_r(\mathbf{x}, \omega_i \to \omega_o) \cos \vartheta_{\mathbf{x},\omega_i} \mathrm{d}\sigma(\omega_i). \tag{2.4}$$

The integral is computed over all incoming directions ω_i on the hemisphere at the surface point $\mathbf{x}$ and gives the outgoing scattered radiance in the outgoing direction ω_o.

We now have an expression for the scattered part of the outgoing radiance at a surface point, and by combining Equation 2.2 with Equation 2.4 we get

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\mathcal{S}^2} L_i(\mathbf{x}, \omega_i) f_r(\mathbf{x}, \omega_i \to \omega_o) \cos \vartheta_{\mathbf{x},\omega_i} \mathrm{d}\sigma(\omega_i). \tag{2.5}$$

Note that the unknown radiance quantity appears on both sides. To know the outgoing radiance L_o at the surface point $\mathbf{x}$ we must know the incoming radiance L_i which is the outgoing radiance from some other surface point in the scene, hence this is an integral equation. This balance equation for radiance was introduced by Kajiya (1986) and is known as the *rendering equation*. The light transport problem is defined as the problem of solving this equation.

2.2.1 The Measurement Equation

Having described how light interacts with the scene, we also need to describe how to make measurements of radiance. We do this by defining and modeling a virtual camera which just like physical cameras has an aperture through which light can enter, and a sensor that can measure radiance. Indeed, images are formed by measuring the sensor response in cameras to the light that enters its aperture and translating the measurements into pixel values.

The camera aperture is modeled as a surface in the scene and is therefore part of $\mathcal{M}$. To generate an image of M pixels, every pixel $j \in \{1, 2, \ldots, M\}$ is associated with a sensor element of the camera sensor, each capable of making measurements on radiance. The camera sensor is modeled as a regular grid of smaller sensor elements each corresponding to a pixel.

The response of the camera sensor with respect to the position at which the sensor is struck by the incident light and the incident direction is described through the *importance function* $W_e(\mathbf{x}, \omega)$. The importance models the ratio of sensor response per unit of power arriving along a ray of light at position $\mathbf{x}$ on the camera aperture from direction ω.

Each pixel value in the image is a measurement of incident radiance by its associated sensor element. The measurement I_j of the j-th pixel is computed through the *measurement equation*

$$I_j = \int_{\mathcal{M}} \int_{\mathcal{S}^2} h_j(\mathbf{x}, \omega) W_e(\mathbf{x}, \omega) L_i(\mathbf{x}, \omega) \cos \vartheta_{\mathbf{x},\omega} \mathrm{d}\sigma(\omega) \mathrm{d}A(\mathbf{x}) \tag{2.6}$$

where $h_j(\mathbf{x}, \omega)$ is the pixel filter. Where the importance function W_e dictates the response of the entire sensor grid to incoming radiance, the pixel filter h_j dictates how much of this response is attributed to the j-th sensor element. A simple example of a pixel filter is the box filter, which has a value of one if the ray of light entering through the aperture at point $\mathbf{x}$ from direction ω strikes the sensor element associated with the pixel, and zero if it does not. Other common filters in image processing are the Gaussian, Lanczos and Mitchell-Netravali filters (D. P. Mitchell and Netravali 1988; Turkowski 1990).

Importance Transport

Just as emitters in the scene emit radiance, the sensors of the scene can be viewed as emitting importance. This is a reciprocity property of light transport and is the reason for the e-subscript of the importance function. We will not go into detail about the consequences of reciprocity but we note that it lies at the foundation of certain light transport algorithms such as bidirectional path tracing which simulates two propagation and scattering processes simultaneously; importance transport in addition to regular radiance transport.

2.2.2 Path Integral Formulation

To find an explicit expression for the value of the measurement, we can express the measurement Equation 2.6 as an integral over *path space*. To define path space, we first define a *path*, which can be seen as the route the light propagates along through the scene. A path x of length k is a series of $k + 1$ vertices

$$x = \mathbf{x}_0 \mathbf{x}_1 \ldots \mathbf{x}_k, \tag{2.7}$$

where the vertices $\mathbf{x}_i \in \mathcal{M}$ for all $i \in \{0, 1, \ldots, k\}$. As such, each path x of length k lies in the Cartesian product space $\Omega_k = \mathcal{M}^{k+1}$. This space contains all paths of length k, and path space which is the space of all paths of arbitrary length is defined naturally as the union of these spaces

$$\Omega = \bigcup_{k=1}^{\infty} \Omega_k. \tag{2.8}$$

The derivation of the path integral formulation now comes down to expanding the measurement equation repeatedly and rearranging it into an infinite sum of integrals, one sum for each path length. This sum can be rewritten as a single integral over path space Ω

$$I_j = \int_\Omega h_j(x) f(x) \mathrm{d}\mu(x) \tag{2.9}$$

where $\mathrm{d}\mu(x) = \prod_{i=1}^{k} \mathrm{d}A(\mathbf{x}_i)$ is the area product measure. This is the *path integral formulation* of the measurement equation.

The integrand is the contribution of a path x to the measurement at the j-th pixel; $f(x)$ is known as the *measurement contribution function* and describes the throughput carried by a path x to the entire camera sensor, and $h_j(x)$ is once again the pixel filter. The measurement contribution function $f(x)$ is a product over the vertices in the path x and takes surface emission, visibility and scattering properties through the BSDF into account. The task of rendering algorithms is to estimate this integral by computing values of the measurement contribution function.

It is common to combine the contribution to the camera sensor and the pixel filter to instead describe the contribution of a path to a specific measurement by a sensor element tied to a pixel. The contribution to the measurement for the j-th pixel is then written as $f_j(x) = h_j(x) f(x)$.

2.3 Monte Carlo Simulation

The problem of rendering images has now been formulated. The virtual camera is modeled as an aperture through which light can enter and strike a sensor array. Each sensor element makes measurements of the radiance carried by the incident light to form pixel values for the image through the integral in Equation 2.9.

The remaining portion of this chapter is now dedicated to the computation of this integral. Universal analytic solutions to the integral have yet to be found as the integrand is ridden with discontinuities and other ill-behaved properties caused by complex geometry and scattering functions. We therefore use numerical methods to solve the integral through computational means.

One-dimensional integrals can be computed using various interpolation techniques based on quadrature rules, such as those of the Newton-Cotes family. To compute multi-dimensional integrals, one strategy is to use Fubini's theorem (Fubini 1907) to reformulate the multi-dimensional integral as a repetition of one-dimensional integrals and apply quadrature methods to compute these. Unfortunately, as the number of dimensions of the integral increases, this approach requires the number of function evaluations to grow exponentially, a problem known as the *curse of dimensionality* (Bellman 1957).

Since the integral problem we are interested in is an integration over the infinite-dimensional path space Ω, quadrature based methods are not an option and we therefore resort to computing the integral using Monte Carlo methods which do not suffer from the curse of dimensionality.

2.3.1 Monte Carlo Integration

The use of Monte Carlo methods is becoming the industry standard in any field related to light transport simulation. The work on distributed ray tracing by Cook, Porter, and Carpenter (1984) was the first usage of Monte Carlo sampling in the context of rendering. Kajiya (1986) employed Monte Carlo methods to solve the rendering equation.

The idea behind computing the measurement integral using Monte Carlo integration is to reformulate it as an expected value that can be computed as a Monte Carlo estimate. To this end, we view each path x as a realization of a random variable X and path space Ω as the set of possible outcomes of this random variable. Let $\pi(x)$ denote the *probability density function* (pdf) of X. The integrand in the path integral formulation of the measurement equation can now be rewritten slightly, and we can immediately identify the integral expression for the j-th pixel measurement as an expected value

$$I_j = \int_\Omega f_j(x)\mathrm{d}\mu(x) = \int_\Omega \frac{f_j(x)}{\pi(x)}\pi(x)\mathrm{d}\mu(x) = \mathbf{E}\left[\frac{f_j(X)}{\pi(X)}\right]. \tag{2.10}$$

This means that the pixel value I_j can be seen as an estimator of the expectation above. Let $x_1, \ldots, x_n$ be n independent and identically distributed realizations of X. A Monte Carlo estimate of the pixel value can then be computed as

$$I_j^{(n)} = \frac{1}{n}\sum_{t=1}^{n} \frac{f_j(x_t)}{\pi(x_t)} \tag{2.11}$$

By the law of large numbers we have $\lim_{n\to\infty} I_j^{(n)} = I_j$, and if we can generate paths x_t and compute their throughputs $f_j(x_t)$ and pdfs $\pi(x_t)$ we have a way to compute an estimate of the pixel value I_j. No matter how many samples we add to form the estimate however, for a finite number of samples n the estimate $I_j^{(n)}$ will always deviate from the true value I_j by a certain amount. This deviation, the error, can be estimated by the sample variance of the estimator in Equation 2.11.

$$\mathrm{Var}\left[I_j^{(n)}\right] = \mathrm{Var}\left[\frac{1}{n}\sum_{t=1}^{n}\frac{f_j(X)}{\pi(X)}\right] = \frac{1}{n}\mathrm{Var}\left[\frac{f_j(X)}{\pi(X)}\right] = \frac{1}{n}\sigma_n^2. \tag{2.12}$$

As long as the sequence $\{\sigma_1^2, \sigma_2^2, \sigma_3^2, \ldots\}$ is bounded, this variance decreases asymptotically to zero as we add more samples and we can form an arbitrarily accurate estimate. The rate at which the noise diminishes as more samples are added is dictated by how large the variance of the Monte Carlo estimator is.

In the context of rendering, the error of the Monte Carlo estimator manifests itself as noise in the image, demonstrated in Figure 2.5.

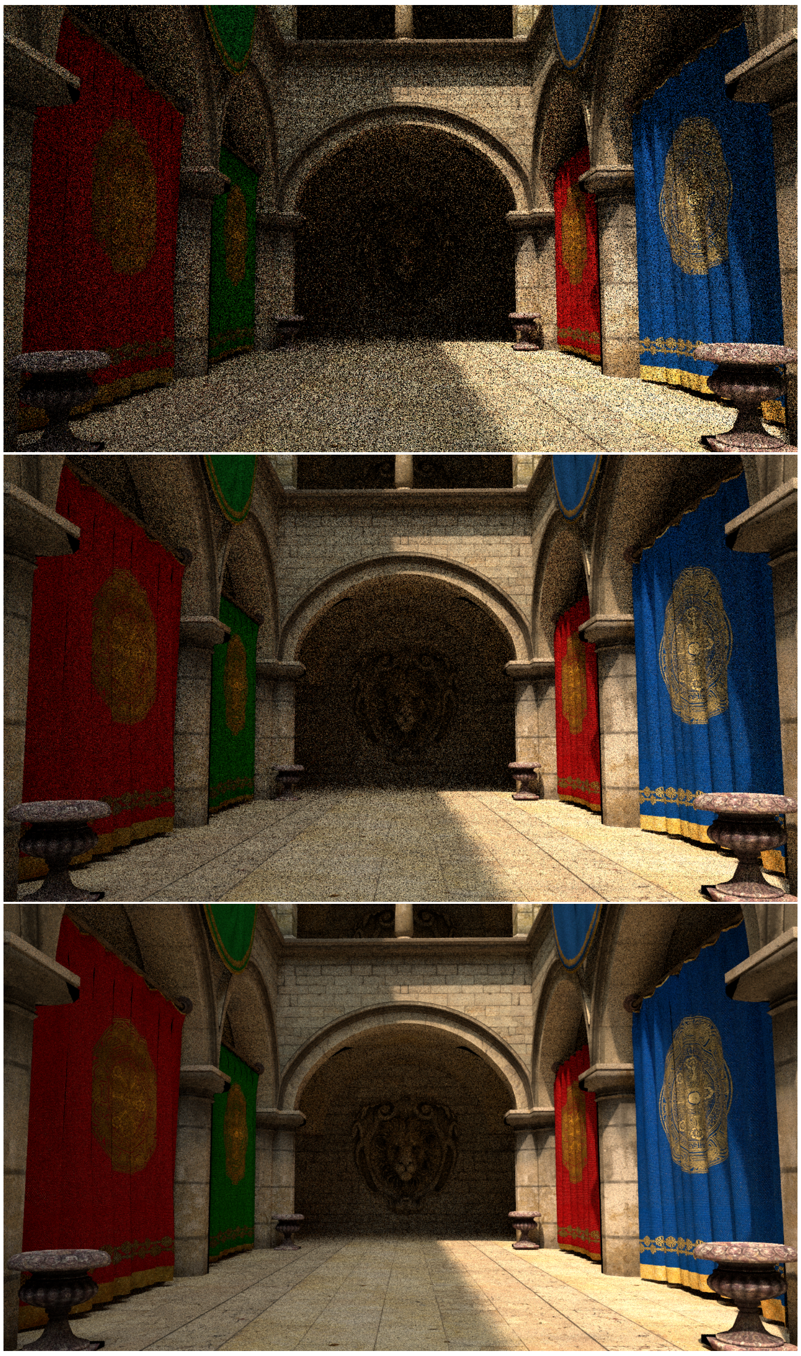


Figure 2.5: Images generated through Monte Carlo integration are noisy since we can never get an infinite number of path samples. The noise is reduced as we add more samples. This is illustrated above where the image becomes cleaner as we move from 4 samples per pixel (top) through 16 samples per pixel (middle) to 64 samples per pixel (bottom).

Variance Reduction

Since the ever present error of a Monte Carlo estimator is tied to its variance, it is important to design the Monte Carlo sampler in a way that minimizes the variance of said estimator. We cannot form an exact estimate, so we want to design the sampling strategy allows our estimate to come sufficiently close to the true value with as few samples as possible. Controlling the sampling strategy in this manner is called *variance reduction.*

From Equation 2.12 we see that if we could sample paths from a probability distribution that was proportional to the measurement contribution function, i.e. $\pi(X) = af_j(X)$ for some positive scalar a, the variance of our estimator would be zero.

$$\frac{1}{n}\mathrm{Var}\left[\frac{f_j(X)}{\pi(X)}\right] = \frac{1}{n}\mathrm{Var}\left[\frac{f_j(X)}{af_j(X)}\right] = \frac{1}{n}\mathrm{Var}\left[\frac{1}{a}\right] = 0 \tag{2.13}$$

Sampling from a distribution proportional to the target function is unfortunately not possible. Any probability distribution must integrate to one over the probability space, therefore we have

$$\int_\Omega \pi(x)\mathrm{d}\mu(x) = a\int_\Omega f_j(x)\mathrm{d}\mu(x) = 1. \tag{2.14}$$

For this to hold we see that the scalar a must be the reciprocal of the integral we were looking for in the first place. This example nevertheless highlights a detail that is important for variance reduction; if we can sample from a distribution that is similar to the target function, we have an opportunity to improve the convergence rate of the estimator even if its variance will never be zero.

This technique of sampling from a distribution that is similar to the function we are interested in is known as *importance sampling*. Even though it is not possible to sample directly from the measurement contribution function f_j, it is a product of many terms and it is often possible to importance sample some of its factors, for instance those related to material BSDFs. Importance sampling is a core part of efficient algorithm design in rendering.

2.3.2 Markov Chain Monte Carlo

We now describe a family of algorithms that are designed to generate samples from probability distributions. The samples generated from these simulations can be used to form Monte Carlo estimates, and are useful for more complex probability distributions when generating samples by using simpler methods such as inversion sampling or rejection sampling is not possible.

Markov chain Monte Carlo (MCMC) methods are a collection of algorithms designed to generate samples from probability distributions by simulating a series of random events known as a *Markov chain* (Gilks, Richardson, and Spiegelhalter 1995). A Markov chain is a stochastic process $X_1, X_2, X_3, \ldots$ with the property that the probability of the next event in the process depends only on the current state

$$Pr(X_{i+1}|X_1, X_2, \ldots, X_i) = Pr(X_{i+1}|X_i). \tag{2.15}$$

The right hand side describes the probability of moving from a state X_i to a state X_{i+1} and it dictates the evolution of the Markov chain. We call this the *transition density* and write it as

$$q(x_i \rightarrow x_{i+1}) = Pr(X_{i+1} = x_{i+1} | X_i = x_i). \tag{2.16}$$

A core concept for MCMC simulation is the *stationary distribution* of the Markov chain. A distribution ϱ is said to be stationary if the detailed balance equation is satisfied.

$$q(x \rightarrow x')\varrho(x) = q(x' \rightarrow x)\varrho(x') \tag{2.17}$$

This means that the distribution from which the Markov chain assumes values has stopped evolving and reached a stationary state. The idea behind MCMC sampling is to construct a Markov chain whose stationary distribution is proportional to some target function we are interested in sampling from. Indeed, if the Markov chain reaches this stationarity and evolves according to the target distribution, then each state x it visits is a realization of the random variable we are trying to simulate and we can register each visited state as a sample. MCMC algorithms are designed in such a way that the transition density q ensures this behaviour of the Markov chain.

Another important concept is that of *ergodicity.* A Markov chain $X_1, X_2, X_3, \ldots$ with stationary distribution ϱ is called *ergodic* if, regardless of the initial distribution, the chain forgets its initial distribution and converges towards ϱ as the chain evolves. This is only true in the limit however, and unless the initial distribution coincides with the target distribution, the samples generated from the chain will never behave exactly as desired given a finite simulation time. This is known as *start-up bias.* It is therefore important to carefully choose the initial distribution to minimize this bias or eliminate it entirely to ensure that the Markov chain generates samples from the desired distribution.

The Metropolis-Hastings Algorithm

There are many MCMC sampling algorithms, one of the most prominent being the *Metropolis-Hastings algorithm* (Hastings 1970; Metropolis et al. 1953). The Metropolis-Hastings algorithm is an MCMC algorithm that aims to generate samples from a target distribution f by generating a Markov chain that has f as stationary distribution. This is achieved by starting from the detailed balance equation (Equation 2.17) and splitting the transition probability q into a proposal and acceptance step

$$q(x \rightarrow x') = r(x \rightarrow x')\alpha(x \rightarrow x'). \tag{2.18}$$

The idea is to sample a candidate state x' from a proposal kernel $r(x \rightarrow x')$ and accepting or rejecting the candidate state with probability $\alpha(x \rightarrow x')$. Either the chain will accept the new state x' or stay in the current state x. Inserting Equation 2.18 into Equation 2.17 and rearranging the terms gives

$$\frac{\alpha(x \rightarrow x')}{\alpha(x' \rightarrow x)} = \frac{r(x' \rightarrow x)\varrho(x')}{r(x \rightarrow x')\varrho(x)}. \tag{2.19}$$

To satisfy this relationship and set f as the targeted stationary distribution, the acceptance probability is defined as

$$\alpha(x \to x') = \min\left(1, \frac{f(x')r(x' \to x)}{f(x)r(x \to x')}\right). \tag{2.20}$$

In its entirety, the Metropolis-Hastings algorithm thus constists of repeatedly sampling a new proposal state from the proposal kernel, computing the acceptance probability and then either accepting or rejecting the new state.

The acceptance probability in the Metropolis-Hastings algorithm is contructed from two fractions that both have a natural role in deciding the transition density so that the resulting Markov chain generates samples from the target distribution f.

- $f(x')/f(x)$ suggests:
  - accept candidate x' if it is more likely under f, i.e. $f(x')/f(x) \geq 1$
  - accept candidate x' with probability $f(x')/f(x) < 1$ if it is less likely under f
- $r(x' \to x)/r(x \to x')$ suggests:
  - the easier it is to reach x' from x, the larger the denominator $r(x' \to x)$ will be and will reduce the acceptance probability.
  - the easier it is to return to x from x', the larger the numerator $r(x \to x')$ will be and will increase the acceptance probability.

The states visited by the resulting Markov chain from the Metropolis-Hastings algorithm are all registered as samples, regardless of if the new proposed states was accepted or rejected, and can later be used to form Monte Carlo estimates.

MCMC algorithms benefit from proposal kernels that approximate the target function well, just as in the case of importance sampling, and the design of good proposal kernels is at the core of designing MCMC algorithms for rendering.

Measurement Integration Using Markov Chains

In the context of rendering and light transport simulation, most interesting is how to compute integrals of the target function using Markov chains. To generate an image, we need to compute measurement integrals for each pixel

$$I_j = \int_\Omega f_j(x)\mathrm{d}\mu(x). \tag{2.21}$$

Instead of driving a Markov Chain with the per-pixel measurement contribution function f_j as stationary distribution for each pixel, MCMC algorithms in rendering target the contribution function to the entire sensor f, so that the Markov chain generates samples that contribute to all measurements of the image simultaneously

$$I_j = \int_\Omega h_j(x)f(x)\mathrm{d}\mu(x). \tag{2.22}$$

Using the Metropolis-Hastings algorithm, we can drive a Markov chain with path space Ω as its state space and the visited states $x_1, x_2, \ldots, x_n$ are paths guaranteed to be distributed proportionally to the target contribution function f. In other words, the probability distribution function of the random variable for each state can be written as a scalar multiple of the target function, $\pi(x) = af(x)$. The estimate of the measurement integral is then

$$I_j \approx \frac{1}{n}\sum_{t=1}^{n}\frac{h_j(x_t)f(x_t)}{\pi(x_t)} = \frac{1}{n}\sum_{t=1}^{n}\frac{h_j(x_t)f(x_t)}{af(x_t)} = \frac{1}{n}\sum_{t=1}^{n}\frac{h_j(x_t)}{a}. \tag{2.23}$$

The remaining task is to compute the proportionality constant

$$a = \int_{\Omega} f(x)\mathrm{d}\mu(x). \tag{2.24}$$

This might seem impossible at first glance, but since this constant is shared among all the M pixels in the image, where M usually is a large number, it is feasible to approximate this constant using a secondary integration technique such as some other Monte Carlo estimator.

2.4 Light Transport Algorithms

When discussing integration of the measurement integral in Section 2.3.1, we noted that if we can generate paths x and compute their probabilities $\pi(x)$, we have a way to estimate the pixel values. The design of light transport algorithm comes down to creating a machinery that samples paths, and it is the implemented sampling scheme that single-handedly determines what the probability distribution functions for the paths are. We are thus entirely free to choose π, the only restriction we have is that, for the law of large numbers to hold, π must be non-zero and positive whenever the measurement contribution function f_j is non-zero and positive.

We also mentioned the concept of variance reduction and came to the conclusion that if the probability density functions of the generated paths are similar to the target function, the variance of the estimator is lower. It is therefore important to design the sampling scheme so that the resulting pdfs π approximate f_j to a high degree, and it is the degree to which different light transport algorithms achieve this that differentiates them.

2.4.1 Random Walks

In general it is not possible to sample directly from the space of light carrying paths and therefore the paths that we want to sample in order to form measurement integral estimates are built successively by using *random walks* through the scene. A random walk is a stochastic process that consists of a succession of random steps in some mathematical space, in our case path space Ω.

A sampler can build paths by generating a random walk that starts either from the sensor or an emitter, and there are techniques that do both simultaneously and then connect the two paths for higher efficiency. As previously we write a path as a series

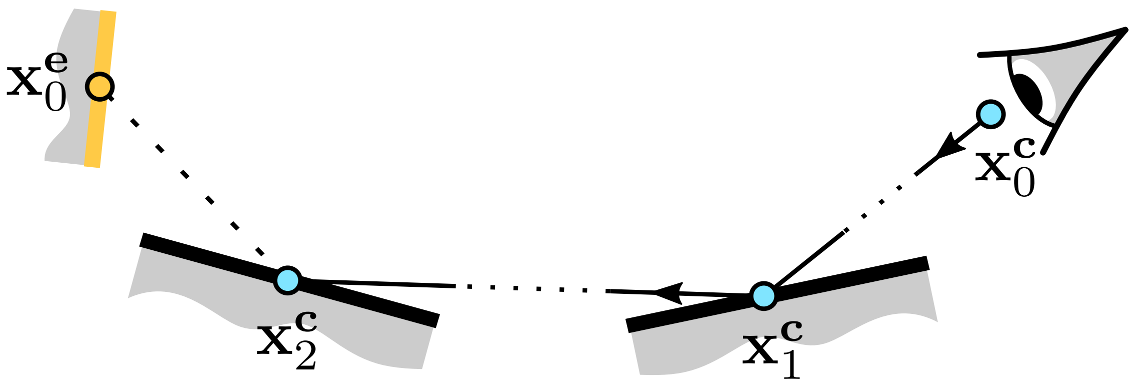

$\mathbf{x}_0^{\mathbf{e}}$
$\mathbf{x}_0^{\mathbf{c}}$
$\mathbf{x}_2^{\mathbf{c}}$
$\mathbf{x}_1^{\mathbf{c}}$

Figure 2.6: Path sampled with random walks originating from both the camera and and the emitter.

of vertices $x = \mathbf{x}_0\mathbf{x}_1 \ldots \mathbf{x}_k$, and we write $\mathbf{x}_i^{\mathbf{c}}$ for vertices belonging to a path sampled from the camera and $\mathbf{x}_i^{\mathbf{e}}$ for vertices belonging to a path sampled from the emitter to distinguish between the two. A full path can be a combination of sub-paths generated through random walks that start from the camera and emitters; for instance, the path $x = \mathbf{x}_0^{\mathbf{c}}\mathbf{x}_1^{\mathbf{c}}\mathbf{x}_2^{\mathbf{c}}\mathbf{x}_0^{\mathbf{e}}$ depicted in Figure 2.6 is a path with three vertices sampled from the camera and one sampled from the emitter.

Propagation and Scattering

Regardless of where the first path vertex was sampled from, generating the rest of the random walk consists of a series of propagation and scattering events until the random walk terminates. Given a path vertex $\mathbf{x}_i$ and outgoing direction ω_i for the ray of light originating at $\mathbf{x}_i$, the next vertex on the path $\mathbf{x}_{i+1}$ is the nearest point in the scene geometry in the ray direction. This point is found through *ray casting*. The ray casting function is defined as

$$\mathbf{r}_{\mathcal{M}}(\mathbf{x}, \omega) = \mathbf{x} + \omega d_{\mathcal{M}}(\mathbf{x}, \omega) \tag{2.25}$$

where $d_{\mathcal{M}}(\mathbf{x}, \omega) = \min\{d \in \mathbb{R} \,|\, \mathbf{x} + \omega d \in \mathcal{M},\ d > 0\}$ returns the distance to the closest point in the geometry in the ray direction. If the ray misses the scene geometry entirely, $\mathbf{r}_{\mathcal{M}}(\mathbf{x}, \omega)$ returns infinity. With this definition we find the next vertex in the random walk as $\mathbf{x}_{i+1} = \mathbf{r}_{\mathcal{M}}(\mathbf{x}_i, \omega_i)$.

When the ray arrives at a new vertex along the path being generated by the random walk, a scattering event occurs and a new outgoing direction is determined. Arriving at a vertex $\mathbf{x}_i$, the next outgoing direction ω_i depends on the material of the geometry at the vertex point, which is described by the bidirectional scattering distribution function. Efficient selection of the outgoing direction thus depends on importance sampling the BSDF $f_r(\mathbf{x}_i, -\omega_{i-1} \rightarrow \omega_i)$, where the current incoming direction ω_{i-1} is the outgoing direction of the previous path vertex $\mathbf{x}_{i-1}$ from which the ray came from. The negation in this incoming direction stems from the fact that the BSDF is defined using directions pointing outward from the surface point. When importance sampling for scattering, it is also important consider the term $\cos\vartheta_{\mathbf{x}_i,\omega_i}$ that affects the throughput.

Termination

The sampling of a path cedes when the random walk process terminates. This can happen when a ray being propagated exits the scene, for instance when the scene depicts and outdoor environment and the ray scatters into the open air towards the sky.

If the scene is completely enclosed however, the ray would propagate and scatter through the environment indefinitely and the random walk would continue forever and the rendering algorithm using the paths to generate an image would be prohibited from doing so. We can solve this by setting a maximum path length, but this has the problem of introducing bias into the measurement contribution of the paths since we no longer have a non-zero probability of sampling the entire path space.

A better way is to terminate the ray probabilistically. At each path vertex $\mathbf{x}_i$, the idea is to select a continuation probability w_i and thus terminate the random walk with probability $1 - w_i$. This way there is still a non-zero probability of sampling all paths, but the random walk will terminate in finite time if w_i is chosen properly. If the random walk continues from vertex $\mathbf{x}_i$, the probability distribution function of the resulting path is multiplied by w_i to ensure that the Monte Carlo estimator remains unbiased. This technique is known as *Russian roulette* (Arvo and Kirk 1990), and normally you start this procedure after a certain number of vertices has been sampled along a path. Russian roulette increases the variance of integral estimator however and one must thereful be careful in choosing the termination probability.

After the random walks terminate, the resulting paths have a measurement contribution throughput that depends on the emission and the materials it has interacted with, and an associated probability distribution function that describes the probability with which the path was sampled. These quantities are then used to form a Monte Carlo sample for the path that will contribute to integral estimates. The probability distribution function of a path x is

$$\pi(x) = \pi(\mathbf{x}_0\mathbf{x}_1 \dots \mathbf{x}_k) = \pi(\mathbf{x}_0) \prod_{i=1}^{k} \pi(\mathbf{x}_i | \mathbf{x}_{i-1}, \mathbf{x}_{i-2}, \dots, \mathbf{x}_0). \tag{2.26}$$

The path may very well carry no light and the throughput consequently be zero, as is the case if none of the constituent path vertices are situated on an emissive surface.

2.4.2 Path Tracing

One of the simplest methods for simulating global illumination is *path tracing* (PT). While it is simple it is also the method that is the most central component of all light transport algorithms in that more advanced algorithms build upon it in one way or another. Path tracing was introduced by Kajiya (1986) as a technique to solve the rendering equation, and builds upon the mechanics of simulating particle transport with Monte Carlo methods as explained by Carter and Cashwell (1975).

In the standard path tracing algorithm also known as unidirectional path tracing, we sample paths from one direction. The algorithm consists of generating a random walk starting from the sensor, resulting in a path $x^{\mathrm{c}} = \mathbf{x}^{\mathrm{c}}_0\mathbf{x}^{\mathrm{c}}_1 \dots \mathbf{x}^{\mathrm{c}}_k$, with $\mathbf{x}^{\mathrm{c}}_0$ lying on the camera aperture. At each vertex along the path, we sample a new direction by importance sampling the BSDF of the material. While we are tracing the path through the scene, we may encounter many emitting surfaces that all contribute to the image. From the series of $k + 1$ vertices constituting the path we can construct k different paths $\mathbf{x}_0^\mathbf{c} \dots \mathbf{x}_i^\mathbf{c}$, $i = 1, \dots, k$, and their aggregate contribution to the measurement integral estimate is

$$F = \sum_{i=1}^{k} \frac{f(\mathbf{x}_0^\mathbf{c} \dots \mathbf{x}_i^\mathbf{c})}{\pi(\mathbf{x}_0^\mathbf{c} \dots \mathbf{x}_i^\mathbf{c})} \tag{2.27}$$

where $\pi(\mathbf{x}_0^\mathbf{c} \dots \mathbf{x}_i^\mathbf{c})$ is the pdf of the i-th sub-path.

Since we are naively tracing paths in the hope of eventually hitting an emitting light source by chance, this technique is often called naive path tracing. Figure 2.7a shows an illustration of how a path is sampled in naive path tracing.

Next Event Estimation

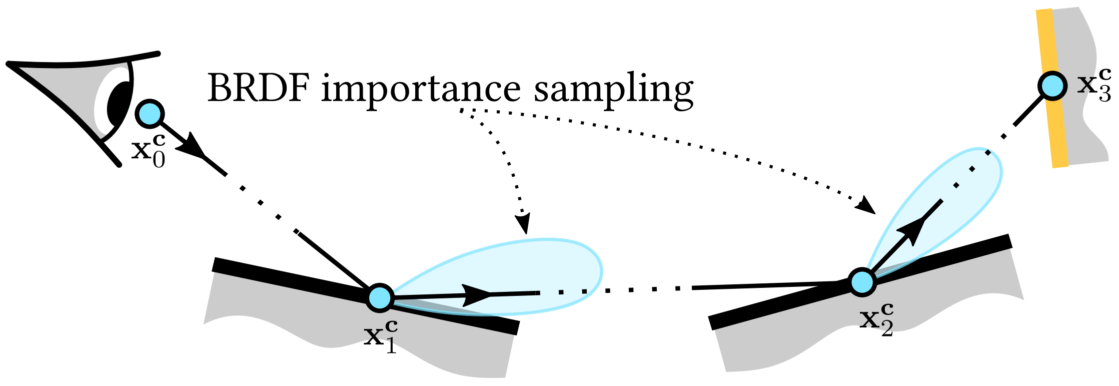
While generating paths, importance sampling the material BSDFs is not always the best option to reduce variance. If the scene consists of mainly diffuse surfaces and a few small light sources that are responsible for the majority of the illumination in the scene, then naively bouncing rays around may result in very few hits on the emitters and many paths will not carry any light. A way to work around this is to use a technique called *next event estimation* (NEE). Next event estimation importance samples the emitters of the scene and is for this reason also known as direct light sampling (Shirley and Wang 1994).

Paths that just terminate without hitting an emitter and thus contributing to the image is wasted computational work and we want to avoid that. Next event estimation samples a vertex $\mathbf{x}_0^\mathbf{e}$ on an emitting light source and sends a shadow ray from the i-th path vertex for each of the k sub-paths towards the sampled point on the emitter to check for visibility, resulting in k paths $\mathbf{x}_0^\mathbf{c} \dots \mathbf{x}_i^\mathbf{c}\mathbf{x}_0^\mathbf{e}$, $i = 1, \dots, k$, where as before $\mathbf{x}_0^\mathbf{c}$ lies on the camera aperture. The contribution of these paths are then

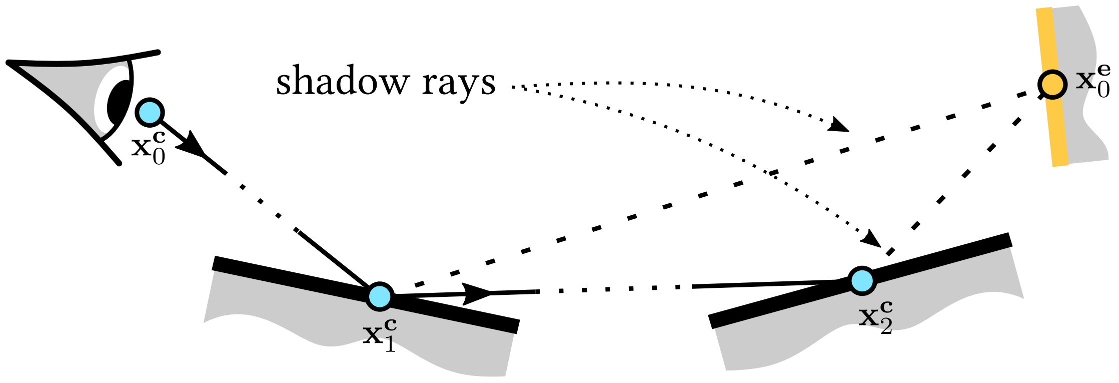
$$F = \sum_{i=1}^{k} \frac{f(\mathbf{x}_0^\mathbf{c} \dots \mathbf{x}_i^\mathbf{c}\mathbf{x}_0^\mathbf{e})}{\pi(\mathbf{x}_0^\mathbf{c} \dots \mathbf{x}_i^\mathbf{c})\pi(\mathbf{x}_0^\mathbf{e})} \tag{2.28}$$

where $\pi(\mathbf{x}_0^\mathbf{e})$ is the pdf of the sampled emitter vertex.

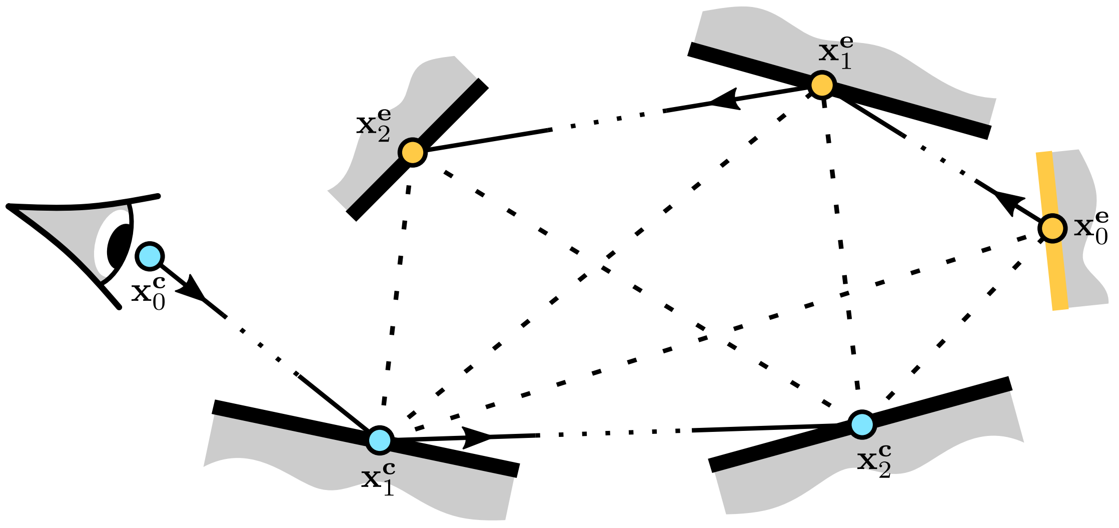
Explicitly connecting the camera paths to a light source in this way can reduce the variance of the measurement integral estimate enormously when the light sources are small and not easily encountered by chance when tracing naively. There are cases when it can reduce efficiency however. The light source may be large and clearly visible, but if the BSDF of the material at the vertex preceding the the emitter is very sharp and scatters light in very specific directions, as is the case with specular and near-specular objects, the path connecting to the emitter may carry very little or no throughput at all. In these cases there is little point in importance sampling the emitter and it is often better to importance sample the BSDF. Figure 2.7b shows an illustration of how a path is sampled with next event estimation.


BRDF importance sampling
$\mathbf{x}_0^{\mathbf{c}}$
$\mathbf{x}_1^{\mathbf{c}}$
$\mathbf{x}_2^{\mathbf{c}}$
$\mathbf{x}_3^{\mathbf{c}}$

(a) Naive/unidirectional path tracing.


shadow rays
$\mathbf{x}_0^{\mathbf{c}}$
$\mathbf{x}_1^{\mathbf{c}}$
$\mathbf{x}_2^{\mathbf{c}}$
$\mathbf{x}_0^{\mathbf{e}}$

(b) Next event estimation.


$\mathbf{x}_1^{\mathbf{e}}$
$\mathbf{x}_2^{\mathbf{e}}$
$\mathbf{x}_0^{\mathbf{e}}$
$\mathbf{x}_0^{\mathbf{c}}$
$\mathbf{x}_1^{\mathbf{c}}$
$\mathbf{x}_2^{\mathbf{c}}$

(c) Bidirectional path tracing.

Figure 2.7: Illustration of paths generated by naive path tracing, next event estimation and bidirectional path tracing. **(a)** Naive PT initiates a random walk from the camera in the hopes of reaching the emitters of a scene by chance. **(b)** NEE connects vertices of the camera paths to the light sources directly. **(c)** BDPT connects paths generated from both the camera and the emitters.

Multiple Importance Sampling

As separate techniques in isolation, the performance of naive path tracing and next event estimation is good in some cases and worse in others. Because of this we can use both techniques at the same time and get the best of both worlds by combining and weighting together paths that were sampled by regular path tracing where we just scatter and propagate the ray naively, and paths that were sampled by using next event estimation to importance sample emitters. This technique is called *multiple importance sampling* (MIS) which stems from the fact that we are importance sampling in multiple ways instead of choosing one way.

Consider the general case when we want to estimate an integral

$$\int_\Omega f(x)\mathrm{d}\mu(x) \tag{2.29}$$

and have n_s different sampling techniques, where we draw n_i samples for the i-th strategy and we have $n = \sum n_i$ samples in total. Denoting the j-th sample from strategy i by x_{ij}, we can then form the *multi-sample estimator* of the integral as

$$F = \sum_{i=1}^{n_s} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(x_{ij}) \frac{f(x_{ij})}{\pi_i(x_{ij})}. \tag{2.30}$$

The estimator above contains weighting functions w_i for each sampling strategy that allows the samples to be weighted differently depending on which technique they were sampled from. F is also unbiased given some weak conditions imposed on the weighting functions, such as that they must form a partition of unity. Designing these weighting functions intelligently can result in variance reduction.

There is great freedom in choosing the weighting functions. A good alternative is to choose the weights according to the *balance heuristic* (Veach and Guibas 1995b)

$$w_i(x) = \frac{n_i \pi_i(x)}{\sum_{k=1}^{n_s} n_k \pi_k(x)}. \tag{2.31}$$

The balance heuristic is a natural way to extend the standard Monte Carlo estimator to the case of multiple sampling strategies. To see this, define the $c_k = n_k/n$ as the fraction of samples coming from the k-th sampling strategy with pdf π_k, and insert the balance heuristic weights into the multi-sample estimator. We get

$$\begin{aligned} F &= \sum_{i=1}^{n_s} \frac{1}{n_i} \sum_{j=1}^{n_i} \left(n_i \frac{\pi_i(x_{ij})}{\sum_k n_k \pi_k(x_{ij})} \right) \frac{f(x_{ij})}{\pi_i(x_{ij})} \\ &= \frac{1}{n} \sum_{i=1}^{n_s} \sum_{j=1}^{n_i} \frac{f(x_{ij})}{\sum_k c_k \pi_k(x_{ij})} \\ &= \frac{1}{n} \sum_{i=1}^{n_s} \sum_{j=1}^{n_i} \frac{f(x_{ij})}{\tilde{\pi}(x_{ij})} \end{aligned} \tag{2.32}$$

where we have used the *combined sample density* to highlight the similarity to the standard Monce Carlo estimator

$$\tilde{\pi}(x) = \sum_{k=1}^{n_s} c_k \pi_k(x). \tag{2.33}$$

The combined sample density functions as a weighted average of the probability distribution functions for the different strategies.

With the multi-sample estimator we can combine next event estimation and unidirectional path tracing. Any fully connected path from the camera to an emitter $x = \mathbf{x}_0\mathbf{x}_1 \ldots \mathbf{x}_k$ can be sampled by both path tracing, $x_{\mathrm{PT}} = \mathbf{x}_0^{\mathrm{c}}\mathbf{x}_1^{\mathrm{c}} \ldots \mathbf{x}_k^{\mathrm{c}}$ where the camera path has encountered the emitter by chance, and next event estimation, $x_{\mathrm{NEE}} = \mathbf{x}_0^{\mathrm{c}}\mathbf{x}_1^{\mathrm{c}} \ldots \mathbf{x}_{k-1}^{\mathrm{c}}\mathbf{x}_0^{\mathrm{e}}$ where the vertex on the emitter was sampled explicitly. With this perspective in mind, regardless of x was actually sampled, we can view the two alternatives x_{PT} and x_{NEE} as two sampling strategies that can be combined in the MIS framework.

In this simple case, we have two sampling strategies, NEE and PT. The samples for the two strategies x_{NEE} and x_{PT} are the same path x and we will drop the subscript for brevity. The multi-sample estimate of the path contribution with weights according to the balance heuristic is then

$$F = f(x)\left(\frac{w_{\mathrm{NEE}}(x)}{\pi_{\mathrm{NEE}}(x)} + \frac{w_{\mathrm{PT}}(x)}{\pi_{\mathrm{PT}}(x)}\right) = \frac{f(x)}{(\pi_{\mathrm{NEE}}(x) + \pi_{\mathrm{PT}}(x))/2}. \tag{2.34}$$

Here we can intuitively see the effect that the MIS scheme has on the contribution estimate. Dividing by the average of the two probability distribution functions of the sampling strategies instead of choosing one or the other, we are less prone to variance in either of the two.

2.4.3 Bidirectional Path Tracing

Combining sampling strategies with MIS in a multi-sample estimator can be extended beyond unidirectional path tracing and next event estimation. *Bidirectional path tracing* (BDPT) is a light transport algorithm that extends this framework into a larger more generalized family of sampling strategies (Lafortune and Willems 1993; Veach and Guibas 1995a).

As the algorithm name suggest, bidirectional path tracing samples paths by random walking both from an emitter and from the camera, and combines many ways to connect the resulting camera and emitter paths at some or all of the vertices in both paths. Each such way to connect the two paths is a *connection strategy*, and the different connection strategies that are considered are weighted in a multiple importance sampling scheme. Figure 2.7c shows an illustration of the many ways that the emitter and camera subpaths can be connected in bidirectional path tracing.

Let k_C and k_E denote the number of vertices of the sampled camera and emitter paths respectively. BDPT samples the emitter and camera paths

$$\begin{aligned} x^{\mathrm{e}} &= \mathbf{x}_0^{\mathrm{e}}\mathbf{x}_1^{\mathrm{e}} \ldots \mathbf{x}_{k_E}^{\mathrm{e}} \\ x^{\mathrm{c}} &= \mathbf{x}_0^{\mathrm{c}}\mathbf{x}_1^{\mathrm{c}} \ldots \mathbf{x}_{k_C}^{\mathrm{c}} \end{aligned} \tag{2.35}$$

and uses these to construct full transport paths

$$x_{s,t} = \mathbf{x}^{\mathbf{e}}_0 \dots \mathbf{x}^{\mathbf{e}}_{s-1}\mathbf{x}^{\mathbf{c}}_{t-1} \dots \mathbf{x}^{\mathbf{c}}_0. \tag{2.36}$$

Here, s and t are the number of vertices used from the camera and emitter sub-paths respectively to make the connection and build the full path, and the tuple (s, t) is the connection strategy representing this specific number of vertices from the two. For full paths of length $k = s + t$ there are $k + 2$ connection strategies, and BDPT employs an MIS scheme to form an estimator that takes all these strategies into account.

Note here that in the connection strategy where $s = 0$, there are no emitter vertices and we refer to the case when the path was sampled entirely from the camera and hit the emitter by chance. This is exactly the case for naive path tracing. The connection strategy where $t = 0$ is almost identical conceptually, but here we instead refer to a scenario where the path was sampled entirely from the light source and hit the camera aperture by chance, like naive path tracing in the opposite direction.

One-Bounce Strategies

We proceed to show some examples of sampling strategies (s, t) in a simple scenario and illustrate how naive PT and NEE belong to a subset of the bidirectional sampling strategies. Consider the case of a transport path with three vertices; one vertex at the camera and emitter each and one vertex between the camera and emitter where a bounce occurs. The available sampling strategies are illustrated in Figure 2.8 where their individual behaviours are also explained. It is easy to see how the number of available sampling strategies grows combinatorially as the full transport paths become longer in more complicated scenarios.

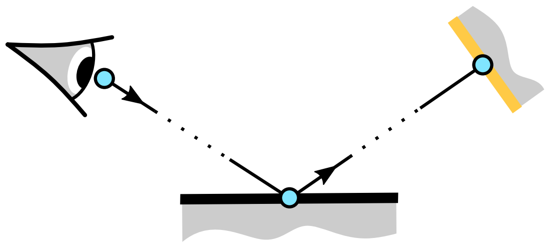
Optimally Combining Strategies

The multi-sample estimator for the throughput is formed by considering and weighting all the different available connection strategies (s, t) that a full path could be connected with from the emitter and camera sub-paths x^{e} and x^{c} sampled by BDPT. The estimator is

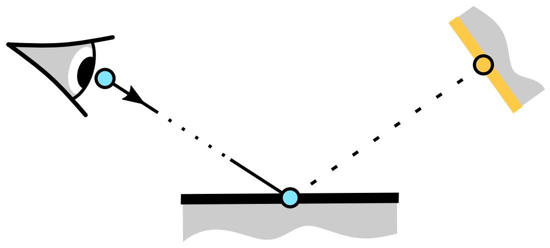
$$F = \sum_{s=0}^{k_C} \sum_{t=0}^{k_E} w_{s,t}(x_{s,t}) \frac{f(x_{s,t})}{\pi_{s,t}(x_{s,t})} \tag{2.37}$$

where the weights $w_{s,t}$ are chosen according to some combination strategy such as the balance heuristic.

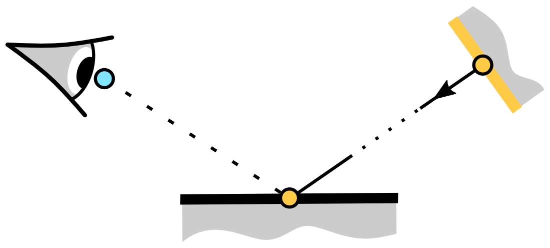
The different connection strategies all have strengths and weaknesses and perform at a varying degree for different path types. Similarly to how next event estimation combines importance sampling of light sources and BSDFs, bidirectional path tracing can increase efficiency substantially by combining the strengths of the different sampling techniques. This can lead to a significant variance reduction when the illumination in the scene is governed by complex light transport paths such as specular and caustic chains between the sensor and the scene emitters. There are cases when a simpler unidirectional path tracing algorithm is to be preferred, where the variance reduction achieved by using a bidirectional sampler is not large enough to warrant the added computational cost.



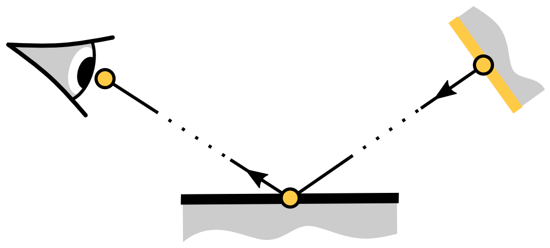
(a) $(s, t) = (0, 3)$: A camera vertex and direction is sampled and the camera ray is propagated and scattered once. The camera path finds the emitter by chance. This is the naive PT strategy.



(b) $(s, t) = (1, 2)$: A camera vertex and direction is sampled and the camera ray is sent out into the scene to find the next vertex on the camera path. One emitter vertex is sampled on the emitter to which the camera path is connected. This is the NEE strategy.



(c) $(s, t) = (2, 1)$: An emitter vertex and direction is sampled and the emitter ray is propagated to the next hit. One camera vertex is sampled to which the emitter path is connected. This strategy is similar to NEE, but instead of directly sampling the light source and connecting to the camera path vertices, the camera aperture is directly sampled and connected to the emitter path vertices.



(d) $(s, t) = (3, 0)$: An emitter vertex and direction is sampled and the emitter ray is propagated and scattered once. The emitter finds the camera aperture by chance. This strategy is analogous to naive PT, with the modification that the random walk starts from the emitter instead of the camera.

Figure 2.8: Bidirectional connection strategies for paths involving one scattering event.

2.4.4 Metropolis Light Transport

Metropolis light transport (MLT) was introduced by Veach and Guibas (1997) and is a Markov chain Monte Carlo technique that samples paths using the Metropolis-Hastings algorithm.

MLT is initialized by estimating the integration constant a across the image using bidirectional path tracing. The samples from this estimation are then also used as initial distribution of the Markov chain. BDPT is unbiased and produces paths distributed according to the contribution function f, which therefore ensures that the Markov chain will have f as stationary distribution and will visit states distributed according to f exactly as it evolves and not be plagued by start-up bias. MLT initialized by an unbiased Monte Carlo integrator is therefore also unbiased.

More paths for the measurement integral estimator are then sampled from the distribution by evolving the chain according to the Metropolis-Hastings algorithm, where new states are accepted or rejected depending on the contribution ratio of the new and old state and the ratio of proposal probabilites. The proposal kernel r proposes changes to the paths that can be partitioned into perturbation and mutation strategies. Perturbations are small changes to the state of the Markov chain that explore the neighbourhood of the path by changing the path vertices by small amounts, and mutations are larger changes to the chain that can change the path structure by for instance adding or removing vertices.

MLT is a highly efficient algorithm to simulate complex light transport involving light source occlusion and long specular path chains, but sometimes has a problem with chains getting stuck in narrow areas of path space associated with high contribution, resulting in firefly artifacts in the image. There are extensions to the traditional MLT algorithm that devise different techniques to alleviate this issue and increase the stratification of the states visited by the Markov chain and consequently the samples generated by the algorithm. For a more thorough review of light transport algorithms based on MCMC simulation, we refer to expositions in the introductory chapters of works by Bitterli (2015) and Jakob (2013).

3 Gradient-Domain Light Transport

This chapter introduces the field of gradient-domain rendering (Lehtinen et al. 2013). Algorithms of this category not only try to estimate the image but also how the image changes, and reconstruct the final image at a higher level of efficiency by using additional information. The change in the image can refer to how the color varies over the image plane or over time if generating image sequences. To calculate the change in the image these algorithms extract information about the image gradient. The key to the success of these algorithms is the ability to do so in a correlated fashion, resulting in a pixel color estimator with lower variance than usual. The components that are estimated in this process is illustrated in Figure 3.1. Gradient rendering works best in scenarios where the frequency spectrum of the contribution function is such that there is more energy for low frequencies than high frequencies, which is often the case in light transport integrands and natural images (Ruderman 1994; Simoncelli and Olshausen 2001).

Applications in computer graphics has found different uses of gradients such as irradiance caching and image editing (Pérez, Gangnet, and Blake 2003; Ward, Rubinstein, and Clear 1988). Previous work commonly uses closed form approximations of the derivative which can not be used during reconstruction to form an unbiased estimation of the final image. Gradient-domain light transport algorithms therefore perform gradient related calculations in an unbiased fashion, allowing the reconstruction of unbiased final images.

In the following we begin by giving a mathematical formulation of the gradients that are estimated, and continue by explaining how the estimated gradient information in the image can be used to retreive the final conventional image through Poisson reconstruction. We further explain how the aforementioned gradients are sampled and present an

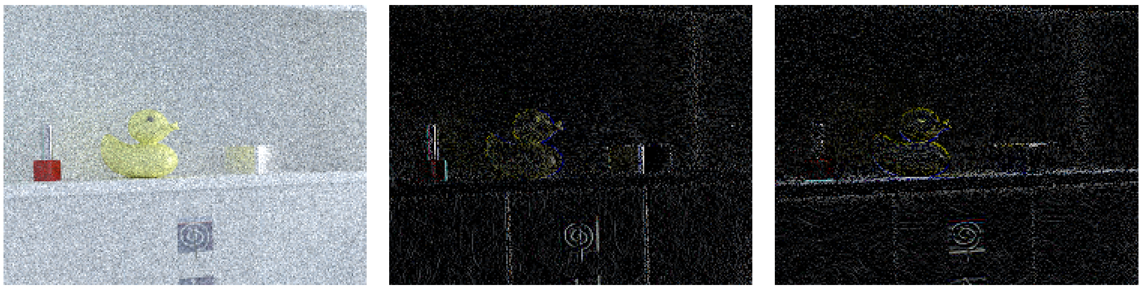


Figure 3.1: Components sampled in the generation of images through gradient-domain rendering. A gradient-domain rendering algorithm computes coarse estimates of the primal image, and horizontal and vertical gradient images, and uses the three parts to reconstruct the final image using a Poisson solver. Crop from the Bathroom 3 scene, 512spp.

overview of how the shift mapping used to generate offset paths for finite differencing is implemented for a range of gradient-domain rendering algorithms. Finally we mention advantages while drawing attention to problems and limitations of current sampling strategies and reconstruction methods.

3.1 Mathematical Formulation

Rendering in the gradient domain relies on the efficient estimation of finite differences, so let us first define the finite difference $\Delta_{i,j}$ between two pixels i and j. It should be computed as the difference between the two pixel values I_i and I_j, hence we have

$$\Delta_{i,j} = I_i - I_j = \int_\Omega h_i(x)f(x)\mathrm{d}\mu(x) - \int_\Omega h_j(x)f(x)\mathrm{d}\mu(x). \tag{3.1}$$

Here, Ω is the space of all paths (path space), h_j is the pixel filter of pixel j, $x \in \Omega$ is a path of arbitrary length k consisting of a sequence of vertices $\mathbf{x}_0, \ldots, \mathbf{x}_k$, $f(x)$ is the image contribution function describing the amount of light reaching the sensor through a given path x, and $\mathrm{d}\mu(x)$ is the area product measure $\prod_{i=0}^{k} \mathrm{d}A(\mathbf{x}_i)$.

Rather than computing these integrals separately and taking their difference, we wish to reformulate the expression to a single integral over path space that computes the gradient. The reason for this is that the quality of the image that is ultimately reconstructed from the gradient information depends on how similar the base and offset paths are (Kettunen et al. 2015). We want the base and offset paths to be as similar as possible, and to achieve this we define T_{ij} as a *shift mapping* that deterministically shifts a base path x to an offset path $T_{ij}(x) = \tilde{x}$. Using this shift mapping we can rewrite the second integral corresponding to pixel j as

$$\begin{aligned} I_j &= \int_\Omega h_j(x)f(x)\mathrm{d}\mu(x) \\ &= \int_{T_{ji}(\Omega)} h_j(T_{ij}(x))f(T_{ij}(x))\mathrm{d}\mu(T_{ij}(x)) \\ &= \int_\Omega h_j(T_{ij}(x))f(T_{ij}(x))\left|\frac{\mathrm{d}T_{ij}}{\mathrm{d}x}\right|\mathrm{d}\mu(x) \\ &= \int_\Omega h_i(x)f(T_{ij}(x))\left|\frac{\mathrm{d}T_{ij}}{\mathrm{d}x}\right|\mathrm{d}\mu(x). \end{aligned} \tag{3.2}$$

Here, in the second row, we have shifted the integration variable x using the shift mapping T_{ij} defined above, and correspondingly shifted the domain of integration Ω in the other direction using the inverse of the shift mapping $T_{ji} = T_{ij}^{-1}$. Following this shift, in the third row, we have performed a change of integration variable from inversely shifted paths to the original path space Ω, hence the appearance of the Jacobian determinant $|\mathrm{d}T_{ij}/\mathrm{d}x|$. The final row is obtained by noting that $h_j(T_{ij}(x)) = h_i(x)$, which follows from properties of pixel filters when shifting paths exactly by a unit pixel distance.

The gradient can now be written as one integration over path space as desired,

$$\begin{aligned}\Delta_{i,j} &= \int_\Omega h_i(x)\left(f(x) - f(T_{ij}(x))\left|\frac{\mathrm{d}T_{ij}}{\mathrm{d}x}\right|\right)\mathrm{d}\mu(x) \\ &= \int_\Omega \left(f_i(x) - f_i(T_{ij}(x))\left|\frac{\mathrm{d}T_{ij}}{\mathrm{d}x}\right|\right)\mathrm{d}\mu(x).\end{aligned} \tag{3.3}$$

Each path x that is sampled can now be shifted deterministically using the shift mapping, forming correlated sample pairs which can be used in the single integral in Equation 3.3 to estimate gradients.

The challenge in the implementation of a sampler for this new integrand by extending a conventional rendering algorithm is to design the shift mapping function. The goal is to construct the shift mapping so that the generated path pairs are similar and close to each other in path space, leading to small differences in the throughput they carry and consequently lower variance in the gradients. Another challenge when building the algorithm is to compute the Jacobian determinant of the designed shift mapping. Current algorithms that sample gradients in this way use shift mapping functions that generate horizontal and vertical offset paths from the base path, that allows for the estimation of horizontal and vertical image gradients to be used in the final image reconstruction step. We will review some of the current gradient rendering algorithms in Section 3.3.

To recover the final image by integrating the gradients in the image reconstruction step is the sole purpose of devising a way to sample the gradients in the first place, and we thus continue by describing this reconstruction step below.

3.2 Poisson Reconstruction

The technical details behind how the gradients are sampled may vary depending on the gradient rendering algorithm used and is often tied to the underlying conventional sampling algorithm. Once the primal and gradient image estimates are formed however, the process of reconstructing the final image by solving a screened Poisson equation is common to every method.

Before giving an account on how shift mappings are implemented and gradient image estimates are formed in practice, which is specific to the type of gradient-domain rendering algorithm chosen, we decribe the basic principle on which gradient rendering relies; that of utilizing the information contained in the gradients to obtain the final image through integration.

3.2.1 The Poisson Equation

We start by formulating the traditional Poisson equation. Consider the scenario where we have measurements $g = (g_x, g_y)$ of a gradient field. In the context of this thesis, g_x and g_y are the horizontal and vertical image gradients that we have access to, given that a rendering algorithm capable of sampling gradients is available. A natural question to ask is what the potential function ϕ that would produce the gradient field g is, so that

$$\nabla\phi = g. \tag{3.4}$$

Specifically in the area of image synthesis, we ask what the image that corresponds to the measured horizontal and vertical gradients is. This question is answered through integration. We are ultimately interested in finding this final image and wish to investigate how to do so through the use of sampled gradients. Consider also the divergence of the gradient field

$$\nabla \cdot g = \frac{\partial g_x}{\partial x} + \frac{\partial g_y}{\partial y} = v. \tag{3.5}$$

Combining 3.4 and 3.5 we can produce a scalar equation for ϕ to find the integral whose derivatives we had

$$\nabla \cdot \nabla \phi = \nabla^2 \phi = v. \tag{3.6}$$

This is the continuous non-screened Poisson equation and its solution gives the potential function matching the gradient field.

3.2.2 Accounting for the Primal Image

From another point of view, the minimizing solution I of all candidate solutions ϕ is the function that minimizes the L_2 distance from the measured gradient data, or more explicitly written as

$$I = \underset{\phi}{\operatorname{argmin}} \, ||\nabla \phi - g||_2. \tag{3.7}$$

That is, the resulting image I from the optimization is the best match to the measurements in the least squares sense. Note now that if you have at hand the derivative of an unknown function and you successfully integrate this derivative without any additional information, the function can only be retreived up to an unknown constant term. This has the implication that if we only integrate measured image gradients, we will have no information about the absolute color and brightness of the image, only how the image varies over the image coordinates. Hence we wish to augment the Poisson equation with a screening term that accounts for and considers measurements of the primal image values. These primal image values are the regular noisy estimates p of the final image I from a conventional rendering algorithm. Instead of evaluating the right hand side of Equation 3.7, we find the solution as

$$I = \underset{\phi}{\operatorname{argmin}} \, (||\alpha(\phi - p)||_2 + ||\nabla \phi - g||_2) \tag{3.8}$$

where α is a reconstruction parameter that decides the relative importance of the screening term when finding the minimizing image I.

In the continuous case, it is possible to prove through variational calculus that the solution 3.8 satisfies the screened Poisson equation

$$\alpha^2 \phi - \nabla^2 \phi = \alpha^2 p - v. \tag{3.9}$$

The same result can be derived in the discrete case, and what you get there is the exact discrete analogy of this. Consider the following system of equations

$$A\phi = b. \tag{3.10}$$

Here, b consists of the measured primal samples and horizontal and vertical gradient samples, and A is a block matrix with the following structure

$$A = \left[\begin{array}{c} \alpha \\ \hline \Delta X \\ \hline \Delta Y \end{array}\right]. \tag{3.11}$$

The top matrix is a diagonal matrix with α on the diagonal, and the middle and bottom matrices perform finite differencing in the horizontal and vertical directions respectively. Then, the pseudo-inverse solution to the system of equations 3.10 takes the measured data in b to evaluate $\alpha^2 p - v$, after which it multiplies this vector by the matrix $[\alpha^2 - \nabla^2]^{-1}$. Ergo, the pseudo-inverse solves the discrete screened Poisson equation to recover I, the best image candidate ϕ whose gradients correspond to g, after screening the solution with the primal measurements p weighted by α.

Loss Functions in the Optimization Problem

Equation 3.8 formulates the Poisson reconstruction optimization problem, which consists of two quantities that we aim to minimize; the respective loss functions related to the primal and gradient measurements.

Changing the loss functions in the optimization problem generally produces a different resulting image I. Changing the loss function related to the primal part has shown to give bad results. In general, we write the Poisson reconstruction optimization problem as

$$I = \underset{\phi}{\operatorname{argmin}} \left(\|\alpha(\phi - p)\|_2 + L_g(\nabla\phi - g) \right) \tag{3.12}$$

where L_g is some loss function that is used for the gradient part.

In our work, we will change the loss function of the gradient part and compare the results to the previously used L_1 and L_2 reconstruction methods where $L_g(\cdot) = \|\cdot\|_1$ and $L_g(\cdot) = \|\cdot\|_2$ respectively.

3.3 Sampling of Gradients

So far we have introduced the concept of a shift mapping function that allows the generation of offset paths from base paths to generate path pairs. The path pairs can then be used to compute the integral that constitutes a gradient sample to generate horizontal and vertical gradient images, which are then used in conjunction with the primal image to reconstruct the final image after solving a screened Poisson equation. We have not however explained how the shift mapping and gradient sampling is implemented in practice, and aim to do this next.

Implementing algorithms that can succesfully sample the integrand in Equation 3.3 comes down to designing the shift mapping operator to be used in conjunction with a conventional Monte Carlo based sampler. Most implementations currently extend a conventional algorithm to generate offset paths that differ from the base path only in the first few path vertices. The shift mapping is designed to offset the base path so that it intersects the screen at an integral number of pixel distances away from the base path in the horizontal or vertical direction apart, trace the offset path through the scene as usual and reconnect it to the base path as soon as possible. A typical simple path pair is illustrated in Figure 1.4.

In the text to come we present an overview of some algorithms designed to sample path pairs using a shift mapping on top of the conventional sampler in order to estimate gradients.

3.3.1 Gradient-Domain Metropolis Light Transport

The first algorithm to utilize gradient sampling for rendering was introduced by Lehtinen et al. (2013). They introduce *gradient-domain metropolis light transport* (G-MLT), which is an extension of regular MLT where the sampler is driven to concentrate its efforts in regions that contribute highly to the gradient of the image rather than areas where the regular throughput integrand is of high magnitude. The Metropolis sampler used to mutate base paths is driven by a target function that considers both regular throughout and gradient throughput.

G-MLT samples base paths using a regular MLT mutator and computes gradients by shifting the base path in the horizontal and vertical direction simultaneously. This is done by driving a single Markov chain, randomly selecting whether to compute the gradient by shifting in the positive or negative direction. To this end, G-MLT uses an extended definition of path space that includes bits to denote the direction of the shift, denoted as

$$\Omega' = \Omega \times \{(+1, 0), (-1, 0), (0, +1), (0, -1)\}. \tag{3.13}$$

The elements of the extended path space Ω' contain a regular base path and a horizontal and vertical shift direction bit. The conventional Metropolis sampler generates extended base paths from Ω', and offset paths are then generated from the base path using the shift mapping function, in the directions specified by the shift direction bits.

Similarly to Veach and Guibas' lens mutator (1997), the shift mapping is implemented by shifting the base path deterministically so that it intersects the screen one pixel distance away from the base path's intersection. After the shift, the path is propagated in the new direction before it is reconnected to the remaining unperturbed part of the base path. To increase the efficiency of the reconnection strategy in the case of specular-diffuse-specular scattering, the offset path is propagated deterministically using Jakob and Marschner's manifold perturbation to stay on the light-carrying part of the specular manifold (2012). A specular manifold is defined as a set of path vertices that satisfy the constraints imposed by positions and directionality of light sources and reflective or refractive surfaces. This case is illustrated in Figure 3.2. Shifting only until the first non-specular vertex hit as in the conventional lens mutation strategy is inefficient since the path will carry no light when reconnected to the next specular vertex that follows.

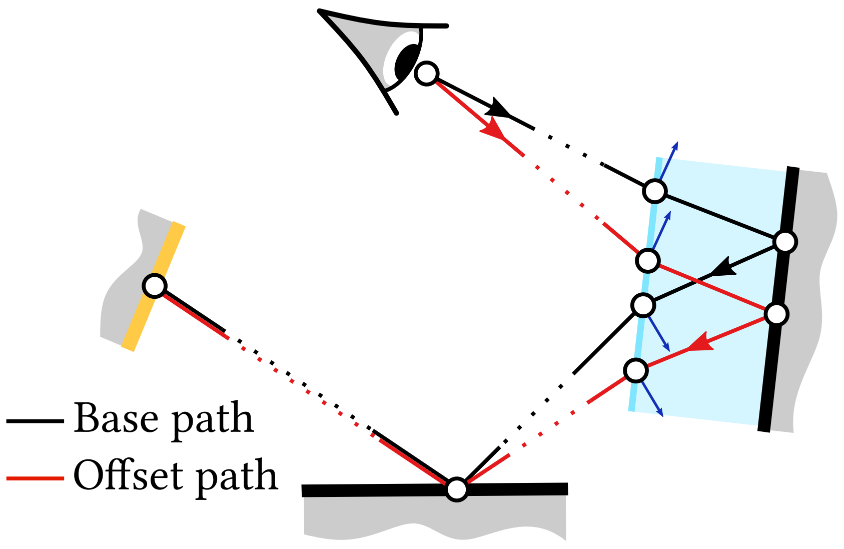


Figure 3.2: Illustration of the manifold perturbation based shift for G-MLT and G-BDPT. The shift is performed such that the path stays on the specular manifold and reconnects to the first diffuse vertex encountered in the case of a specular-diffuse-specular chain.

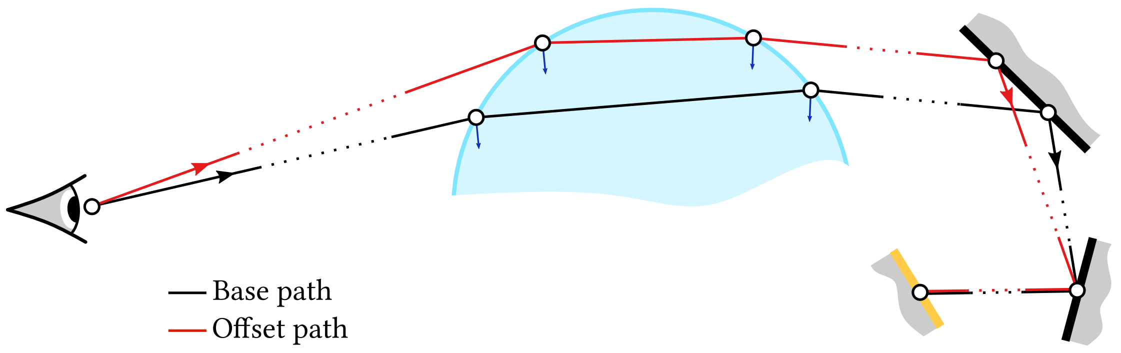


Figure 3.3: Illustration of the half vector preserving shift used in G-PT that propagates the offset path along the first specular chain and reconnects to the base path as soon as the current offset vertex and the next two consecutive base vertices are classified as diffuse. The shift is performed such that the half vectors (blue arrows) are preserved along the first specular chain.

Improved Sampling Strategies

Manzi et al. (2014) bring improvements to the gradient-domain Metropolis framework in the form of better weighting strategies to combine sampling techniques, a modified symmetric shift mapping operator that reduces variance and the occurence of singularities in the gradients, and structure adaptive gradient sampling that can obtain sparser gradients by taking scene features such as geometric discontinuities and texture edges into account.

3.3.2 Gradient-Domain Path Tracing

Kettunen et al. (2015) showed that it is possible to extend the gradient sampling framework to non-Metropolis based Monte Carlo samplers such as path tracing as well, and proceed to introduce *gradient-domain path tracing* (G-PT). The method employs a simpler shift mapping than the one used for G-MLT, giving an advantage to G-PT in terms of ease of implementation. The shift is half-vector preserving for the first specular chain, and reconnects to the base path as soon as the current offset vertex and both the current and next base vertices are classified as diffuse, illustrated in Figure 3.3. For each base path, the algorithm generates an offset path for each four neighboring pixels and outputs four gradient images and one primal image used to reconstruct the final image.

To ensure the correct computation of pixel differences during sampling, G-PT builds on the symmetric shift mapping operator introduced by Manzi et al. (2014). The method further extends this shift mapping to allow for a multiple importance sampling scheme, leading to a significant reduction in variance. The gradient is computed as a sum of two integrals, one using the forward mapping and one its inverse.

$$
\begin{aligned}
\Delta_{i,j} = &\int_{\Omega} h_i(x) w_{ij}(x) \left(f(x) - f(T_{ij}(x)) \left| \frac{\mathrm{d}T_{ij}}{\mathrm{d}x} \right| \right) \mathrm{d}\mu(x) + \\
&\int_{\Omega} h_j(x) w_{ji}(x) \left(f(x) - f(T_{ji}(x)) \left| \frac{\mathrm{d}T_{ji}}{\mathrm{d}x} \right| \right) \mathrm{d}\mu(x)
\end{aligned}
\tag{3.14}
$$

where the MIS weights w_{ij} and w_{ji} add upp to one and account for shifts that are not invertible, such as when the reconnection from the offset path to the base path fails.

Shift mapping functions cause a change in path densities, and the Jacobian determinant accounts for this transformation. When shifting a path towards a region of path space that is sampled with much higher density by the underlying sampler, commonly close to concave edges and corners in the scene geometry, this can lead to arbitrarily large Jacobians, resulting in clearly visible artifacts in the final image. To avoid this the MIS scheme interprets the forward and inverse mappings T_{ij} and T_{ji} as two ways to sample a base path x; either by sampling it directly, or sampling an offset path $\tilde{x} = T_{ij}(x)$ and generating the base path from the offset path through the inverse mapping $x = T_{ji}(\tilde{x})$. This yields the MIS weight

$$
w_{ij}(x) = \frac{\pi(x)}{\pi(x) + \pi(T_{ij}(x)) |\mathrm{d}T_{ij}/\mathrm{d}x|}
\tag{3.15}
$$

where π is the probability density for paths generated by the sampler. Taking the magnitude of the Jacobian determinant into account, the MIS technique combines different ways of generating the path pairs used for gradient estimation which turns out to be highly efficient in suppressing the problems caused by the change in path densities from the shift, as shown in the bottom row of Figure 3.4.

Bidirectional Sampling for G-PT

As an extension to gradient sampling with unidirectional path tracing, Manzi et al. (2015) introduce *gradient-domain bidirectional path tracing* (G-BDPT). Just as when comparing the performance of standard path tracing to its bidirectional counterpart, G-BDPT is generally more efficient than G-PT when the additional computational overhead of a more expensive bidirectional path sampler is warranted, often in difficult light transport scenarios involving caustics or small, occluded light sources that are difficult to solve using unidirectional methods.

Because the individual number of sampled paths from a bidirectional path tracer is large, using the same shift function as as the one employed in G-PT that naively applies the half vector preserving shift mapping to each sampled path would be prohibitively expensive. G-BDPT therefore selectively removes some of the conventional bidirectional connection strategies to circumvent this and reduce the cost of offset path generation. The algorithm removes sampling strategies that include a specular vertex as the connecting vertex between light and eye subpaths, since connections with non-specular vertices contribute little to the throughput and allow for cheaper computation of the shift mapping function and its Jacobian.

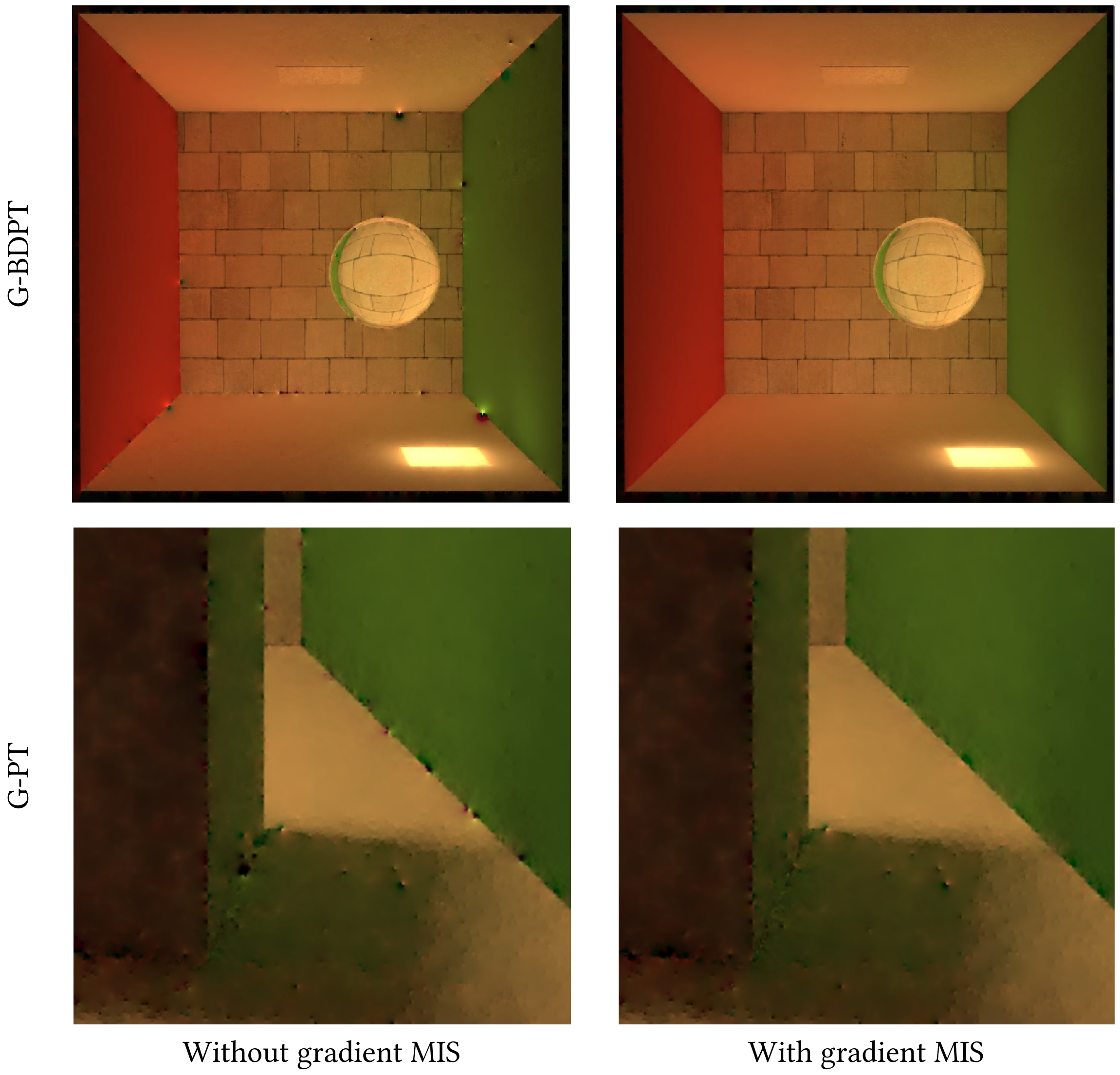


Figure 3.4: The multiple importance sampling scheme that takes the magnitude of the Jacobian determinant of the shift mapping into account is highly efficient in removing artifacts caused by the change in path densities from the shift. Rendered images from Kettunen et al. (2015) and Manzi et al. (2015).

For each gradient sample, G-BDPT combines the gradient-MIS technique employed by G-PT with the regular MIS technique from BDPT by considering both gradient sampling strategies and bidirectional path sampling strategies. The MIS weights are

$$w_{ij;st}(x) = \frac{\pi_{s,t}(x)}{\sum_{k=0}^{s+t} \pi_{k,s+t-k}(x) + \pi_{k,s+t-k}(T_{ij}(x))|\mathrm{d}T_{ij}/\mathrm{d}x|} \tag{3.16}$$

where s, t are the usual BDPT sampling strategies. Similarly to the G-PT case, the combined MIS is efficient in reducing sampling artifacts in the gradients close to concave sections in the scene geometry, illustrated in the top row of Figure 3.4.

In contrast to G-PT, G-BDPT uses the shifting strategy based on manifold perturbation as in G-MLT since conventional BDPT too is expressed naturally in a surface position based path parametrization. The shifted offset path is always reconnected to the base path at the second diffuse vertex (not counting the camera vertex) whereas G-PT requires two consecutive diffuse vertices, as mentioned above. The shift employed by G-BDPT therefore generally produces offset paths that are more similar to the base path than than the offset paths generated by the G-PT shift mapping. Practically, shifting a base path to generate an offset path is done by offsetting pixel by a unit pixel distance, re-trace the first specular chain to the first diffuse vertex hit, then reconnecting back to the second diffuse vertex hit on the base path along the second specular chain through manifold perturbation.

3.3.3 Other Extensions

To improve temporal coherence when rendering animation sequences, Manzi, Kettunen and collaborators (2016) extends the gradient sampling framework to the temporal domain. In their *temporal gradient-domain path tracing* (T-GPT) algorithm, they sample temporal gradients across animation frames and mixed second order spatio-temporal differences in addition to the regular spatial gradients sampled in the gradient-domain rendering algorithms presented above. The final frames are recovered by solving a three-dimentional spatio-temporal screened Poisson equation. They also achieve variance reduction through adaptive sampling, and through the use of motion vectors that take the movement of scene and camera into account when computing gradients. T-GPT has the advantage of leveraging temporal correlation between consecutive animation frames to render sequences at a higher efficiency than ordinary methods which normally render frames individually and blend the frames in a post-processing step.

Other conventional algorithms such as photon mapping (Jensen 2001) and vertex connection and merging have also been extended to implement shift mapping strategies in order to sample gradients. Hua et al. (2017) introduces *gradient-domain photon density estimation* (G-PM) as an extension to stochastic progressive photon mapping (Hachisuka and Jensen 2009), and *gradient-domain vertex connection and merging* (G-VCM) was similarly introduced as an extension to vertex connection and merging by Sun et al. (2017). It is likely that more extensions to the gradient-domain will develop as conventional rendering methods are improved and that new techniques for sampling correlated path pairs at higher efficiency will emerge in the future.

3.4 Advantages and Limitations

The key factor for the success of gradient-domain rendering methods is the ability to perform gradient estimation with shift mappings that produce highly correlated path pairs of base and offset paths. Rather than computing finite differences between two sums of random adjacent paths that are uncorrelated, gradients are estimated by integrating the difference between these correlated path pairs. Typical image contribution functions in rendering contexts representing natural images are typically dominated by low frequency energy. Taking finite differences between path pairs that are in close proximity in path space removes much of the energy from the signal, which is exploited in the Poisson reconstruction. Kettunen et al. (2015) show both empirically and through Fourier analysis how gradient sampling and Poisson reconstruction combines information from the primal image and gradient images by recovering low frequency details using regular rendering, and utilizing the low variance of the sampled gradients to reduce high frequency noise.

Conversely, these rendering methods can perform at a lower efficiency than desired if the shift mapping fails to generate path pairs with high enough correlation to warrant the additional computational overhead. When rendering hair, foliage or other high-frequency, sub-pixel detail, there is less correlation between the base paths and their corresponding shifted paths that constitute the gradient samples since the base paths and offset paths take different routes in path space more often. Reconnection of offset paths to the base path may also fail more often when rendering scenes containing geometry with high-frequency details. The design of more intelligent shift mappings and the development of other means to utilizing correlated sampling with control variates is an active area of research (Rousselle, Jarosz, and Novák 2016). This venue of work towards improvement is orthogonal and complementary to the investigations conducted in this thesis.

3.4.1 Problems Related to the Reconstruction

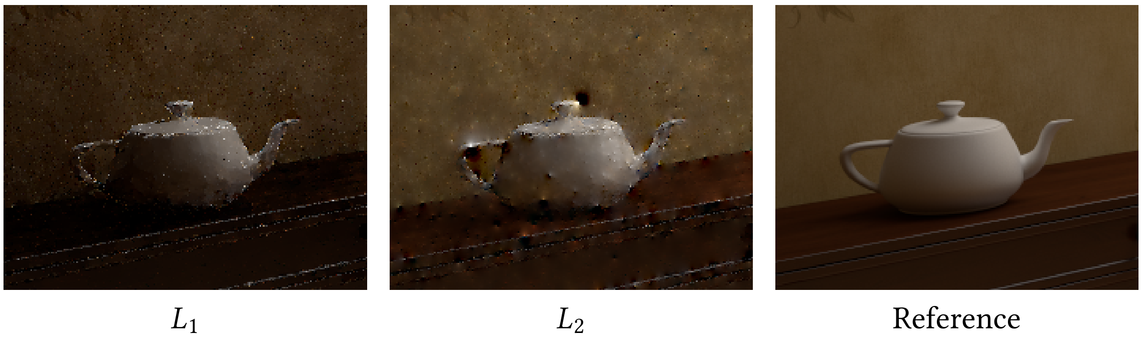


Figure 3.5: Problems with traditional reconstruction methods highlighted in a 16 spp render of the BOOKSHELF scene. *(Left)* L_1 reconstruction supresses outliers well but introduces bias, resulting in a darker image. *(Middle)* L_2 reconstruction is unbiased but introduces dipole-shaped artifacts. *(Right)* Reference image.

Figure 3.5 illustrates some problems with previous reconstruction approaches. Reconstruction using the L_2 loss for the gradient part produces an unbiased image, but it is sensitive to outliers in the gradient image samples, resulting in undesired dipole-shaped artifacts in the final image L_1 reconstruction suppresses this issue at the cost of introducing visible bias. This opens up a direction for further development and an incentive to investigate alternate, improved reconstruction schemes for gradient-domain rendering. As an example, Manzi, Vicini, and Zwicker (2016) developed a reconstruction approach that uses scene normals, texture values, depth and ambient occlusion in addition to the sampled horizontal and vertical gradients to achieve better results.

Motivated by the benefits to be gained from the potential improvements in the reconstruction step, the remainder of this thesis is dedicated to developing a new reconstruction method using a different loss function, which could lead to improved final reconstructed image results.

4 Huber Loss Reconstruction

Until this point we have introduced the reader to the field of light transport simulation, and have specifically given an introduction to the theory and thought process behind extending conventional simulation in rendering to the gradient domain. After highlighting the current state of this sub-group of the field and pointing out problems in the Poisson reconstruction step common to all gradient rendering methods, we are now ready to develop a new reconstruction approach in an attempt to alleviate those issues for gradient-domain path tracing in particular.

This chapter gives an overview of the method developed in this thesis. A motivation behind the use of Huber loss reconstruction for gradient-domain path tracing is given by emphasizing potential benefits while pointing out the necessary obstacles to overcome. We proceed to present and motivate the choice of model family, how our data was generated, how the model was trained and how its ability to generalize to new data was estimated.

4.1 Motivation

In current implementations of gradient-domain light transport algorithms, it is either the L_1 loss or the L_2 loss that is used for the gradient loss part when solving the Poisson reconstruction problem. Using L_1 for the primal loss results in strong bias and is therefore avoided. In the Poisson reconstruction process we are performing maximum likelihood estimation under the following assumptions:

- There exists an unknown final image I that we are trying to recover by integrating its estimated gradients g.
- The sampled primal image p is the unknown final image I corrupted by i.i.d. noise with a known variance.
- The sampled gradient images $g = (g_x, g_y)$ are the gradients of the unknown final image ∇I corrupted by i.i.d. noise with a known variance, not necessarily the same as the variance of the sampled primal image.

Using the L_1 error, one assumes Laplace distributions for the noise in the primal and gradient images, whereas Gaussian distributions are assumed when using the L_2 error. These assumptions are simplifications and unrealistic to a degree as they assume stationarity of noise, independence between primal image noise and gradient image noise, fixed distributions and more. Indeed, for a given pixel, individual samples from the simulation are distributed in a way which can not in general be described as Gaussian.

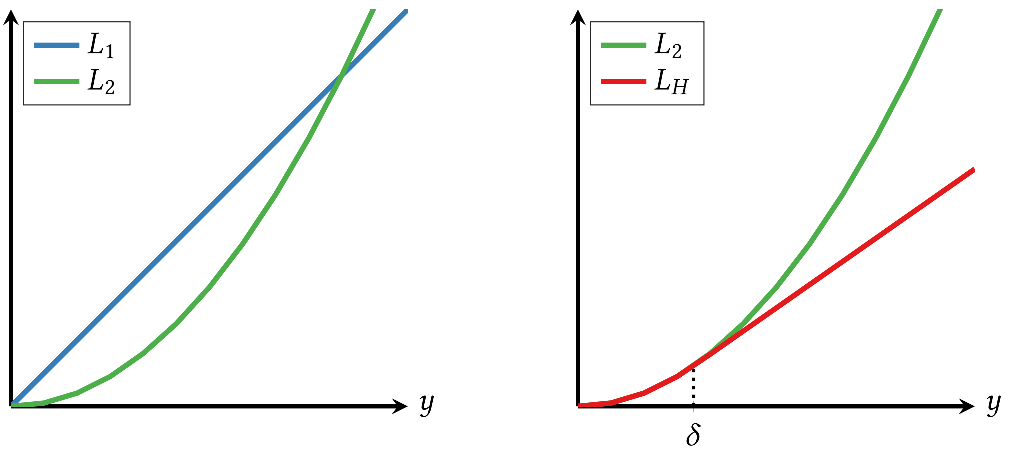


Figure 4.1: Shapes of the compared loss functions. *(Left)* L_1 loss and L_2 loss functions. *(Right)* Huber loss retains the quadratic shape of L_2 for $y < \delta$ and behaves like L_1 for larger y.

The colors in the image pixels do not consist of single samples however but are rather averages of a number of individual samples by virtue of Monte Carlo sampling. The noise at any given pixel thus tends toward a Gaussian distribution as the sample count is increased, by the central limit theorem. For this reason it is justifiable to assume a Gaussian distribution for the noise and consequently use the L_2 error in the maximum likelihood estimation, as long as the sample count is sufficient. That being said, the sample count considered sufficient depends strongly on how well-behaving the distribution of the individual samples is. It is evident empirically that such a sufficiency is not reached in general at reasonable sample counts, as more complicated parts of the image can commonly result in high-magnitude outliers, especially in the gradients.

4.1.1 The Huber Loss Function

The Huber loss function is plotted compared to L_1 and L_2 in Figure 4.1. The function can be defined piecewise (Huber 1964) as

$$L_H(y, \delta) = \begin{cases} y^2, & |y| \leq \delta \\ 2\delta|y| - \delta^2, & |y| > \delta. \end{cases} \tag{4.1}$$

Huber loss is quadratic for small values of y and linear for large values. It combines L_1 loss and L_2 loss by smoothly extending a parabola by a straight line at a cut-off parameter δ. We want to use Huber loss for the gradients in order to properly treat well-behaving samples and give less importance to extreme-value outliers in the sampled gradient images, while keeping the optimization process of the Poisson reconstruction convex. Using Huber loss for the primal part does similar to L_1 introduce too much bias to produce satisfying results.

By the reasoning above, we assume a Gaussian distribution for small-magnitude noise and a Laplace distribution when the noise deviates strongly. The Huber loss function (4.1) satisfies these assumptions as it behaves like the L_2 loss function near the center and the

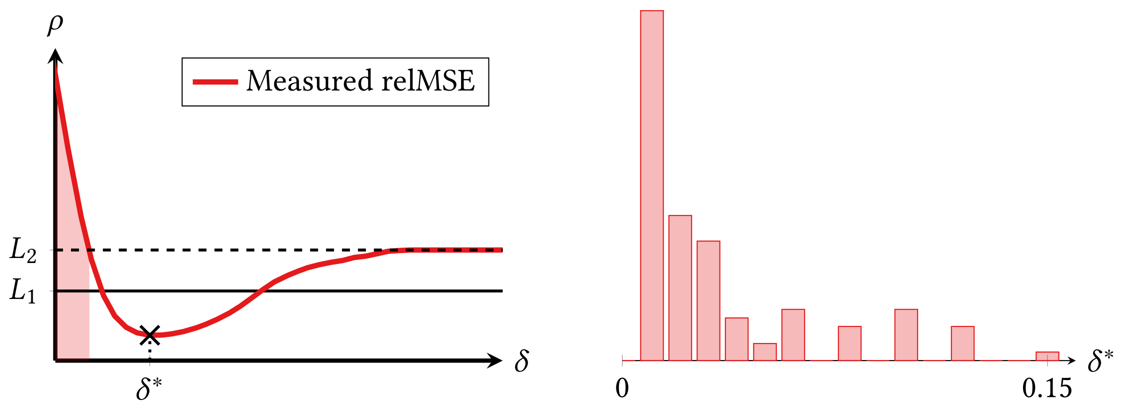


Figure 4.2: Performance of Huber loss reconstruction for different δ. *(Left)* Characteristic log-log plot of measured relMSE ρ resulting from Huber loss reconstruction with different δ on a typical scene from our data set. Reconstruction with large δ approaches the L_2 solution. Using too small δ from the shaded area gives poor performance and should be avoided. *(Right)* Distribution of the optimal parameter δ^* for different scenes. The variation in ρ over different δ and the variation in δ^* over different scenes highlights the need for a method capable of selecting which δ should be used for a given reconstruction scenario.

L_1 loss function toward the tails, which means that high magnitude outliers will be taken less into consideration while still being unbiased for the rest of the noise.

4.1.2 Reduction of Bias

L_1 loss can suppress outliers efficiently compared to L_2 loss att the cost of introducing bias in the reconstructed image. For a successful application of Huber loss to the reconstruction problem it would be preferrable if Huber loss could suppress outliers efficiently without the added downside of introducing as much bias as L_1.

4.1.3 Huber Loss in Practice

Unfortunately, using Huber loss reconstruction is not simply a matter of choosing an arbitrary cut-off parameter and instantly receive better results. In the right plot of Figure 4.2 we see that some scenes have their optimal Huber loss parameter higher than others, and in the left plot we see that using a value that is too low compared to the optimum gives poor results. The Huber loss parameter should therefore be variable and selected per scene.

We want the solution to reduce to the L_2-solution when there are no outliers present in the gradient image and therefore choose to keep the choice of $\alpha = 0.2$ which has been proven to work well by Lehtinen et al. (2013). We support this choice by showing reconstructed images for different α in Appendix B. There is still the problem of choosing the cut-off parameter δ however, the goodness of which is dependent on the rendering scenario.

Figure 4.2 shows the performance of Huber loss reconstruction for different cut-off parameters δ for a given scene, and the distribution of optimal cut-off parameters δ^* over different scenes. For most scenes, there is an interval for δ where Huber is a better choice with regard to relMSE than L_1 or L_2, but using values that stray too low from the optimal value can produce very poor results. In addition to this, the distribution of optimal parameters δ^* varies from scene to scene, so that an optimal value in one scene may be a horrible choice for another scene. Since it is not possible to know which value will be optimal for a certain rendering scenario a priori, it is important to devise a method for predicting δ-values based on metrics that can be extracted during rendering in order to use Huber loss reconstruction effectively. We achieve this through an optimization approach to model parameter estimation. In the coming sections we thus set out to:

- Select a parametric model with desirable properties to predict δ.
- Generate data with statistics from different rendering scenarios.
- Measure how well different δ perform depending on rendering scenario.
- Design a cost function to be minimized when training the model parameters.
- Estimate the trained model's ability to generalize to unseen scenarios.

Poisson Reconstruction with Alternate Loss Functions

When solving the minimization problem specified in Equation 3.8 using some other loss function than L_2, the solution can no longer be found as the pseudo-inverse solution to a sytem of equations. Furthermore, the minimizing solution no longer satisfies the screened Poisson equation exactly; instead we are solving an equation that is as similar as possible while attempting to suppress gradient outlier sensitivity.

For this reason when we are using L_1 or Huber loss for the gradient part, we use the ProxImaL solver which is based on iteratively reweighed least squares implementing the Chambolle-Pock algorithm (Chambolle and Pock 2011; Heide et al. 2016).

4.2 Data Set Structure and Notation

When describing our method in the coming sections, we will refer to components of our data set and therefore begin by defining its structure and notation here. A *sample* or *rendering scenario* is defined as the data we get from rendering a scene at a certain sample count or spp. We denote these samples $\mathbf{y}$ and refer to them as samples or rendering scenarios interchangably, and each $\mathbf{y}$ contains the measurements we have made for that scenario. The sample with label s is denoted $\mathbf{y}_s$. These samples should not be confused with the pixel throughput samples that are averaged to form a measurement of the pixel color through Monte Carlo integration. Rather, the $\mathbf{y}_s$ contain information about which scene, camera angle, lights and render settings are used to produce the image in each case. For the remainder of this thesis however, it suffices to see the samples $\mathbf{y}_s$ as labels to idenfity each rendering scenario that we will make certain measurements on.

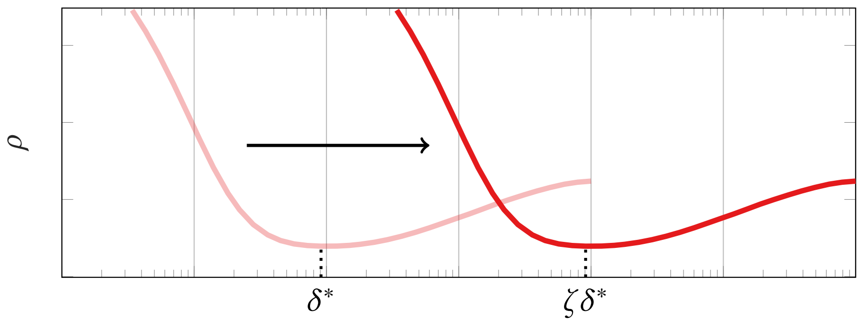


Figure 4.3: Behaviour of δ-relMSE curves when scaling the brightness of the sampled primal and gradient images. The curves have the same shape after scaling the brightness by a factor $\zeta = 100$ while the optimal δ has shifted and been scaled accordingly by the same factor, indicating that a predictive model for δ should scale its output accordingly when exposed to scaled input data.

In the following text we will refer to these measurements on the samples $\mathbf{y}_s$ as *regressors*, and will denote them $\boldsymbol{\beta}_s$. Here, the s subscript is used to indicate that the regressor contains measurements from sample $\mathbf{y}_s$.

The set of all our samples is denoted Y. We denote the number of scenes in the data set as N, the number of spp configurations per scene as S, and let $N' = N \cdot S$ denote the total number of samples in the data set Y. In other words, Y consists of N' samples $\mathbf{y}_s$, $s \in \{1, 2, \ldots, N'\}$ where the samples come from N different scenes each rendered at S different spp.

4.3 Selecting the Parametric Model

The thought process behind selecting a model for the prediction of the Huber loss parameter δ is based on dimensionality analysis, a desire to achieve invariance to the brightness scale in the scenes, and on heuristic arguments about the feasibility of a certain type of model.

Brightness invariance in the predictive model is important. It is desirable for the model to generalize as much as possible and work across a wide array of brightness levels in the scenes. If our model would work well only for limited levels of brightness, the model would not be very useful in general.

When an image's brightness is multiplied by a scalar factor ζ, the relMSE remains unchanged when measured against the similarly scaled reference image and the optimal Huber parameter δ^* is shifted accordingly by the same factor ζ, as shown in Figure 4.3. This happens because the difference between the noisy and reference images is a factor ζ larger, and so scaling the Huber parameter by ζ will give the same optimization problem as before the multiplication. In order for the model we aim to build to make sense, it needs to behave in the same way and scale the output accordingly when an input is scaled.

Based on the above it is much more believable that a multiplicative model would be a good fit than a linear model which cannot be made to stand dimensionality analysis and simultaneously scale correctly as a function of brightness. The model chosen to predict the Huber loss parameter δ can thus be written as

$$H(\boldsymbol{\beta}_s;\boldsymbol{\theta}) = \exp\Big(\sum_u \boldsymbol{\theta}(u)\cdot\log\boldsymbol{\beta}_s(u)\Big). \tag{4.2}$$

Here, $\boldsymbol{\beta}_s$ is a vector containing the measured regressors of rendering scenario $\mathbf{y}_s$, $\boldsymbol{\theta}$ is a vector of model parameters corresponding to those features and u is the regressor index. In other words, $\boldsymbol{\theta}(u)$ indicates the response of the model to regressor $\boldsymbol{\beta}_s(u)$.

In the rest of the chapter, we will describe how the regressors $\boldsymbol{\beta}_s$ are measured when we generate data. This data consisting of the regressors will then be used to estimate or train the unknown model parameters $\boldsymbol{\theta}$. Then, for a rendering scenario $\mathbf{y}_s$ with index s, if we have a set of estimated model parameters $\boldsymbol{\theta}_c$ and regressors $\boldsymbol{\beta}_s$ computed from $\mathbf{y}_s$, this model will produce predictions for the Huber loss parameter as

$$\delta_s = H(\boldsymbol{\beta}_s;\boldsymbol{\theta}_c). \tag{4.3}$$

Later we will describe how model parameters are estimated in different rounds using slightly varying data sets, and we therefore use the subscript c on the model parameters $\boldsymbol{\theta}$ to distinguish between these.

4.4 Training the Model Parameters

With a choice of the parametric model H for the Huber loss parameter δ at hand, we turn to the problem of estimating the model parameters $\boldsymbol{\theta}$. We want our model to represent the connection between features in a given rendering scenario and a Huber loss parameter δ that would work well in said scenario. Given a set of observations that contain information about the scene features - the computed regressors $\boldsymbol{\beta}_s$ - and a measure of how good or bad different δ-values are, the goal is to train the model parameters so that the model learns these relationships and can reliably predict δ that would produce desirable results for future, not yet encountered rendering cases. We therefore gather data with which we associate a tailored cost function and set out to find model parameters $\boldsymbol{\theta}$ that produce cost-minimizing δ-values.

In the following we explain the kind of data that is gathered and computed during rendering, how it is passed as input to the model, how the cost function is designed and subsequently how the cost function is minimized as part of the parameter estimation procedure.

4.4.1 Data Generation

In order to train our parametric model H to serve as a predictor for δ, we need information about how features and metrics from different rendering scenarios relate to the performance of different δ-parameters used in Huber loss reconstruction. The necessary data generation is thus two-fold, we need to:

1) Render images of scenes and measure feature statistics relevant to the model

2) Reconstruct the images for a range of δ and measure the performance in relMSE by comparing the results to reference images

This gives us a way to tell the model whether or not it is doing a good job at predicting values for δ with its current parameters, and more importantly how the parameters need to change in order to come closer to an optimal solution.

We compute relMSE according to the following formula that is commonly used throughout rendering research and literature. The per-pixel relMSE ρ_j for each pixel j is computed and summed over the three color channels R, G and B of the image I, after which the average is taken across all M pixels in the image

$$\begin{aligned}\rho &= \frac{1}{M}\sum_{j=1}^{M}\rho_j \\ &= \frac{1}{M}\sum_{j=1}^{M}\frac{(I_j^R - I_{j,\mathrm{ref}}^R)^2 + (I_j^G - I_{j,\mathrm{ref}}^G)^2 + (I_j^B - I_{j,\mathrm{ref}}^B)^2}{(\frac{1}{3}(I_j^R + I_j^G + I_j^B))^2 + \epsilon},\end{aligned} \tag{4.4}$$

where we use $\epsilon = 0.001$. Per-pixel relMSE values are visualized in Figure 4.4, and the relMSE ρ for a rendered image is computed as the average of that relMSE-image.

During light transport simulation, the renderer is gathering sample measurements of color throughput that are used to form the primal and gradient estimates. The number of pixel samples n that is simulated is the samples per pixel. If n pixel samples are gathered for each pixel in every output buffer from the renderer such as primal and gradients, each measurement $z_t, t \in \{1, 2, \ldots, n\}$ is a random variable, assumed independent of and identically distributed to the others. The estimates of the primal and gradient images consist of an average value of the n pixel samples, the *sample mean* random variable

$$\bar{z} = \frac{1}{n}\sum_{t=1}^{n} z_t. \tag{4.5}$$

In the selected model we are thus interested in statistics of the sample mean random variable $\bar{z}$ related to the mean measurements of primal or gradient colors in the pixels; the value we get after taking the mean of our n samples z_t for each pixel.

Forming the Regressor Vector

The per-pixel statistics that are computed on-line during rendering is used to form the vector of regressors $\boldsymbol{\beta}_s$ with features from a scenario y_s. These regressor vectors are used as input along with model parameters $\boldsymbol{\theta}$ to the predictive model H in Equation 4.2 which if designed properly takes the information contained in the regressor features and produces a suitable parameter δ for Huber loss reconstruction.

Let p_j and g_j be primal and gradient colors for pixel j respectively. Naturally, p_j and g_j are formed by taking the mean of the n samples associated with pixel j as in Equation 4.5. The statistics are first computed on a per-pixel basis, and then the mean or variance over all the pixels in an image is measured and used as feature regressors in the model.

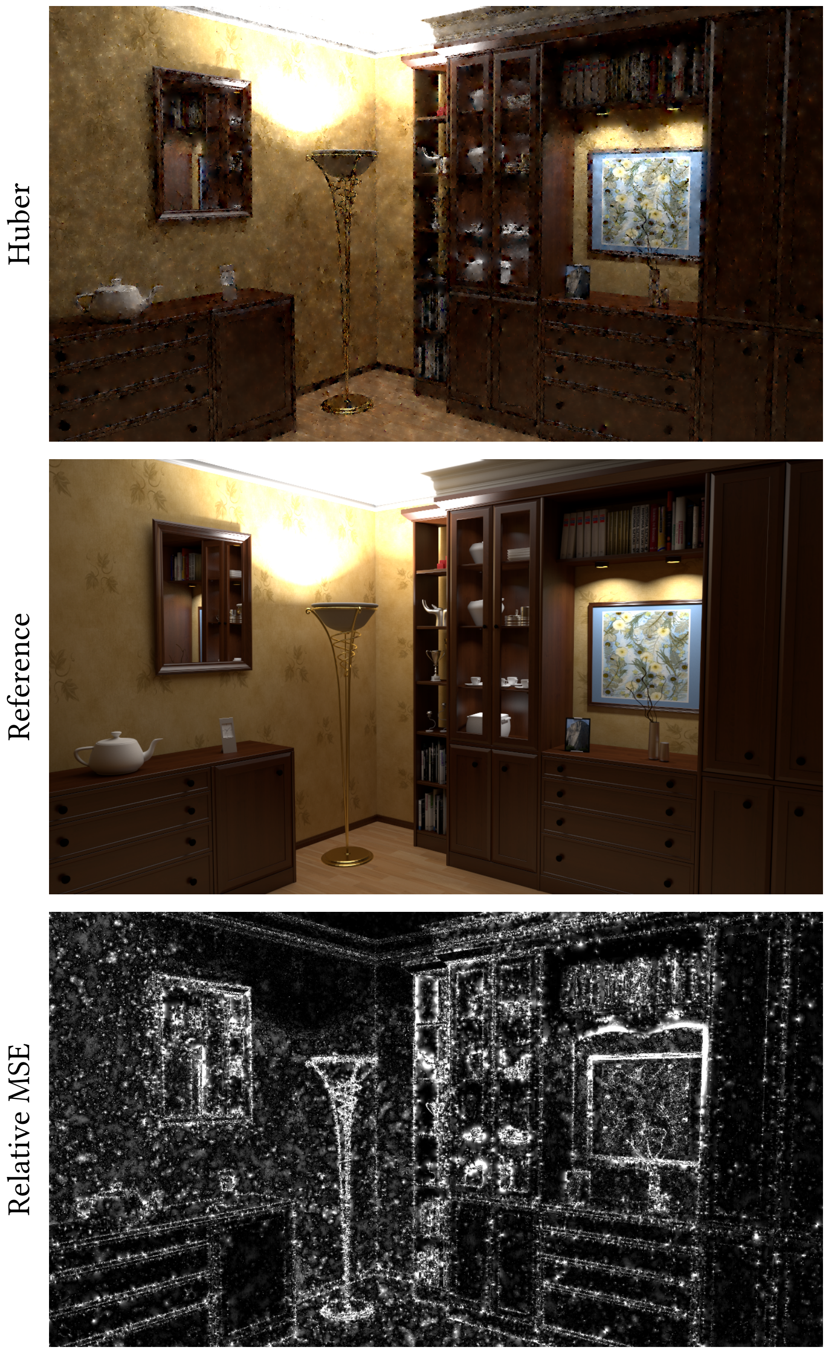


Figure 4.4: The performance of a reconstruction method is measured by its ability to produce results close to a reference image. *(Top)* 8spp image reconstructed using Huber loss. *(Middle)* Reference image. *(Bottom)* Relative MSE measured between the reconstructed image and the reference image, darker areas indicating a lower value.

For our model, we choose to gather metrics and feature statistics such as the variance, skewness and kurtosis of the samples that give indicative information about the complexity and behaviour of the light transport in a given rendering scenario. We refer the reader to statistics literature such as Mood (1950) for an introduction to central moments and other statistical measures.

The elements of the regressor vector are thus

$$\begin{aligned}
\boldsymbol{\beta}(1) &= C, \qquad C \in \mathbb{R} \\
\boldsymbol{\beta}(2) &= \mathrm{Mean}[\mathrm{Var}[p_j]] \\
\boldsymbol{\beta}(3) &= \mathrm{Mean}[\mathrm{Var}[g_j]] \\
\boldsymbol{\beta}(4) &= \mathrm{Var}[\mathrm{Var}[p_j]] \\
\boldsymbol{\beta}(5) &= \mathrm{Var}[\mathrm{Var}[g_j]] \\
\boldsymbol{\beta}(6) &= \mathrm{Mean}[p_j] \\
\boldsymbol{\beta}(7) &= \mathrm{Mean}[\mathrm{Skew}[p_j]] \\
\boldsymbol{\beta}(8) &= \mathrm{Mean}[\mathrm{Skew}[g_j]] \\
\boldsymbol{\beta}(9) &= \mathrm{Mean}[\mathrm{Kurt}[p_j]] \\
\boldsymbol{\beta}(10) &= \mathrm{Mean}[\mathrm{Kurt}[g_j]].
\end{aligned} \tag{4.6}$$

One such regressor vector is computed for each rendering scenario, in other words we have a vector $\boldsymbol{\beta}_s$ computed for each $\mathbf{y}_s$. In our study we compare different variants of the model H where each variant can incorporate any or all of the listed regressors and therefore some of the features in vector $\boldsymbol{\beta}$ may be excluded. If for instance it was found that the skewness regressor added no predictive power to the model, these elements of the feature vector $\boldsymbol{\beta}$ would not be used.

Regressor Formulas

To generate the regressor vector described in Equation 4.6 we have to compute standardized central moments for the sample mean random variable $\bar{z}$ during rendering in an on-line fashion. On-line formulas compute a running value for the entity of interest as opposed to storing all samples and performing computations post-simulation and are a necessity to circumvent storage limitations. Algorithms and formulas for the on-line computation of the variance of a random variable such as z_t have long been available (Chan, Golub, and LeVeque 1982, 1983). These algorithms have been extended to not only encompass variance calculation but abritrary order moments too (Pebay 2008; Terriberry 2008).

These formulas can compute central moments for a sample random variable z_t, while in our model we want to incorporate the aforementioned feature statistics of the sample mean random variable $\bar{z}$, the pixelwise primal estimates p_j or gradient estimates g_j. For this we derive arbitrary-order formulas in Appendix A. The derived formulas can be used to compute the variance, skewness and kurtosis of the sample mean random variable $\bar{z}$

which we list here for reference

$$\begin{aligned}\mathrm{Var}[\bar{z}] &= \bar{\mu}_2 = \frac{\mu_2}{n} \\ \mathrm{Skew}[\bar{z}] &= \bar{\gamma}_3 = \frac{n\mu_3}{(n\mu_2)^{3/2}} \\ \mathrm{Kurt}[\bar{z}] &= \bar{\gamma}_4 = \frac{n\mu_4}{(n\mu_2)^2},\end{aligned} \tag{4.7}$$

where μ_k is the k:th central moment of the individual z_t.

Renderer and Render Settings

All of the rendering was done using a custom version of the Mitsuba renderer (Jakob 2015). The reference images were rendered using the unmodified bidirectional path tracing implementation, and the images constituting our data set were rendered with an extended gradient-domain path tracing implementation, where the sampler was instrumented to compute the per-pixel second, third and fourth order central moments μ_2, μ_3 and μ_4 for the primal and gradient images in an on-line fashion, and store them in buffers along with the regular renderer output. The computed per-pixel central moments are then used in the formulas 4.7 to compute the statistics for $\bar{z}$. For this thesis, all of the data was generated using the uniform sampler and box filter in the settings of the renderers.

Data was generated for $N = 15$ different scenes and each scene was rendered $S = 8$ different spp setups; at 4, 8, 16, 32, 64, 128, 256 and 512 spp. Our data set thus consists of $N' = N \cdot S = 120$ rendering scenarios y_s, $s \in \{1, 2, \ldots, 120\}$. The scenes are a mix of commonly used scenes in rendering research and scenes depicting plausible real-life scenarios. Information about the scenes used, their origin and reference renders is displayed in Appendix C.

4.4.2 Model Configurations

The regressor $\boldsymbol{\beta}(6)$ in Equation 4.6 is the average brightness in the sampled primal image. This is included in the model parameters in order to achieve invariance to the brightness scale in the models we investigate. It is computed from the other parameters so that the δ predicted by the model has the correct unit. Explicitly, the response of this regressor - the model parameter $\boldsymbol{\theta}(6)$ - is calculated from the rest of the parameters as

$$\boldsymbol{\theta}(6) = 1 - 2\boldsymbol{\theta}(2) - 2\boldsymbol{\theta}(3) - 4\boldsymbol{\theta}(4) - 4\boldsymbol{\theta}(5). \tag{4.8}$$

Constraining the brightness parameter in this fashion is essentially the same as normalizing the rendered image's brightness, predicting δ for this normalized image and then multiplying the solution by the original brightness.

To decide which statistical features the final model should incorporate we wish to measure how the performance of the model is affected if a certain regressor is added or removed. It is also of interest to see how the model performs when exposed to reconstruction problems of different brightness if the model parameters are constrained to be invariant to brightness scale or if they are unconstrained.

The tested configurations of the model are

1: $\boldsymbol{\beta}(1), \boldsymbol{\beta}(2), \boldsymbol{\beta}(3), \boldsymbol{\beta}(4), \boldsymbol{\beta}(5), \boldsymbol{\beta}(6)$ *(constrained brightness parameter)*

2: $\boldsymbol{\beta}(1), \boldsymbol{\beta}(2), \boldsymbol{\beta}(3), \boldsymbol{\beta}(4), \boldsymbol{\beta}(5), \boldsymbol{\beta}(6)$ *(free brightness parameter)*

3: $\boldsymbol{\beta}(1), \boldsymbol{\beta}(2), \boldsymbol{\beta}(3), \boldsymbol{\beta}(4), \boldsymbol{\beta}(5)$

4: $\boldsymbol{\beta}(1), \boldsymbol{\beta}(2), \boldsymbol{\beta}(3), \boldsymbol{\beta}(4), \boldsymbol{\beta}(5), \boldsymbol{\beta}(6), \boldsymbol{\beta}(7), \boldsymbol{\beta}(8), \boldsymbol{\beta}(9), \boldsymbol{\beta}(10)$

5: $\boldsymbol{\beta}(1), \boldsymbol{\beta}(2), \boldsymbol{\beta}(3), \boldsymbol{\beta}(6)$

6: $\boldsymbol{\beta}(1), \boldsymbol{\beta}(4), \boldsymbol{\beta}(5), \boldsymbol{\beta}(6)$.

The rationale behind these model selections is to start with a model that uses basic statistical measures such as the mean and variance, and then make modifications by adding or removing higher order moments to see if the model performance is improved. We also want to see if the brightness constraint in Equation 4.8 helps with scale invariance. For our baseline, we therefore start with model 1 that incorporates regressors $\boldsymbol{\beta}(1)$-$\boldsymbol{\beta}(6)$.

In our investigation we perform modifications to the baseline model and analyze what happens to the performance. We investigate modifications to the brightness consideration of the model by either keeping the feature parameter unconstrained, or by dropping it entirely. We also consider adding complexity to the model by considering the skewness and kurtosis features $\boldsymbol{\beta}(7)$-$\boldsymbol{\beta}(10)$. Lastly, we investigate simplifications from the baseline by either dropping the variance of variance features $\boldsymbol{\beta}(4)$-$\boldsymbol{\beta}(5)$ or the mean variance features $\boldsymbol{\beta}(2)$-$\boldsymbol{\beta}(3)$.

4.4.3 Designing the Cost Function

In order to optimize the model parameters $\boldsymbol{\theta}$ we need to decide how to give a score that indicates the goodness of a set of parameter estimates. The optimizer will search the space of possible model parameters, and while doing this it needs a measure of how good the current parameter set being explored is in order to know where to search for improvements. Giving the optimizer this measure is done by choosing a cost function which we denote $J(\boldsymbol{\theta})$. This cost function assigns a measure of performance to a set of model parameters $\boldsymbol{\theta}$.

We write the cost function we have chosen as

$$J(\boldsymbol{\theta}) = \sum_{s} \varphi_s(H(\boldsymbol{\beta}_s; \boldsymbol{\theta})). \tag{4.9}$$

Here, H is our multiplicative model that generates δ, $\boldsymbol{\theta}$ are the model parameters that we want to optimize, $\boldsymbol{\beta}_s$ is the measured regressors for rendering scenario $\mathbf{y}_s$ and φ_s is a loss function associated with rendering scenario $\mathbf{y}_s$.

A flowchart of how this function is evaluated is shown in Figure 4.5. Given a set of model parameters $\boldsymbol{\theta}$, for each sample $\mathbf{y}_s$, we:

1) Take the measured statistics $\boldsymbol{\beta}_s$

2) Predict a Huber parameter $\delta_s = H(\boldsymbol{\beta}_s; \boldsymbol{\theta})$

$$\begin{tikzcd}
 & H(\boldsymbol{\beta}_1, \boldsymbol{\theta}_c) = \delta_1 \arrow[r] & \varphi_1(\delta_1) \arrow[ddr] & \\
 & H(\boldsymbol{\beta}_2, \boldsymbol{\theta}_c) = \delta_2 \arrow[r] & \varphi_2(\delta_2) \arrow[dr] & \\
\boldsymbol{\theta}_c, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m \arrow[uur] \arrow[ur] \arrow[dr] \arrow[dddr] & \vdots & \vdots & \sum_s \varphi_s(\delta_s) = J(\boldsymbol{\theta}_c) \\
 & H(\boldsymbol{\beta}_s, \boldsymbol{\theta}_c) = \delta_s \arrow[r] & \varphi_s(\delta_s) \arrow[ur] & \\
 & \vdots & \vdots & \\
 & H(\boldsymbol{\beta}_m, \boldsymbol{\theta}_c) = \delta_m \arrow[r] & \varphi_m(\delta_m) \arrow[uuur] &
\end{tikzcd}$$

Figure 4.5: The anatomy of the cost function $J(\boldsymbol{\theta})$. Given a training set T_c, the model $H(\boldsymbol{\beta}; \boldsymbol{\theta})$ is used to predict a Huber parameter δ_s for each sample $\mathbf{y}_s$ in the training set. Regressors $\boldsymbol{\beta}_s$ are passed to the model for each sample along with a set of model parameters $\boldsymbol{\theta}_c$ that we want to assess the performance of. The goodness of each predicted parameter is measured by a custom loss function φ_s for each sample. The sum of these loss functions constitute the total cost function that we want to minimize.

3) Evaluate some function $\varphi_s(\delta_s)$

after which all the evaluations are summed.

We have not yet said anything about the characteristics of φ_s. In order for it to be suitable for use in the learning process of the model parameters, the only criteria we have is that it should be differentiable and give a measure of how good a certain predicted δ_s is at reconstructing the output in scenario $\mathbf{y}_s$ generated with a gradient-domain rendering method.

We could choose a loss function such as L_2 or L_1 centered around the optimal parameter δ^* for all the φ_s, but looking at the measured relMSE curves such as in the left hand side of Figure 4.2, we see that it is worse (the relMSE is higher) to choose a δ that is too low, than it is to select a δ that is too high. In other words, the error from not selecting the optimal parameter δ^* is not symmetrical, and we want the learning process to take this into consideration.

Therefore, rather than selecting a regular symmetric loss function, we construct custom asymmetric loss functions for φ_s using the measured relMSE data through the following procedure, visualized in Figure 4.6:

a) The relMSE ρ is measured over a range of cut-off parameters δ for each rendering setup $\mathbf{y}_s$.

b) We normalize the δ-ρ curves so that the interpolated relMSE from all configurations are of similar weight and all samples $\mathbf{y}_s$ are considered equally important. We also perform logarithmic transformation.

c) We fit analytic functions to the measured data in logarithm space, on the left and right side of the optimum, to ensure that we have a differentiable extension to the estimated δ-ρ curves that does not flatten out completely. We fit a quadratic function for values less than the optimal cut-off δ^*, and a Charbonnier loss function for values larger than the optimum (Barron 2017; Charbonnier et al. 1994).

d) The functions are transformed back to linear space.

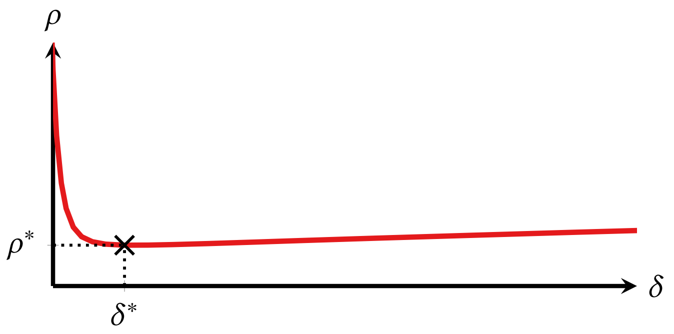
Fitting the component functions φ_s of the total cost function J in this way helps the learning process, since for an optimization scheme to work efficiently it is desirable to have a cost function that is decreasing towards the optimum regardless of what parameter values are explored in the process. One recurring theme throughout machine learning and neural network design is that the gradient of the cost function must be large and predictable enough to serve as a good guide for the learning algorithm. Functions that saturate and become very flat undermine this objective because they make the gradient become very small (Goodfellow, Bengio, and Courville 2016).

Parameters for the fitted functions are precomputed and stored for each sample with no added computational cost during model training.

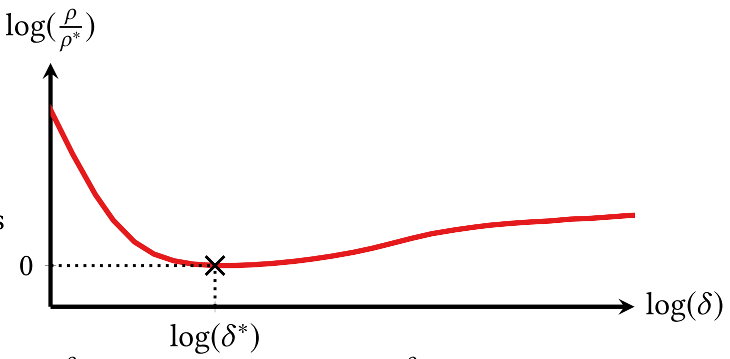
4.4.4 Finding Cost-Minimizing Model Parameters

With the cost function 4.9 selected the next step is to find parameters $\boldsymbol{\theta}$ for the predictive model H that minimizes $J(\boldsymbol{\theta})$. This is what is commonly known as training the model.

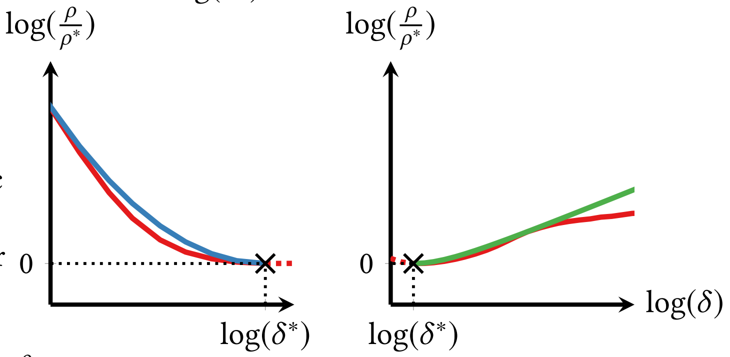
a) Measured δ-relMSE curve for a typical sample. Our model should predict δ that minimizes the summed relMSE over the training set.



b) We normalize the measured relMSE ρ by the optimal value ρ^* and take the logarithm in both axes. In these units the curve exhibits quadratic behaviour near the optimum.



c) We then fit a quadratic loss function for $\delta < \delta^*$ and a Charbonnier loss function for $\delta > \delta^*$ which is quadratic near the optimum and approaches a linear function for larger δ.



d) Exponentiating the fitted functions and subtracting by 1 gives us a loss function φ for the sample that is well-defined for all δ with a minimum value at δ^*.

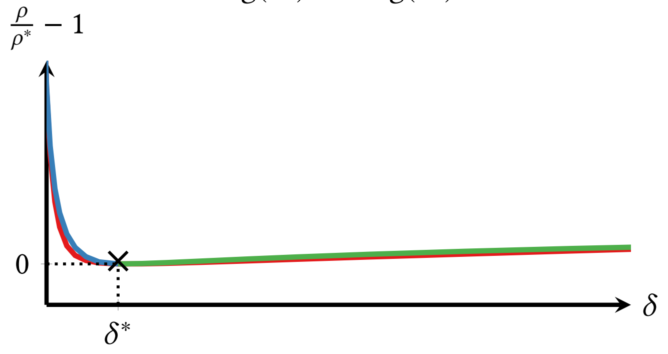


Figure 4.6: Procedure used to fit functions to the measured δ-ρ curves for each sample. These fitted cost functions φ are used in the total cost function minimized during the optimization and ensures that we can guide the optimization towards a minimum even if the model predicts δ outside of the measured interval.

We want to find parameters $\boldsymbol{\theta}$ for the model that produces high performing δ when used in the Huber loss reconstruction. After reconstructing an image using the δ produced by the trained model, the relMSE of the reconstructed image compared to the reference should be as small as possible, ideally at the measured optimum.

From the complete data set, the set of all our rendering scenarios

$$Y = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_s, \dots, \mathbf{y}_{N'}\}, \quad N' = 120, \tag{4.10}$$

model parameters are trained on a subset $T \subset Y$ known as the *training set*. The $J(\boldsymbol{\theta})$ to be minimized is a sum of relMSE values interpolated from the pre-computed δ-ρ curves associated with each sample in the training set T. The performance of the trained parameters on unseen rendering scenarios are tested on the remaining subset $V = Y \setminus T$ known as the *validation set*.

Samples from a scene are either included or excluded entirely in training to preserve the isolation of the validation set from the training procedure. To clarify, the case is never such that some of the spp configurations from a scene lies in the training set T while the others lie in the validation set V, a scene is always put in one or the other. We want to prevent information contained in the validation set to be available during training, which would happen if we were to hold out only a part of the configurations at different sample counts of a specific scene. If this was not done, testing against the validation set would be a less effective indicator of generalization since the information contained in the validation set would be partially included in training and the scenes in the validation set would no longer be considered unencountered to the model.

The cost function is minimized using the trust-region-reflective algorithm (Branch, Coleman, and Y. Li 1999; Byrd, Schnabel, and Shultz 1988; *MATLAB Optimization Toolbox* 2017), exploring 50 initial parameter values to reduce the probability of terminating at an undesirable local minimum.

4.5 Improving Model Stability

The parameters resulting from a round of optimization on a certain training set may vary and be heavily influenced by how the data was partitioned into training and validation sets, as there are many ways to perform this segmentation. The optimization may also be prone to overfitting. Ideally the training and validation sets should both be sufficiently large and diverse to represent a wide range of scenarios occuring in the rendering context. In our case, the size and diversity of the data is limited, partly constrained by the number of modelled scenes publicly available, and partly by the computational cost of producing renders of said scenes. These limitations are often present in general, and it is therefore not always feasible to produce entirely satisfactory data sets when they are not readily available from public sources.

With these considerations in mind, we aim to devise a model training schema that attempts to supress the sensitivity to the underlying samples. In Figure 4.7 we can see that the performance of the model parameters not only varies depending on the samples used for training, but that taking the mean of many parameters trained on slightly different training sets can yield vastly better results. This empirical observation suggests that an ensemble model may be suitable to achieve better robustness.

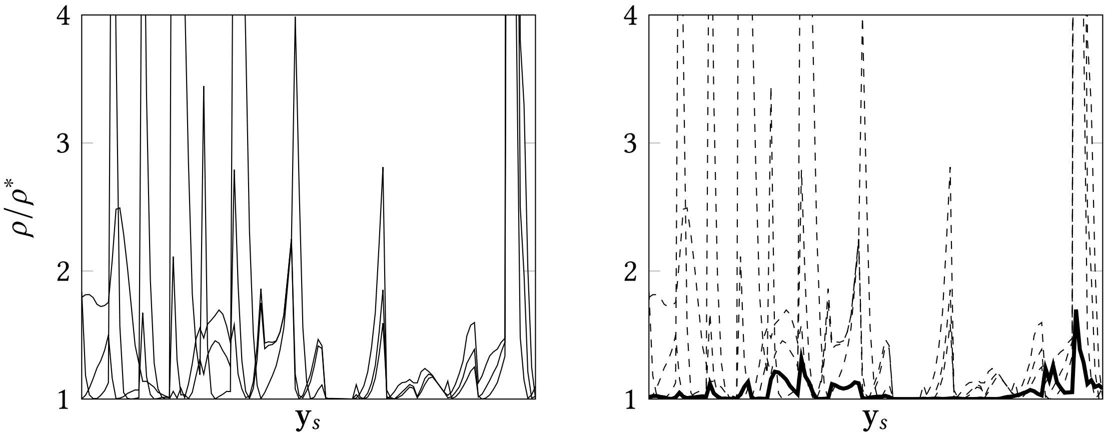


Figure 4.7: Performance of individual model parameters versus their mean. *(Left)* The parameters and their performance resulting from optimization on a single training set can depend heavily on the composition of samples in the set that was used for training. *(Right)* Taking the mean of many parameters that have been trained on different training sets results in better and more consistent performance. The improved performance of averaged parameters hints at the use of an ensemble modelling approach to improve stability.

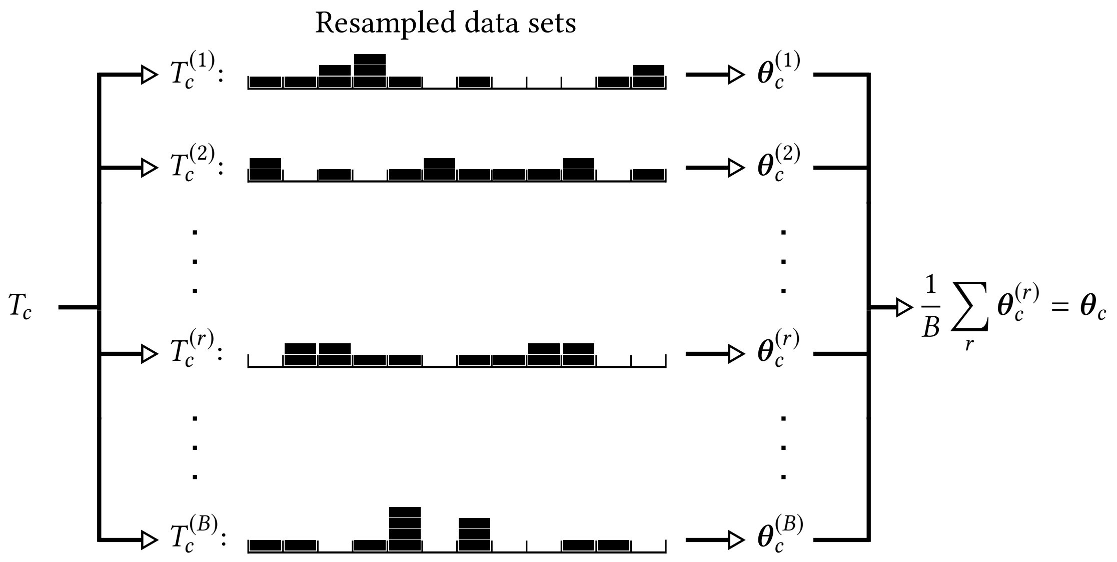


Figure 4.8: Bootstrap aggregating schema. For a given training set T_c, we form B bootstrap sample sets $\{T_c^{(1)}, T_c^{(2)}, \ldots, T_c^{(B)}\}$ by resampling with replacement from T_c. Training the model on a bootstrap sample set $T_c^{(r)}$ results in bootstrapped model parameters $\boldsymbol{\theta}_c^{(r)}$. The mean of the B bootstrapped parameters now generates the bagging model parameters $\boldsymbol{\theta}_c$.

4.5.1 Bootstrap Aggregating

A common ensemble method is bootstrap aggregating, often referred to as *bagging* (Breiman 1996). The method works by performing the training procedure several times on using data that has been bootstrapped from the training set, for instance by resampling from the training set with replacement. For more information about the standard bootstrapping technique, we refer to the work by Efron and Tibshirani (1994). The most suitable cases for bootstrap aggregating are when small changes in the training set can result in large changes in the model parameters, and the best enhancement can be seen when the model instances vary heavily from eachother (Barutçuoğlu and Alpaydın 2003).

Given a partitioning of the data into training and validation sets T_c and V_c, we train an ensemble model using bootstrap aggregating. Out of the samples from the 12 scenes in the training set T_c we form B bootstrapped sample sets $\{T_c^{(1)}, T_c^{(2)}, \ldots, T_c^{(B)}\}$ by sampling from the training set with replacement. In our work we select $B = 50$. Each of the bootstrapped sample sets are then used to train a set of model parameters $\boldsymbol{\theta}_c^{(r)}$, $1 \leq r \leq B$ and the bagging model parameters are formed by taking the arithmetic mean of these B bootstrapped model parameter sets

$$\boldsymbol{\theta}_c = \frac{1}{B}\sum_{r=1}^{B} \boldsymbol{\theta}_c^{(r)}. \tag{4.11}$$

Bootstrapping the original training set introduces variation in the data since the resulting bootstrapped sets will not be identical. The bagging procedure is visualized in Figure 4.8.

The model parameters resulting from the bagging procedure is less prone to overfitting the training data.

4.6 Estimating Generalization Performance

The performance of a set of model parameters on new data is ultimately the most important metric to describe the success of a model. Indeed, we are not interested in building a model able to reproduce the results we already have at hand in our training data, but rather to produce useful results in scenarios that fall within the same category but never have been encountered before. The validation error measured on a validation set held out during training is often used for this, but such an estimate can be sensitive to how the validation and training sets were selected. We are therefore interested in producing confidence intervals for our generalization estimate in order to receive some notion of how accurate this estimate is.

To this end, we improve our assessment of the generalization ability of a model by using cross-validation to produce confidence intervals. In cross-validation, the process of splitting the data Y into training set V and validation set T are repeated multiple times in what is known as cross-validation folds. In each round c of splitting, a set of model parameters $\boldsymbol{\theta}_c$ computed as in 4.11 is trained on the training set T_c using the bootstrap aggregating procedure outlined in Section 4.5.1, and the performance of the model with parameters $\boldsymbol{\theta}_c$ is measured in relMSE against the validation set V_c.

There are many different ways to perform cross-validation and the different methods can generally be categorized into exhaustive methods and non-exhaustive methods.

4.6.1 Exhaustive Cross-Validation

Exhaustive cross-validation methods train and measure validation error on all possible ways to partition the data into training and validation sets. One such method is Leave-q-out cross-validation (Celisse and Robin 2008). For a data set of N scenes, Leave-q-out cross-validation uses q scenes for the validation set V_c and the rest of the samples for the training set T_c. The cross-validation procedure then consists of C_q^N folds, where

$$C_q^N = \binom{N}{q}, \qquad 0 \leq q \leq N, \tag{4.12}$$

i.e. one fold for each of the ways to select q scenes out of N to be used for validation. Even for moderately large N and for values of q approaching $N/2$, Leave-q-out cross-validation can become computationally infeasible due to the large number permutations. A special case of Leave-q-out cross-validation is Leave-1-out cross-validation, frequently abbreviated as *LOOCV*.

4.6.2 Non-Exhaustive Cross-Validation

Non-exhaustive cross-validation methods attempt to approximate their exhaustive counterpart while trying to remain computationally feasible. Common examples are k-fold cross-validation (Zhang 1993) and Monte Carlo cross-validation (Xu and Liang 2001). k-fold cross-validation partitions the data into k equally sized folds, and performs training and testing in k rounds. In each round, one of the k folds is selected as validation set and the remaining $k - 1$ sets are used to form the training set. When $k = N$, k-fold cross-validation becomes identical to LOOCV. Studies suggest that k-fold cross-validation may outperform LOOCV (Kohavi 1995). Monte Carlo cross-validation randomly partitions the data into training and validation set. An advantage of k-fold cross-validation over Monte Carlo cross-validation is that all samples are guaranteed to be used both for training and validation, and that each sample is used for validation exactly once. An advantage of Monte Carlo cross-validation is that the relative size of the training and validation sets is not dependent of the number of cross-validation folds.

Given the relatively small size of our data set by modern machine learning standards, using an exhaustive approach remains computationally feasible and an exhaustive Leave-q-out cross-validation approach is therefore chosen to assess validation performance. We choose $q = 3$ so that in each round of splitting, 20% of the total number of scenes from the data set Y is used to form the validation set V_c and the remaining 80% is used to form the training set T_c. There is no overlap between the training and validation sets and together they cover the total data set, i.e. we have

$$\begin{aligned} &T_c \cup V_c = X, \\ &T_c \cap V_c = \varnothing, \qquad \forall c \in \{1, 2, \dots, C_q^N\}. \end{aligned} \tag{4.13}$$

An ensemble set of model parameters $\boldsymbol{\theta}_c$ is trained using the bootstrap aggregating schema outlined in Figure 4.8 for each of the C_q^N ways of splitting our data set into 80% training and 20% validation samples.

With $q = 3$ and $N = 15$, using this approach will produce $C_{q-1}^{N-1} = 91$ values of validation error for each sample. A histogram representing the distribution of these validation error values for one sample is shown on the right hand side of Figure 4.9. These histograms can be used to form confidence intervals of the validation error resulting from using model parameters trained by our procedure to predict δ-values for each sample point. The left hand side of Figure 4.9 shows a plot of the average validation error together with one standard deviation confidence intervals produced by our procedure. These asymmetric cross-validation intervals are formed by log-transforming the histograms, computing the symmetric one standard deviation confidence intervals in logarithm space, and exponentiating them back to linear space.

4.7 Final Model Training

The steps taken so far has been part of a procedure to compare different models and assess their performance. During training we hold out portions of our data set to use as validation, to estimate how the model would perform if exposed to new data. When a model and training method is selected however, we naturally want to use all the data available to us. To form the final model parameters, we can take the trained ensemble model parameters from the different cross-validation rounds and use an approach known

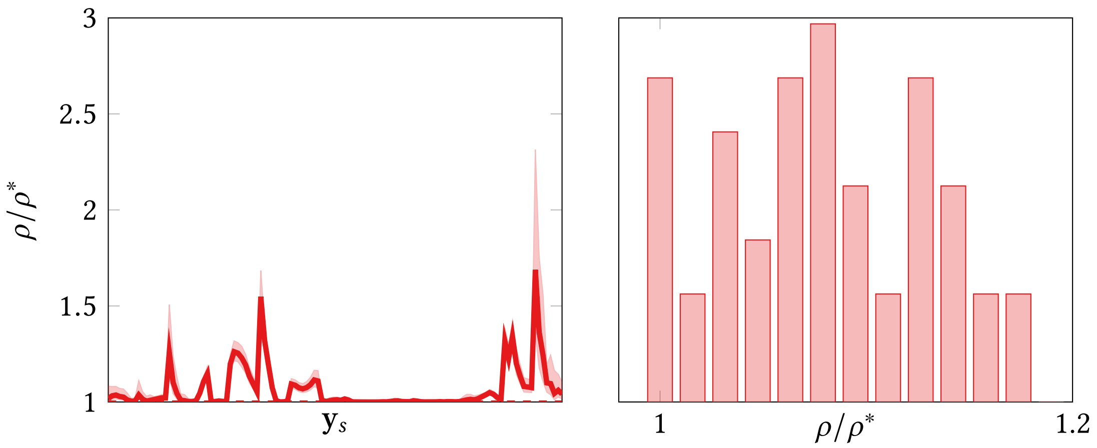


Figure 4.9: Cross-validation estimation of bootstrap aggregated model parameter performance. The cross-validation routine performs boostrap aggregating on model parameters trained on many training sets T_c and estimates the model's ability to generalize by measuring the validation error against the corresponding validation set V_c. This generates multiple values of validation error for each sample $\mathbf{y}_s \in Y$. *(Left)* Mean validation error and one standard deviation confidence interval computed from cross-validation results. *(Right)* Distribution of validation error from different cross-validation rounds for a single sample.

as cross-validation aggregating (Barrow and Crone 2013). The final parameter estimate is written as an aggregate of all bagged parameters from the cross-validation rounds.

$$\boldsymbol{\theta} = \frac{1}{C_q^N} \sum_{c=1}^{C_q^N} \boldsymbol{\theta}_c = \frac{1}{C_q^N} \sum_{c=1}^{C_q^N} \frac{1}{B} \sum_{r=1}^{B} \boldsymbol{\theta}_c^{(r)} = \frac{1}{B \cdot C_q^N} \sum_{c=1}^{C_q^N} \sum_{r=1}^{B} \boldsymbol{\theta}_c^{(r)} \tag{4.14}$$

5 Results

The focus is now turned to presenting the results of the framework we have devised. The methods depicted and motivated in Chapter 4 are used to train a predictive model H (equation 4.2) for the Huber loss parameter δ and the model is investigated and benchmarked here. Different variants of the model, each taking into account a different number of scene feature statistics as regressors, are trained and their performance is compared. We also investigate to which extent our goal of having a model invariant to brightness scale is achieved. Additionally, the bias introduced by reconstruction using Huber loss is estimated and this estimation is portrayed and compared against the state of the art. We finally present the model parameters $\boldsymbol{\theta}$ and their distributions from different cross-validation rounds, of which we take the arithmetic mean to form the final parameter values.

In the coming sections we will present results in the form of relMSE plots across all of our scenes. The figures contain a plot of normalized relMSE on the vertical axis, and rendering scenarios on the horizontal axis. The relMSE is normalized by the optimal relMSE measured using Huber loss reconstruction; the relMSE achieved when using the optimal δ parameter. Each point on the horizontal axis is a usual rendering scenario $\mathbf{y}_s$ in the form a scene-spp configuration, such as Crytek Sponza-64spp, and the plot will show the normalized relMSE that was measured when this scene-spp configuration was used as test data. The rendering scenarios on the horizontal axis is grouped by scene and ordered within the scene group according to increasing spp count. The results are always presented through validation performance, to assess generalization performance.

5.1 Effect of Regressors on Model Performance

The models labeled 1-6 listed in Section 4.4.2 each incorporate a different composition of regressors and we wish to explore the most suitable model configuration. Out of the regressors we have computed during the data generation, we wish to find a which of these to take into consideration in the model to achieve high performance and stability, and which of the regressors to leave out of the model. We therefore start off with the baseline model that uses configuration 1 and set out to investigate how modifications to this regressor setup in different regards affects the performance. For instance, we are interested in determining if equal performance can be achieved with a less complex model by removing some regressors from model consideration, or if higher performance is possible by incorporating more regressors in a model of higher complexity than the baseline.

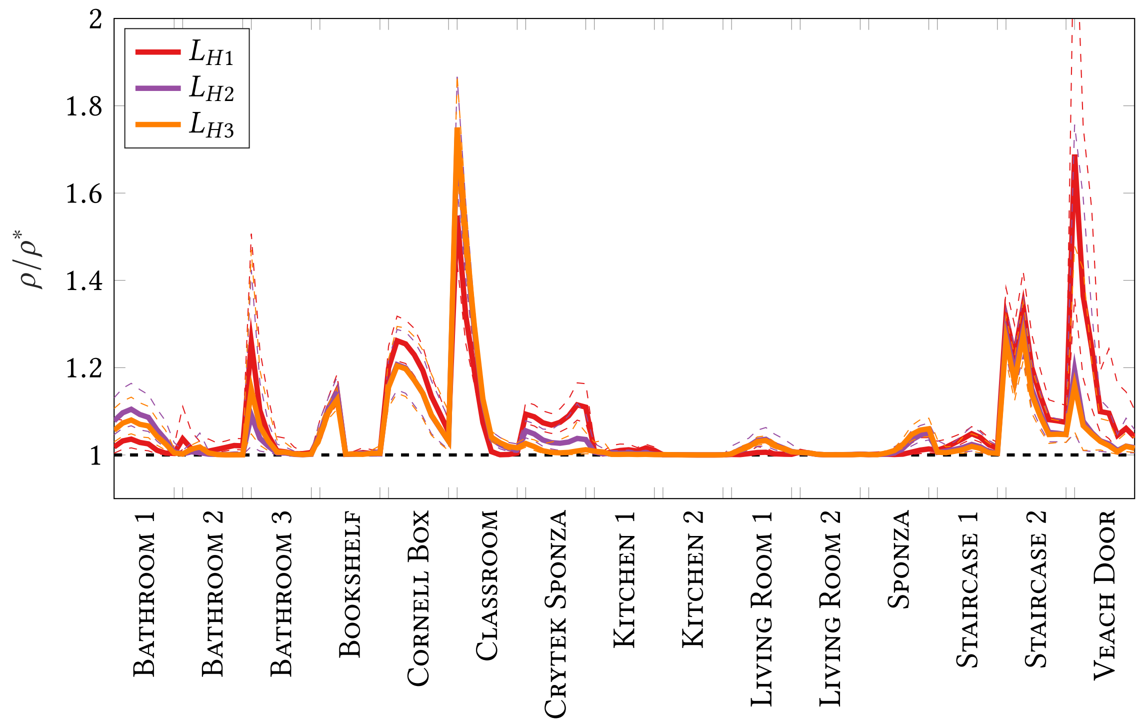


Figure 5.1: Effect of brightness regressor on model performance. The baseline model configuration 1 is compared against model configuration 2 where the constraint on the brightness parameter is removed and against model configuration 3 where the brightness regressor is excluded from the model. There is no significant difference between the configurations while configuration 1 ensures correct unit in the output δ.

5.1.1 Altering the Brightness Constraint

First we investigate the effect of the altering how the brightness regressor $\boldsymbol{\beta}(6)$ is considered in the models. The baseline configuration 1 is compared to configuration 2 where the constraint on the brightness parameter is removed, and configuration 3 where the brightness regressor is excluded entirely. Figure 5.1 shows relMSE plots of the different configurations with cross-validation confidence intervals of one standard deviation. We would like to keep the brightness regressor and its constraint to ensure unit correctness in the output, and therefore investigate the impact of leaving either out.

From these plots we can see that for most scenes the difference between the configurations is not significant enough to give preference to one over any other, and the overall performance is mostly unaffected by the two modifications to the parameter configuration of the baseline model.

5.1.2 Including Skewness and Kurtosis Regressors

Next we want to investigate whether adding complexity to the model in the form of incorporating more regressors can improve its performance. The baseline model is thus

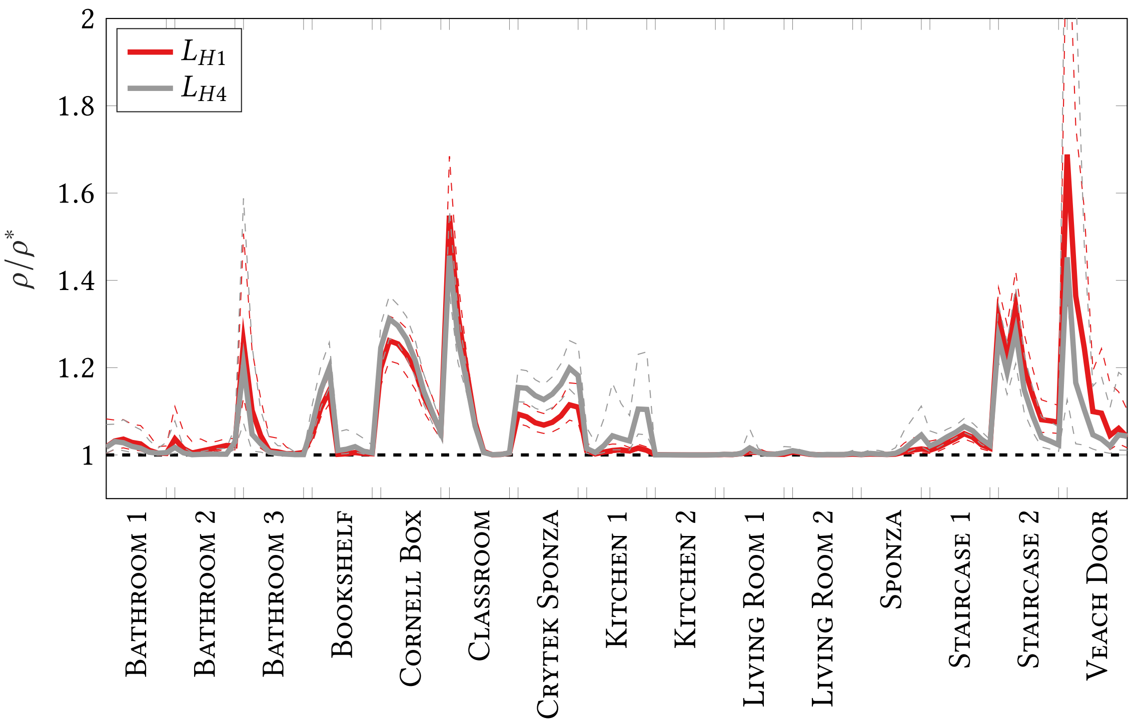


Figure 5.2: Effect of skewness and kurtosis regressors on model performance. The baseline model configuration 1 is extended in configuration 4 with the addition of the skewness and kurtosis regressors. The difference in model performance is not significant enough to warrant the added complexity.

modified to include the higher order skewness and kurtosis regressors in Equation 4.6. The model is benchmarked with and without this modification, i.e. we compare model configurations 1 and 4, and the performance results is plotted in Figure 5.2.

The results show that there is not a significant increase in predictive power to be gained by adding the regressors for skewness and kurtosis, and we therefore exclude them from the final model seeing as they also require additional computations during sampling that generally not performed in existing renderers.

5.1.3 Excluding Variance Regressors

Lastly, we are interested in investigating what the effects of simplifying the model are on its performance. This investgation is done by excluding either the mean variance regressors or the variance of variance regressors from the model and comparing against the baseline. The results are plotted in Figure 5.3.

We see that for some scenes the model performance is reduced to an extent which is not entirely dismissable compared to the baseline, giving an indication that the variance in the pixels are an important measure when deciding how to treat outliers in the sampled gradients through the Huber loss cut-off parameter.

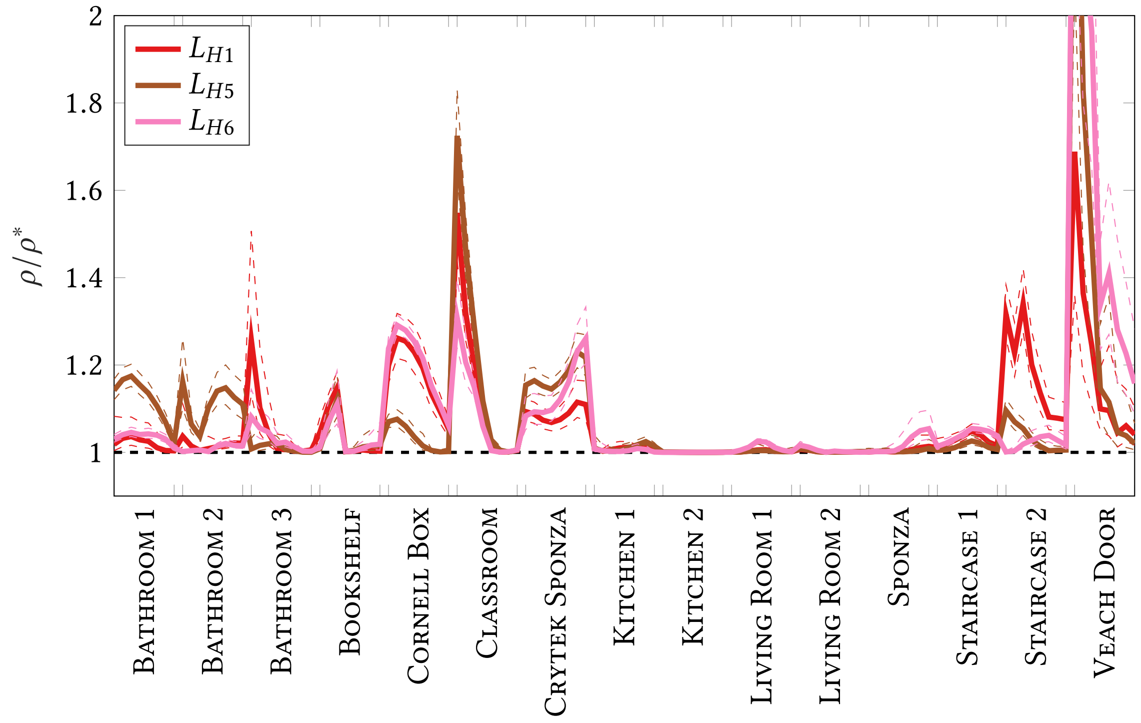


Figure 5.3: Effect of variance regressors on model performance. The baseline model configuration 1 is simplified in configuration 5 by removing the variance of variance regressors and in configuration 6 by removing the mean variance regressors.

5.1.4 Invariance to Brightness Scale

In addition to simply testing how the brightness parameter and its dimensional constraint affects model performance in terms of relMSE, it is interesting to see what its impact on scalability across scenes of vastly varying mean brightness has. To test this we measure the performance of model configurations 1, 2 and 3 on the regular scene set but also on the same set where all the data from the renderer has been brightened by a factor of 1000. The comparison is shown with relMSE measurements in Figure 5.4.

As previously stated, a model that is invariant to brightness scale should produce δ that results in the same relMSE regardless of the absolute brightness value in the rendering scenario. If the input is scaled, the output parameter from the model should scale accordingly and perform equally well as before the input was scaled. We see that this is clearly the case for the baseline model configuration that incorporates the brightness regressor and has its parameter constrained to ensure the correct unit in the output, whereas the other configurations fail to preserve the model performance under the scaling operation.

5.2 Selecting the Model Configuration

Based on the performance and characteristics observed while testing the different model configurations, it is not difficult to come to the conclusion that the baseline model using configuration 1 is to be prefered. The model incoporates the primal brightness regressor to ensure unit correctness in the output and as demonstrated is able to maintain constant performance when the brightness of the input is modified. In addition to this, as we have investigated, adding complexity to the model by including skewness and kurtosis regressors did not lead to a significant improvement in the performance of the predictions. Simplifications to the model by exluding variance regressors on the other hand did hurt the performance to a degree. We can thus conclude that out of the configurations we have tested, the baseline model is to be prefered.

5.3 Bias Reduction

Gradient-domain path tracing using L_1 reconstruction is efficient in suppressing outliers in the sampled gradients that lead to dipole shaped artifacts when using L_2 reconstruction. This benefit with L_1 reconstruction comes at a cost of introducing bias in the reconstructed image. We have claimed that Huber loss reconstruction should introduce less bias than L_1 and this claim will now be supported empirically.

We compare esimates of the bias introduced by Huber and L_1 reconstruction and inspect how close they come to L_2 performance. To estimate the bias introduced from the use of a specific loss function in the reconstruction, we render 100 images of the Kitchen 1 scene, reconstruct each image using the L_1 and Huber loss functions and then take the mean of the 100 rendered images in both cases. Comparing these averaged images against the reference image then gives an estimate of the error resulting from statistical bias in the reconstruction.

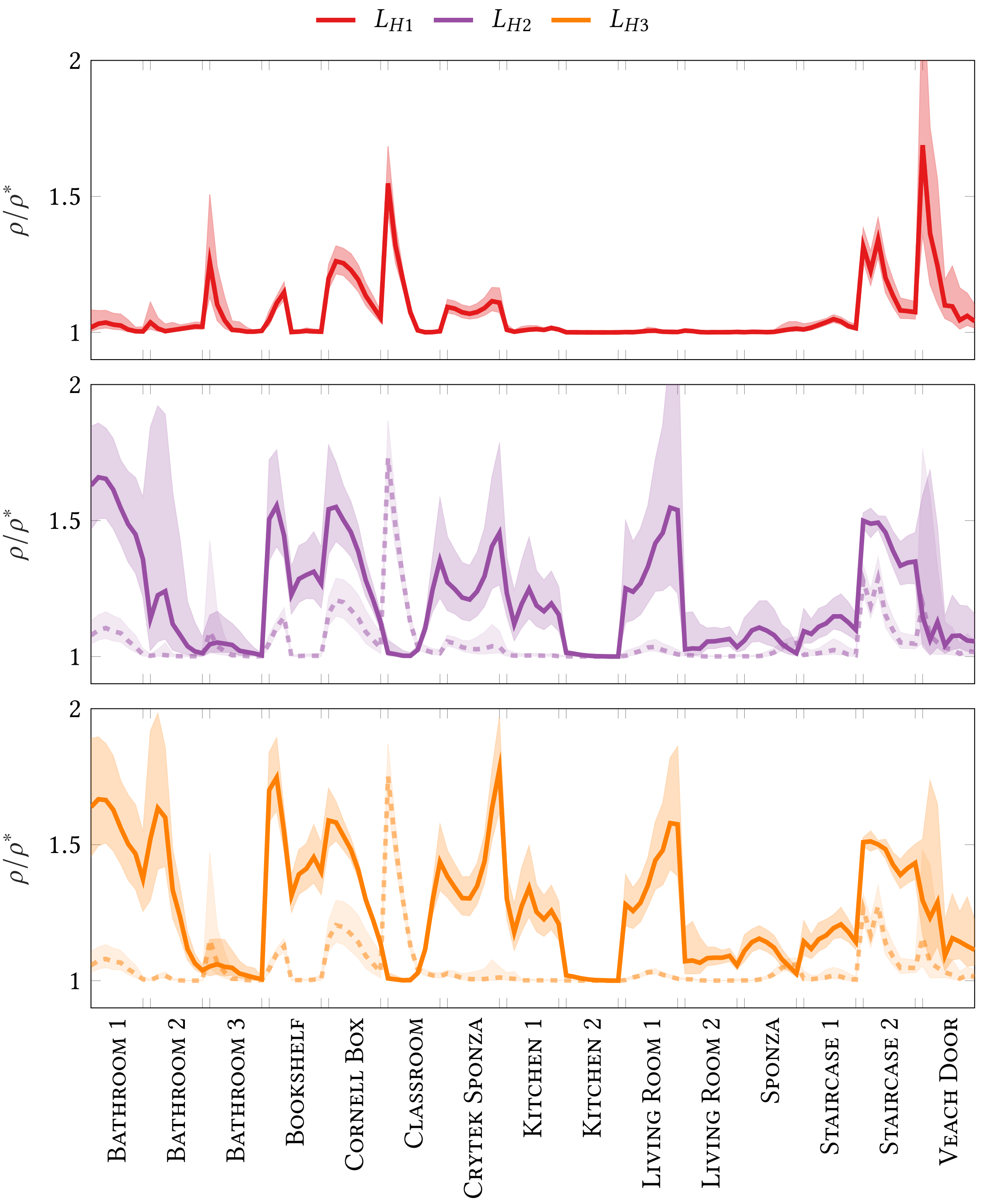


Figure 5.4: Behaviour of model configurations 1, 2 and 3 when used on reconstruction problems whose brightness has been scaled up. The dashed lines show performance on unscaled data. The baseline model configuration 1 produces output that scales in accordance with the underlying input, and the performance is identical in both cases. On the other hand, configuration 2 that removes the constraint on the brightness parameter and configuration 3 that excludes the parameter entirely fails to scale well and behaves unpredictably.

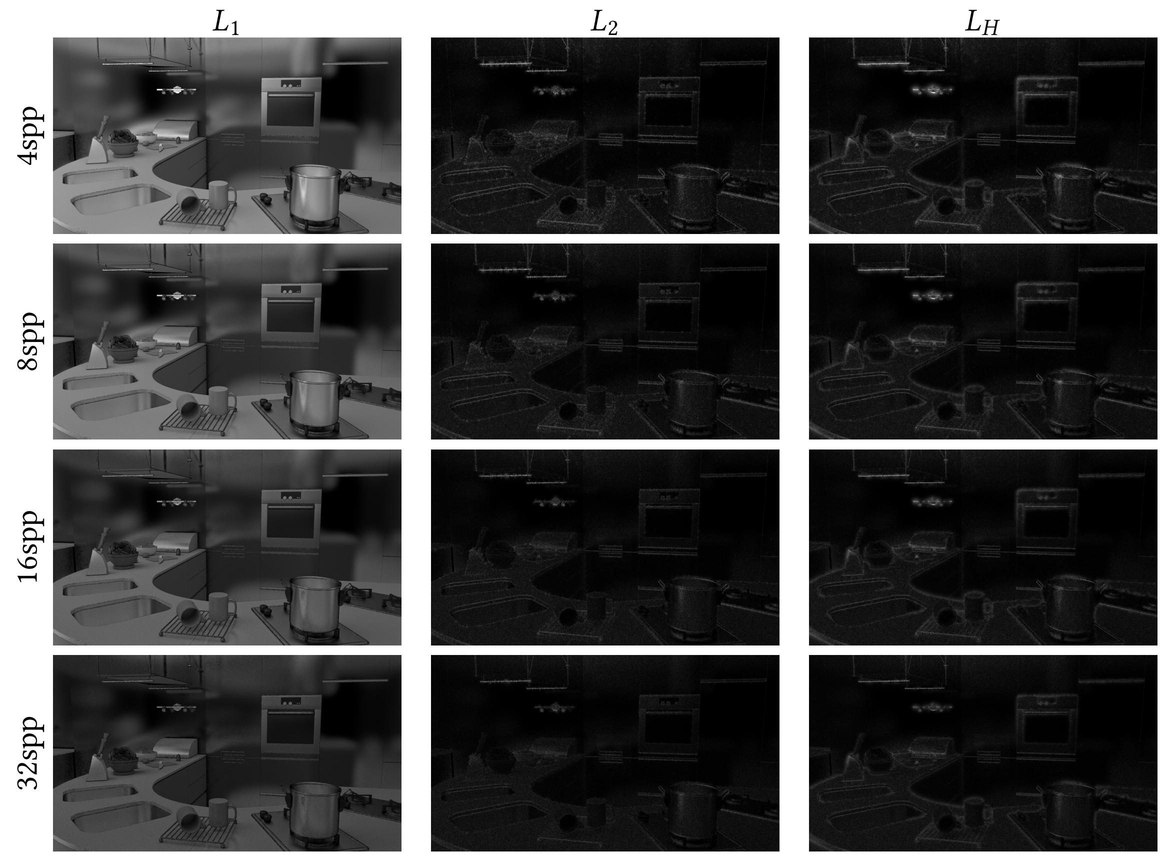


Figure 5.5: Estimation of the bias introduced by L_1, L_2 and Huber loss reconstruction. An unbiased, fully converged image would appear completely black. The Huber loss reconstruction comes considerably closer to the L_2 solution than the biased L_1 reconstruction, indicating an improvement over L_1 from a bias perspective.

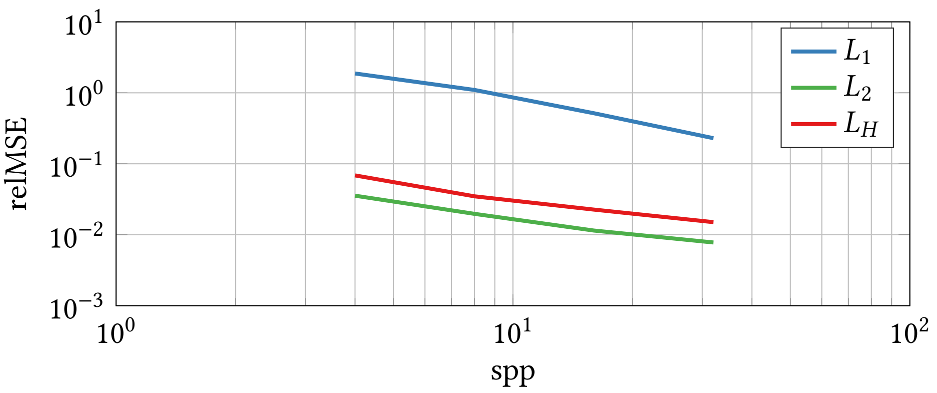


Figure 5.6: RelMSE plot of the estimated bias introduced during reconstruction. The bias from L_1 reconstruction is an order of magnitude larger in terms of relMSE than the bias from Huber reconstruction which behaves closer to L_2 reconstruction in this regard.

The images for visual inspection are shown in Figure 5.5. Here, a completely unbiased and converged image would appear completely black. It should be noted that the fact that the image of the L_2 estimate is not completely black and shows a difference from the reference image is due to unconverged noise and not bias, a consequence of the low spp count in the individual renders that are averaged. It is evident from these images that the bias introduced by L_1 reconstruction is of higher magnitude than that introduced through Huber loss reconstruction.

In addition to a visual inspection of the magnitude of the bias, a relMSE plot is shown in Figure 5.6 from which we can conclude and reiterate that Huber loss reconstruction comes significantly closer to L_2 reconstruction in terms of the bias introduced.

5.4 Model Parameters

The exhaustive cross-validation schema that was used to estimate confidence intervals for the performance of our model that we described in Section 4.6 results in multiple bagged model parameters. We show histogram approximations of the parameter distributions in Figure 5.7 and compute the final model parameters using Equation 5.1. The final model parameters are formed by taking the mean of the parameter distributions. They are plotted as the red vertical lines in the histograms and are listed below.

$$\begin{aligned}
\boldsymbol{\theta}(1) &= -2.58 \\
\boldsymbol{\theta}(2) &= 1.04 \\
\boldsymbol{\theta}(3) &= -0.86 \\
\boldsymbol{\theta}(4) &= -0.29 \\
\boldsymbol{\theta}(5) &= 0.53 \\
\boldsymbol{\theta}(6) &= -0.33
\end{aligned} \tag{5.1}$$

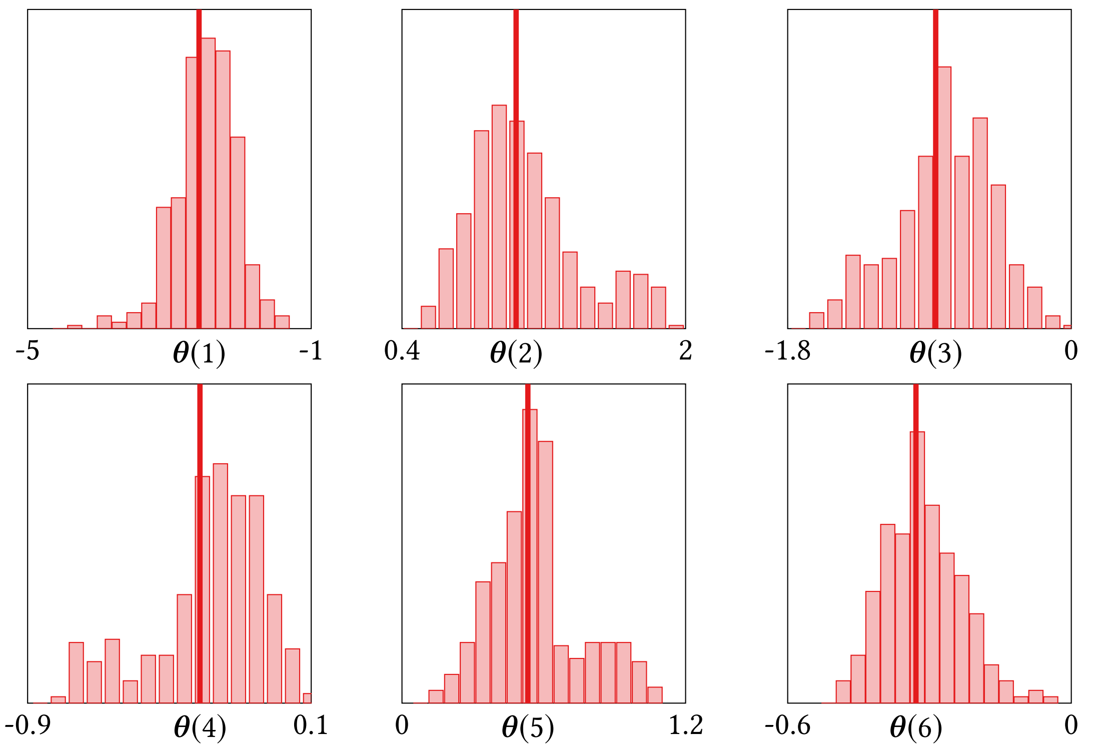


Figure 5.7: Histogram distributions of bagged model parameters from all the rounds of the exhaustive cross-validation schema. The final model parameters are formed by taking the mean of the histograms, the values at the red vertical lines.

5.5 Final Model Performance

The most important result to present of all is that of the trained Huber loss model performance versus the standard L_1 and L_2 reconstruction methods. Based on the previous investigations, we use the baseline configuration as our final model and proceed to show measured and visual results when the model is used to predict δ for Huber loss reconstruction.

5.5.1 Measured Performance

In Figure 5.8 we present the performance of the trained final model using the baseline configuration versus L_1 and L_2 from a relMSE point of view, with cross-validation confidence intervals computed as described in Section 4.6.

It is apparent that Huber loss reconstruction with a cut-off parameter δ predicted from the trained model produces reconstructions closer to the reference images in terms of relMSE on average. There are cases such as in the Classroom scene however where the L_1 reconstruction performs better than the measured theoretical optimum of reconstruction with Huber loss, but their performance seems to come closer at higher spp where there is less noise in the input to be outright suppressed in the reconstruction by L_1 loss.

Note also that for most scenes, the computed relMSE from the predicted model is relatively stable over cross-validation rounds. In other words, the computed standard deviation is low and consequently the estimated confidence intervals narrow. This gives strength to the approach we have employed in that it shows little sensitivity to the underlying samples used to train the model parameters. With that said, the performance in the Classroom, Staircase 2 and Veach Door scenes is less stable, and these are dominated by many specular interactions and occluded light sources. For these scenes, the confidence intervals are wider which means that the model performance was more sensitive to the composition of the training set. If the training set is composed mainly of scenes that do not represent the same sort of complex lighting scenarios, then the resulting model parameters may perform poorly. To improve in these scenes it would be benificial to perform the training using a data set that contains many more scenes with the described complex properties. If this was done it is possible that incorporating other regressors would give a more significant improvement to performance.

5.5.2 Visual Performance

In Figures 5.9-5.12 we show visual results of our trained model's performance for most of our scenes. The results of the reconstruction using Huber loss is shown next to the corresponding images reconstructed using L_1 and L_2 loss.

As we also showed previously, Huber loss introduces considerably less bias than L_1 loss. It is also apparent across most scenes that Huber loss suppresses the dipole artifacts resulting from outliers when compared to L_2 loss. Where L_1 successfully suppresses outliers at the cost of introducing bias, Huber loss suppresses outliers but inherits the uneven blotches in the images from L_2 loss instead of introducing bias. As the sample count is increased, the three reconstruction methods naturally come closer to eachother;

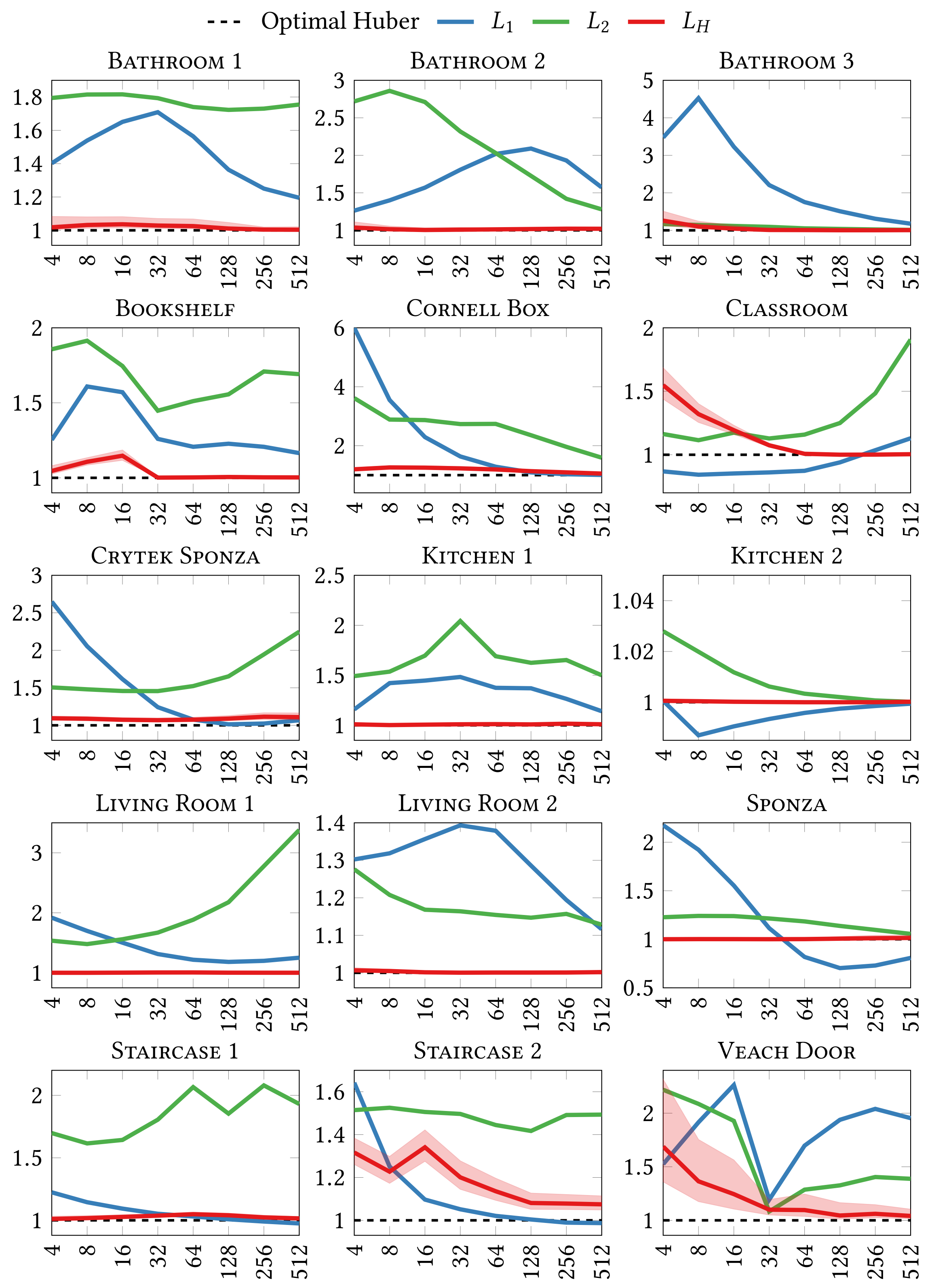


Figure 5.8: Performance of our trained Huber model versus L_1 and L_2. The vertical axis shows the error as a multiple of the optimal Huber parameter performance for that scene. The trained model outperforms L_1 and L_2 in the majority of the scenes, and often hits near optimal Huber performance.

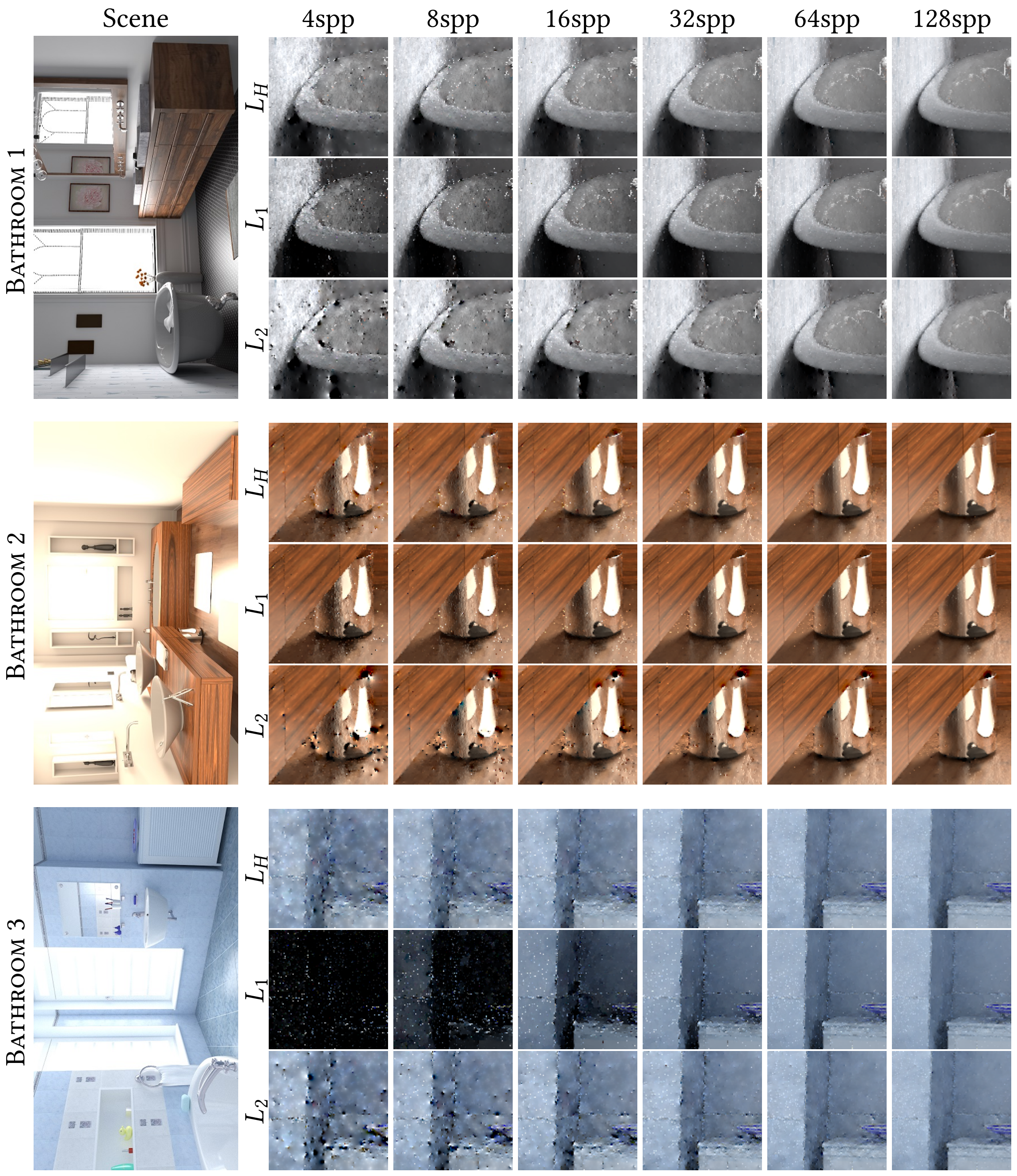


Figure 5.9: Visual model performance versus L_1 and L_2 for the BATHROOM 1, BATHROOM 2 and BATHROOM 3 scenes.

the bias in L_1 loss is diminished and the spots in Huber loss gradually vanishes, and it is not obvious which of these side-effects has less negative impact on the visual perception of the image quality.

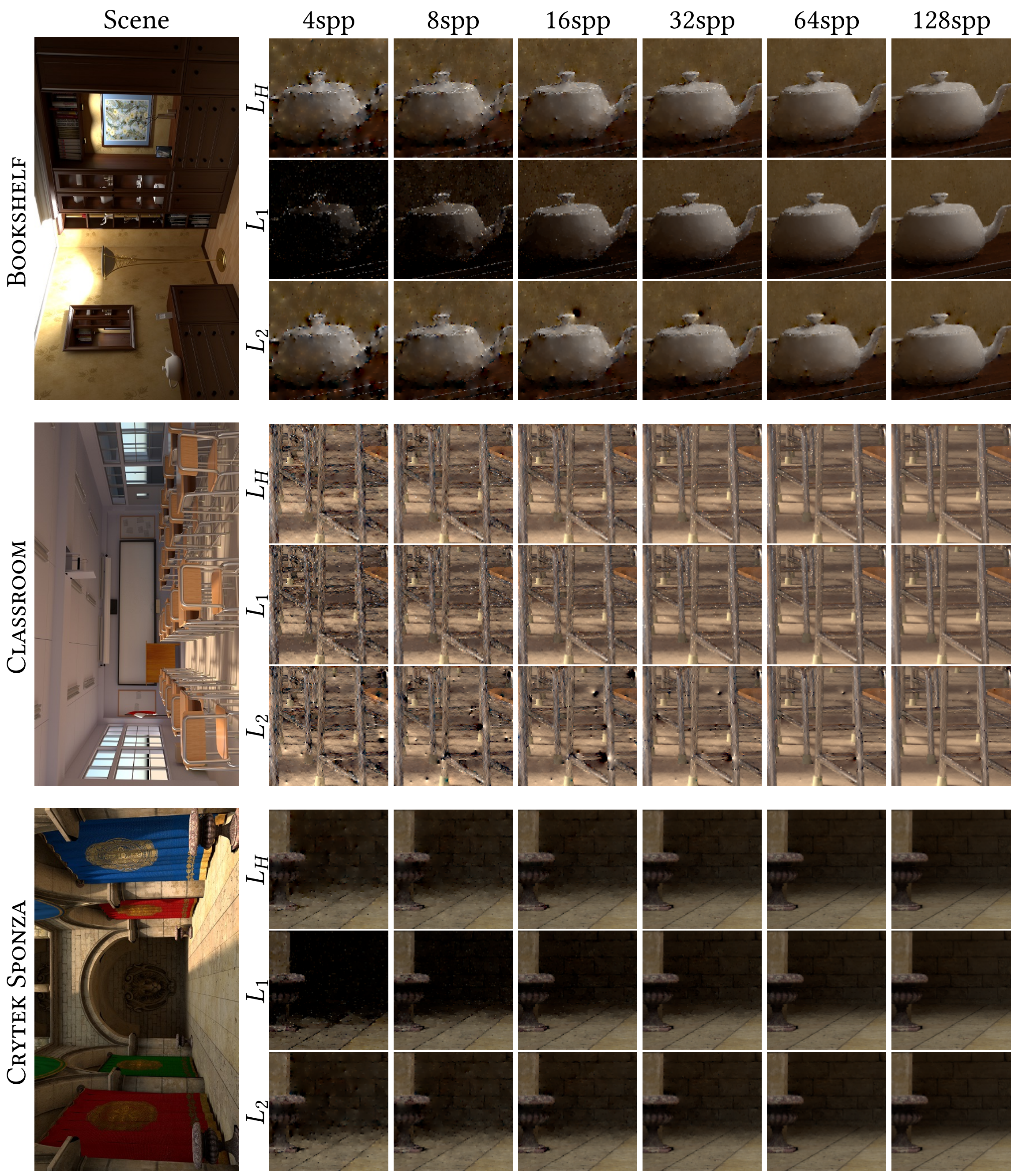


Figure 5.10: Visual model performance versus L_1 and L_2 for the BOOKSHELF, CLASSROOM and CRYTEK SPONZA scenes.

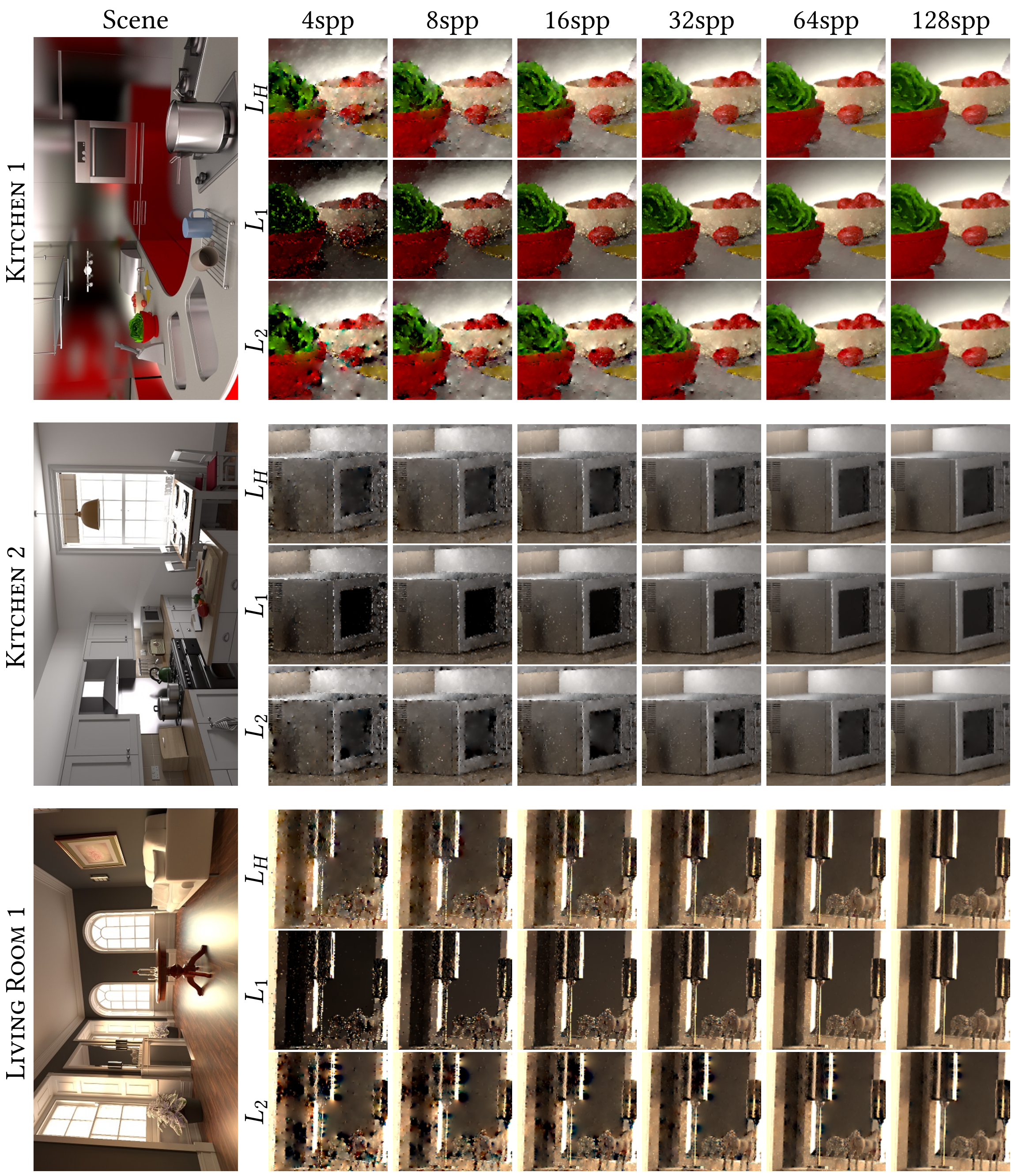


Figure 5.11: Visual model performance versus L_1 and L_2 for the Kitchen 1, Kitchen 2 and Living Room 1 scenes.

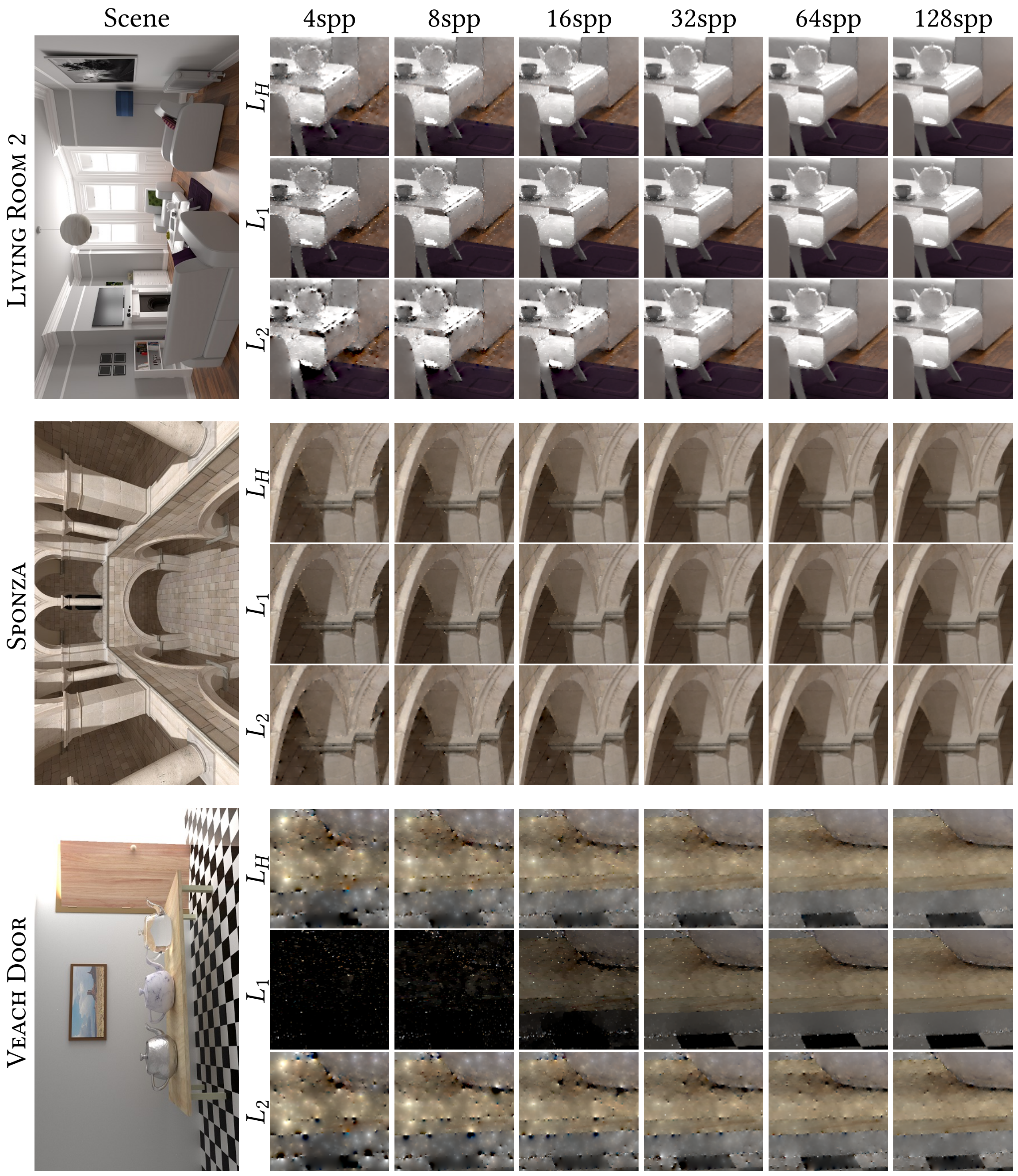


Figure 5.12: Visual model performance versus L_1 and L_2 for the LIVING ROOM 2, SPONZA and VEACH DOOR scenes.

6 Conclusion

In this thesis, we have presented an introduction to the underlying theory of direct gradient estimation using Monte Carlo sampling and have investigated a new reconstruction method for gradient-domain path tracing.

To motivate this research, we reviewed the reconstruction methods that are used traditionally and identified flaws with each method. We identify that the use of Huber loss instead of L_1 or L_2 loss for the gradient part in the reconstruction step can yield better results, if the cut-off parameter for the Huber loss function is chosen properly. Additionally, we note that the optimal choice of this parameter varies depending on which scene is being reconstructed, and that no choice of parameter works well across all scenes.

We proceeded to develop a method to select the cut-off parameter δ for a rendering scenario automatically. Specifically, we trained a multiplicative model that takes statistics computed during rendering into account to predict a parameter that would work well in the given scenario. The model generally performs better in terms of relMSE than previous methods across our set of scenes, and is invariant to the absolute brightness of the scene being rendered. The method is easy to implement as an extension to existing gradient-domain renderers.

Visually we have shown that the results produced by Huber loss reconstruction using parameters predicted by our trained model resemble the results from L_2 reconstruction with considerably less dipole artifacts. We have also demonstrated a significant reduction in the bias introduced by Huber compared to L_1 reconstruction.

The ability of the model trained by our method to generalize to unseen data was estimated using an exhaustive cross-validation schema, where parts of the total data set was held out from training and used as validation set. While the performance of the model is mostly insensitive to how the training and validation sets were partitioned, there are cases where the test performance is low. This indicates that, when the training and validation sets have been partitioned in this specific way, the majority of the scenes in the validation set are of different character than those used for training, indicating a flaw in the size and diversity of our data set.

Our final model incorporates regressors calculated from absolute brightness and pixel variances, and we have conducted an analysis on the effect of either adding regressors calculated from higher order moments for a more complex model, or removing regressors for a simpler model. We found that variance is a necessary characteristic of the scene to capture in the regressors in order to predict good cut-off parameters, but that regressors based on higher order standardized moments such as skewness and kurtosis added no significant predictive power in our setting. This has the advantage of simple implementation in existing frameworks, often at no additional cost in cases where per-pixel variances are computed for other features of a renderer.

In summary, the work conducted in this thesis has shown that even with a simple model considering only basic mean statistics of a rendering scenario as regressors, it is possible to predict and produce a cut-off parameter for the Huber loss function that performs well in comparison to previous loss functions when reconstructing images in gradient-domain path tracing.

6.1 Future Work

Our analysis on the optimal configuration of model regressors was made my selectively adding or removing regressors to be considered from the model. There are more sophisticated means of automatically adding and removing parameters to a model in the case where many potential parameters are involved, which would be necessary to exhaust more potential model configurations in our case.

The generation of large and diverse data sets for rendering relies on both computationally heavy simulations and demand on artistry to generate semi-realistic scenery. There are many scenes available in online repositories, but it is often the case that they need to be modified to work in a specific renderer or sampling algorithm when materials, shapes or other scene components of the scene description have yet to be implemented or defined. For these reasons the data set used in our work is very small in the context of machine learning and parameter estimation, and it would be preferrable to continue our analysis with a larger and more diverse data set to diminish the risk of overfitting and bad generalization performance.

To thoroughly investigate the potential gain from using Huber loss over L_1 loss for image reconstruction, one would have to compare final renders of images from both methods in more detail, and the renders need to be of a higher sample count than of those used in our research. From our comparisons, it is easy to see that Huber introduces less dipole artifacts than L_2 but still has unevenness in the image for low sample counts, and it would be interesting to analyze further if this is preferred visually over the bias introduced by reconstruction using L_1 loss. Production images are rendered using a sample count that is larger than those used here, and for that reason a comparison at higher spp is necessary.

The true performance comparison between our method and previous approaches needs to consider the computational overhead of calculating and outputting regressors from the renderer. This consideration has not been made in our analysis. As noted previously however, pixel variances are often computed regardless for things like adaptive sampling, and the computation time of even higher order moments should intuitively represent a tiny portion of the total render time.

In order to check the generality of the trained model parameters and the model training procedure, an investigation on the performance when using other gradient-domain rendering methods such as G-MLT or G-BDPT needs to be made. It is possible and indeed likely that the distributions of per-pixel primal and gradient samples differ between different methods which may have an impact on the model performance. For the same reasons it is also important to make the same investigation about the effect of using other underlying sampling strategies such as low-discrepancy sampling, and other pixel filters such as the Mitchell-Netravali filter.

Lastly, we note that a natural step for further research in this direction is to investigate a deep learning approach to the image reconstruction problem using neural networks, either to generate the final image from the input directly, to predict Huber loss parameters or as a pre-conditioning step to other reconstruction methods.

References

Arvo, J. (1995): “Analytic methods for simulated light transport”. PhD thesis. Yale University.

Arvo, J. and Kirk, D. (1990): “Particle transport and image synthesis”. *ACM SIGGRAPH Computer Graphics* 24.4, pp. 63–66.

Barron, J. T. (2017): “A More General Robust Loss Function”. *arXiv preprint arXiv:1701.03077.*

Barrow, D. K. and Crone, S. F. (2013): “Crogging (cross-validation aggregation) for forecasting — A novel algorithm of neural network ensembles on time series subsamples”. *Neural Networks (IJCNN), The 2013 International Joint Conference on.* IEEE, pp. 1–8.

Barutçuoğlu, Z. and Alpaydın, E. (2003): “A comparison of model aggregation methods for regression”. *Artificial Neural Networks and Neural Information Processing — ICANN/ICONIP 2003.* Springer, pp. 76–83.

Bellman, R. E. (1957): “Dynamic Programming”.

Bitterli, B. (2015): *Informed choices in primary sample space.*

Bitterli, B. (2016): *Rendering resources.* URL: `https://www.benedikt-bitterli.me/resources/`.

Branch, M. A., Coleman, T. F., and Li, Y. (1999): “A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems”. *SIAM Journal on Scientific Computing* 21.1, pp. 1–23.

Breiman, L. (1996): “Bagging predictors”. *Machine learning* 24.2, pp. 123–140.

Byrd, R. H., Schnabel, R. B., and Shultz, G. A. (1988): “Approximate solution of the trust region problem by minimization over two-dimensional subspaces”. *Mathematical programming* 40.1, pp. 247–263.

Carter, L. L. and Cashwell, E. D. (1975): *Particle-transport simulation with the Monte Carlo method.* Tech. rep. Los Alamos Scientific Lab., N. Mex.(USA).

Celisse, A. and Robin, S. (2008): “Nonparametric density estimation by exact leave-p-out cross-validation”. *Computational Statistics & Data Analysis* 52.5, pp. 2350–2368.

Chambolle, A. and Pock, T. (2011): “A first-order primal-dual algorithm for convex problems with applications to imaging”. *Journal of mathematical imaging and vision* 40.1, pp. 120–145.

Chan, T. F., Golub, G. H., and LeVeque, R. J. (1982): “Updating formulae and a pairwise algorithm for computing sample variances”. *COMPSTAT 1982 5th Symposium held at Toulouse 1982.* Springer, pp. 30–41.

Chan, T. F., Golub, G. H., and LeVeque, R. J. (1983): “Algorithms for computing the sample variance: Analysis and recommendations”. *The American Statistician* 37.3, pp. 242–247.

Chandrasekhar, S. (1950): “Radiative transfer”.

Charbonnier, P., Blanc-Feraud, L., Aubert, G., and Barlaud, M. (1994): “Two deterministic half-quadratic regularization algorithms for computed imaging”. *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference.* Vol. 2. IEEE, pp. 168–172.

Christensen, P. H., Jarosz, W., et al. (2016): “The path to path-traced movies”. *Foundations and Trends® in Computer Graphics and Vision* 10.2, pp. 103–175.

Cook, R. L., Porter, T., and Carpenter, L. (1984): “Distributed ray tracing”. *ACM SIGGRAPH Computer Graphics.* Vol. 18. 3. ACM, pp. 137–145.

Efron, B. and Tibshirani, R. J. (1994): *An introduction to the bootstrap.* CRC press.

Fubini, G. (1907): “Sugli integrali multipli”. *Rend. Acc. Naz. Lincei* 16, pp. 608–614.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. (1995): *Markov chain Monte Carlo in practice.* CRC press.

Goodfellow, I., Bengio, Y., and Courville, A. (2016): *Deep learning.* MIT press.

Grimaldi, F. M. (1665). *Physico-mathesis de lumine, coloribus, et iride. Bononiæ. Reviewed in Philosophical Transactions of the Royal Society of London* 6.79, pp. 3068–3070.

Guilbault, G. G. (1990): *Practical fluorescence.* Vol. 3. CRC Press.

Gunturk, B. K. and Li, X. (2012): *Image restoration: fundamentals and advances.* CRC Press.

Hachisuka, T. and Jensen, H. W. (2009): “Stochastic progressive photon mapping”. *ACM Transactions on Graphics (TOG)* 28.5, p. 141.

Hastings, W. K. (1970): “Monte Carlo sampling methods using Markov chains and their applications”. *Biometrika* 57.1, pp. 97–109.

Heide, F., Diamond, S., Nießner, M., Ragan-Kelley, J., Heidrich, W., and Wetzstein, G. (2016): “ProxImaL: Efficient image optimization using proximal algorithms”. *ACM Transactions on Graphics (TOG)* 35.4, p. 84.

Hua, B.-S., Gruson, A., Nowrouzezahrai, D., and Hachisuka, T. (2017): “Gradient-Domain Photon Density Estimation”. *Computer Graphics Forum.* Vol. 36. 2. Wiley Online Library, pp. 31–38.

Huber, P. J. (1964): “Robust estimation of a location parameter”. *The Annals of Mathematical Statistics* 35.1, pp. 73–101.

Jakob, W. A. (2013): *Light transport on path-space manifolds.* Cornell University.

Jakob, W. A. (2015): *Mitsuba renderer, 2010.* URL: `http://www.mitsuba-renderer.org`.

Jakob, W. A. and Marschner, S. (2012): "Manifold exploration: a Markov Chain Monte Carlo technique for rendering scenes with difficult specular transport". *ACM Transactions on Graphics (TOG)* 31.4, p. 58.

Jarosz, W. (2008): *Efficient Monte Carlo methods for light transport in scattering media.* University Of California, San Diego.

Jensen, H. W. (2001): *Realistic image synthesis using photon mapping.* AK Peters, Ltd.

Kajiya, J. T. (1986): "The rendering equation". *ACM Siggraph Computer Graphics.* Vol. 20. 4. ACM, pp. 143–150.

Kettunen, M., Manzi, M., Aittala, M., Lehtinen, J., Durand, F., and Zwicker, M. (2015): "Gradient-domain path tracing". *ACM Transactions on Graphics (TOG)* 34.4, p. 123.

Kinoshita, S., Yoshioka, S., and Miyazaki, J. (2008): "Physics of structural colors". *Reports on Progress in Physics* 71.7, p. 076401.

Kohavi, R. (1995): "A study of cross-validation and bootstrap for accuracy estimation and model selection". *IJCAI.* Vol. 14. 2. Stanford, CA, pp. 1137–1145.

Kolb, C., Mitchell, D., and Hanrahan, P. (1995): "A realistic camera model for computer graphics". *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques.* ACM, pp. 317–324.

Lafortune, E. P. and Willems, Y. D. (1993): "Bi-directional path tracing".

Lehtinen, J., Karras, T., Laine, S., Aittala, M., Durand, F., and Aila, T. (2013): "Gradient-domain metropolis light transport". *ACM Transactions on Graphics (TOG)* 32.4, p. 95.

Manzi, M., Kettunen, M., Aittala, M., Lehtinen, J., Durand, F., and Zwicker, M. (2015): "Gradient-domain bidirectional path tracing". *Proc. Eurographics Symposium on Rendering.* Vol. 1. 2, p. 3.

Manzi, M., Kettunen, M., Durand, F., Zwicker, M., and Lehtinen, J. (2016): "Temporal gradient-domain path tracing". *ACM Transactions on Graphics (TOG)* 35.6, p. 246.

Manzi, M., Rousselle, F., Kettunen, M., Lehtinen, J., and Zwicker, M. (2014): "Improved sampling for gradient-domain metropolis light transport". *ACM Transactions on Graphics (TOG)* 33.6, p. 178.

Manzi, M., Vicini, D., and Zwicker, M. (2016): "Regularizing image reconstruction for gradient-domain rendering with feature patches". *Computer graphics forum.* Vol. 35. 2. Wiley Online Library, pp. 263–273.

MATLAB Optimization Toolbox (2017). The MathWorks, Natick, MA, USA.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953): "Equation of state calculations by fast computing machines". *The journal of chemical physics* 21.6, pp. 1087–1092.

Mitchell, D. P. and Netravali, A. N. (1988): "Reconstruction filters in computer-graphics". *ACM Siggraph Computer Graphics* 22.4, pp. 221–228.

Mood, A. M. (1950): "Introduction to the Theory of Statistics."

Nicodemus, F. E. (1965): "Directional reflectance and emissivity of an opaque surface". *Applied optics* 4.7, pp. 767–775.

Pebay, P. P. (2008): *Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments.* Tech. rep. Sandia National Laboratories.

Pérez, P., Gangnet, M., and Blake, A. (2003): "Poisson image editing". *ACM Transactions on graphics (TOG)* 22.3, pp. 313–318.

Porter, T. and Duff, T. (1984): "Compositing digital images". *ACM Siggraph Computer Graphics.* Vol. 18. 3. ACM, pp. 253–259.

Rousselle, F., Jarosz, W., and Novák, J. (2016): "Image-space control variates for rendering". *ACM Transactions on Graphics (TOG)* 35.6, p. 169.

Ruderman, D. L. (1994): "The statistics of natural images". *Network: computation in neural systems* 5.4, pp. 517–548.

Shirley, P. and Wang, C. (1994): "Direct lighting calculation by monte carlo integration". *Photorealistic Rendering in Computer Graphics*, pp. 52–59.

Simoncelli, E. P. and Olshausen, B. A. (2001): "Natural image statistics and neural representation". *Annual review of neuroscience* 24.1, pp. 1193–1216.

Sun, W., Sun, X., Carr, N. A., Nowrouzezahrai, D., and Ramamoorthi, R. (2017): "Gradient-Domain Vertex Connection and Merging".

Terriberry, T. B. (2008): *Computing higher-order moments online.*

Turkowski, K. (1990): "Filters for common resampling tasks". *Graphics gems.* Academic Press Professional, Inc., pp. 147–165.

Veach, E. (1998): "Robust monte carlo methods for light transport simulation". PhD thesis. Stanford University Stanford.

Veach, E. and Guibas, L. J. (1995a): "Bidirectional estimators for light transport". *Photorealistic Rendering Techniques.* Springer, pp. 145–167.

Veach, E. and Guibas, L. J. (1995b): "Optimally combining sampling techniques for Monte Carlo rendering". *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques.* ACM, pp. 419–428.

Veach, E. and Guibas, L. J. (1997): "Metropolis light transport". *Proceedings of the 24th annual conference on Computer graphics and interactive techniques.* ACM Press/Addison-Wesley Publishing Co., pp. 65–76.

Ward, G. J., Rubinstein, F. M., and Clear, R. D. (1988): "A ray tracing solution for diffuse interreflection". *ACM SIGGRAPH Computer Graphics* 22.4, pp. 85–92.

Wu, J., Zheng, C., Hu, X., and Li, C. (2011): "An Accurate and Practical Camera Lens Model for Rendering Realistic Lens Effects". *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2011 12th International Conference on.* IEEE, pp. 63–70.

Xu, Q.-S. and Liang, Y.-Z. (2001): "Monte Carlo cross validation". *Chemometrics and Intelligent Laboratory Systems* 56.1, pp. 1–11.

Zhang, P. (1993): "Model selection via multifold cross validation". *The Annals of Statistics*, pp. 299–313.

A Standardized Central Moments

Let $\{z_1, z_2, \ldots, z_n\}$ be a set of independent and identically distributed random variables, $z_t \sim z\, \forall t \in \{1, 2, \ldots, n\}$. In the context of this thesis, the set of random variables can be thought of as the collection of samples that that contribute to the color of a pixel, z_t being one such color throughput sample. Define the *weighted sample mean random variable* as

$$\bar{z} = \sum_t w_t z_t \tag{A1}$$

where the weights w_t form a partition of unity

$$\sum_t w_t = 1. \tag{A2}$$

The purpose of this chapter is to derive formulas for standardized central moments. Specifically, we want a formula for the standardized central moment of arbitrary order k, for the sample mean random variable $\bar{z}$. For instance, the skewness and kurtosis of the sample mean is the third order and fourth order standardized central moments, respectively.

The k:th standardized central moment for a random variable z is defined as the ratio between the k:th central moment and the k:th power of the standard deviation

$$\gamma_k = \frac{\mu_k}{\sigma^k} = \frac{\mathbf{E}\big[(z-\mu)^k\big]}{\big(\mathbf{E}\big[(z-\mu)^2\big]\big)^{k/2}}, \tag{A3}$$

where $\mu = \mathbf{E}[z]$ is the true mean of z. Let $\bar{\gamma}_k$ denote the standardized central moment of the sample mean random variable of order k. To derive the formula for $\bar{\gamma}_k$, first note that the expectation of $\bar{z}$ is equal to the expectation of its constituent random variables z

$$\mathbf{E}[\bar{z}] = \mathbf{E}\Big[\sum_t w_t z_t\Big] = \sum_t w_t \mathbf{E}[z_t] = \sum_t w_t \mu = \mu \sum_t w_t = \mu. \tag{A4}$$

We then have

$$\bar{\gamma}_k = \frac{\bar{\mu}_k}{\bar{\sigma}^k} = \frac{\mathbf{E}\big[(\bar{z}-\mu)^k\big]}{\big(\mathbf{E}\big[(\bar{z}-\mu)^2\big]\big)^{k/2}}. \tag{A5}$$

The expectation within the parenthesis of the denominator is the variance $\bar{\sigma}^2$ of the sample mean random variable $\bar{z}$ and a special case of the numerator with $k = 2$. We thus proceed to evaluate the expectation in the numerator.

$$\begin{aligned}\bar{\mu}_k &= \mathbf{E}\big[(\bar{z}-\mu)^k\big]\\ &= \mathbf{E}\Big[\Big(\sum_t w_t z_t - \mu\Big)^k\Big]\\ &= \mathbf{E}\Big[\Big(\sum_t w_t(z_t-\mu)\Big)^k\Big].\end{aligned} \tag{A6}$$

Expanding the product in (A6) consisting of k sums and subsequently rearranging the sums gives

$$\begin{aligned}&\mathbf{E}\Big[\Big(\sum_t w_t(z_t-\mu)\Big)^k\Big]\\ =&\mathbf{E}\Big[\Big(\sum_{t_1} w_{t_1}(z_{t_1}-\mu)\Big)\Big(\sum_{t_2} w_{t_2}(z_{t_2}-\mu)\Big)\cdots\Big(\sum_{t_k} w_{t_k}(z_{t_k}-\mu)\Big)\Big]\\ =&\mathbf{E}\Big[\sum_{t_1}\sum_{t_2}\cdots\sum_{t_k}\big\{w_{t_1}(z_{t_1}-\mu)w_{t_2}(z_{t_2}-\mu)\cdots w_{t_k}(z_{t_k}-\mu)\big\}\Big]\end{aligned} \tag{A7}$$

Taking the expectation is a summation and we are therefore allowed to switch the order of the expectation and the sums. By doing this and gathering the weights outside of the expectation, (A7) becomes

$$\begin{aligned}&\mathbf{E}\Big[\sum_{t_1}\sum_{t_2}\cdots\sum_{t_k}\big\{w_{t_1}w_{t_2}\cdots w_{t_k}(z_{t_1}-\mu)(z_{t_2}-\mu)\cdots(z_{t_k}-\mu)\big\}\Big]\\ =&\sum_{t_1}\sum_{t_2}\cdots\sum_{t_k}\big\{w_{t_1}w_{t_2}\cdots w_{t_k}\mathbf{E}[(z_{t_1}-\mu)(z_{t_2}-\mu)\cdots(z_{t_k}-\mu)]\big\}\end{aligned} \tag{A8}$$

Now, for all $t_u \neq t_v$ where $u, v \in \{1, 2, \ldots, k\}$, the random variables z_{t_u} and z_{t_v} are identically distributed with expected value μ. They are also independent, which means that

$$\begin{aligned}\mathbf{E}[(z_{t_u}-\mu)(z_{t_v}-\mu)] &= \mathbf{E}[(z_{t_u}-\mu)]\mathbf{E}[(z_{t_v}-\mu)]\\ &= (\mathbf{E}[z_{t_u}]-\mu)(\mathbf{E}[z_{t_v}]-\mu)\\ &= (\mu-\mu)(\mu-\mu)\\ &= 0.\end{aligned} \tag{A9}$$

By this observation, the only surviving terms in the sum (A8) are those where $u = v$ and we get

$$\bar{\mu}_k = \sum_t w_t^k \mathbf{E}\big[(z_t-\mu)^k\big] = \mu_k \sum_t w_t^k. \tag{A10}$$

We now have the formula for the k:th order standardized central moment of the weighted sample mean random variable $\bar{z}$ as

$$\bar{\gamma}_k = \frac{\mu_k \sum_t w_t^k}{\big(\mu_2 \sum_t w_t^2\big)^{k/2}} \tag{A11}$$

In the common case where uniform weights are used for each sample z_t when computing the sample mean $\bar{z}$, we have $w_t = n^{-1} \; \forall t$ and (A11) becomes

$$\bar{\gamma}_k = \frac{n\mu_k}{(n\mu_2)^{k/2}}. \tag{A12}$$

In summary, we now have the variance, skewness and kurtosis of $\bar{z}$ readily available as

$$\mathrm{Var}[\bar{z}] = \bar{\mu}_2 = \frac{\mu_2}{n} \tag{A13}$$

$$\mathrm{Skew}[\bar{z}] = \bar{\gamma}_3 = \frac{n\mu_3}{(n\mu_2)^{3/2}} \tag{A14}$$

$$\mathrm{Kurt}[\bar{z}] = \bar{\gamma}_4 = \frac{n\mu_4}{(n\mu_2)^{2}} \tag{A15}$$

B Visual Comparison of α Parameter

In the Poisson reconstruction step of gradient-domain rendering, α is a paramter that controls the relative weight between primal and gradient samples. Lehtinen et al. (2013) suggest the use of $\alpha = 0.2$ which we have adopted in this work. We proceed to show visual results of reconstructed images for a few select scenes that have been reconstructed using a range of different α. For each α that was used, we used the cut-off parameter δ^*, the δ that produced the best result in terms of relMSE.

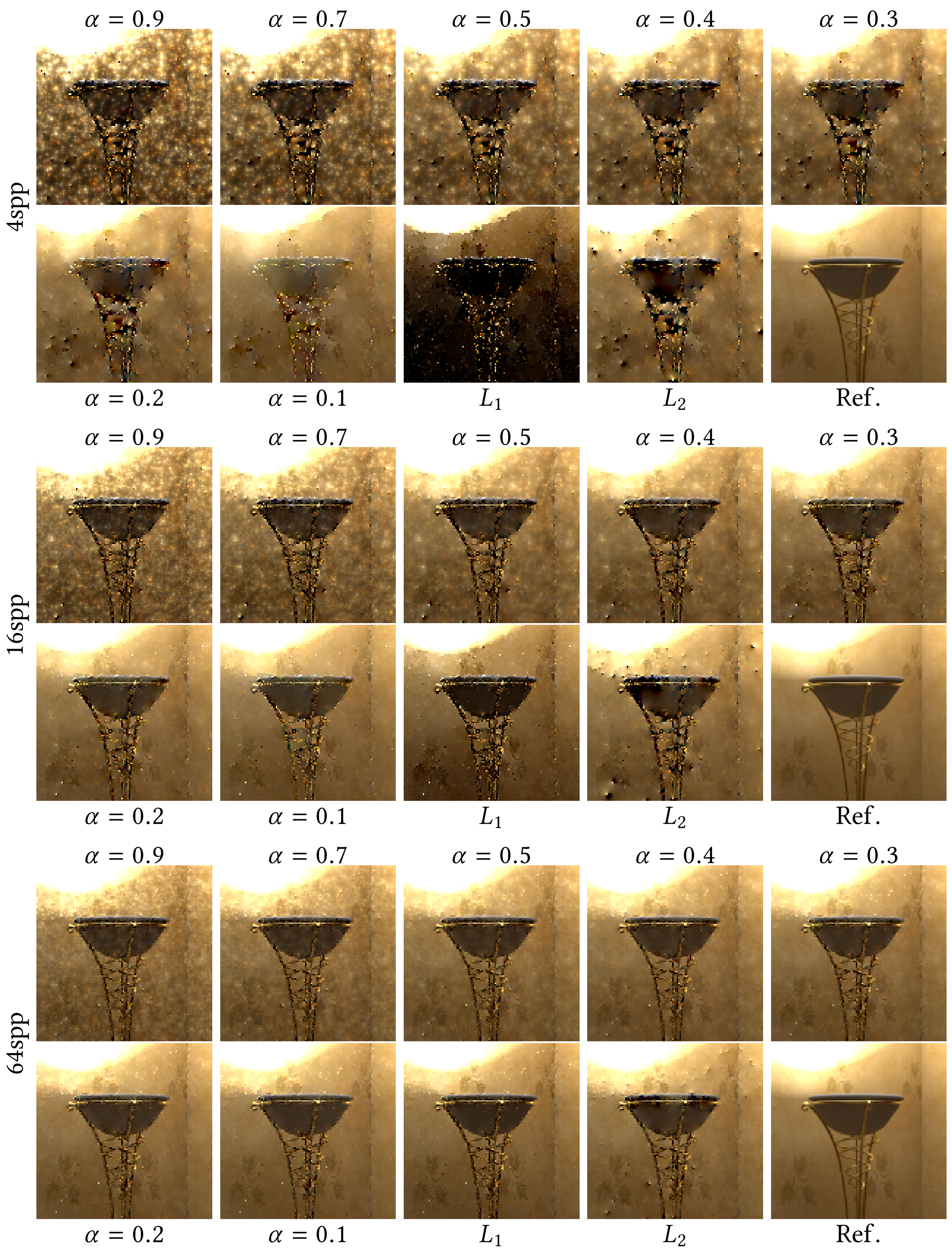


Figure B1: α comparison for the BOOKSHELF scene.

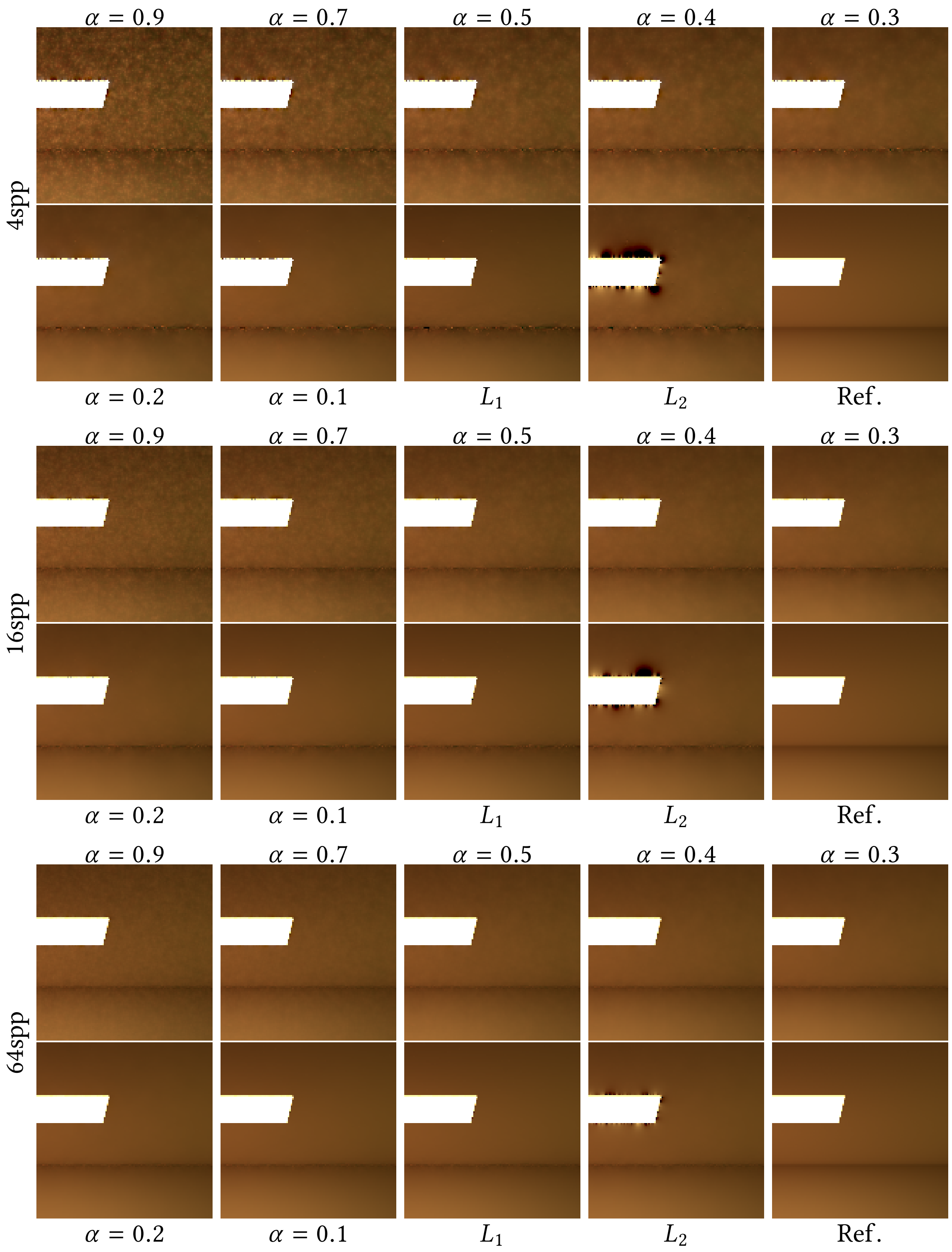


Figure B2: α comparison for the Cornell Box scene.

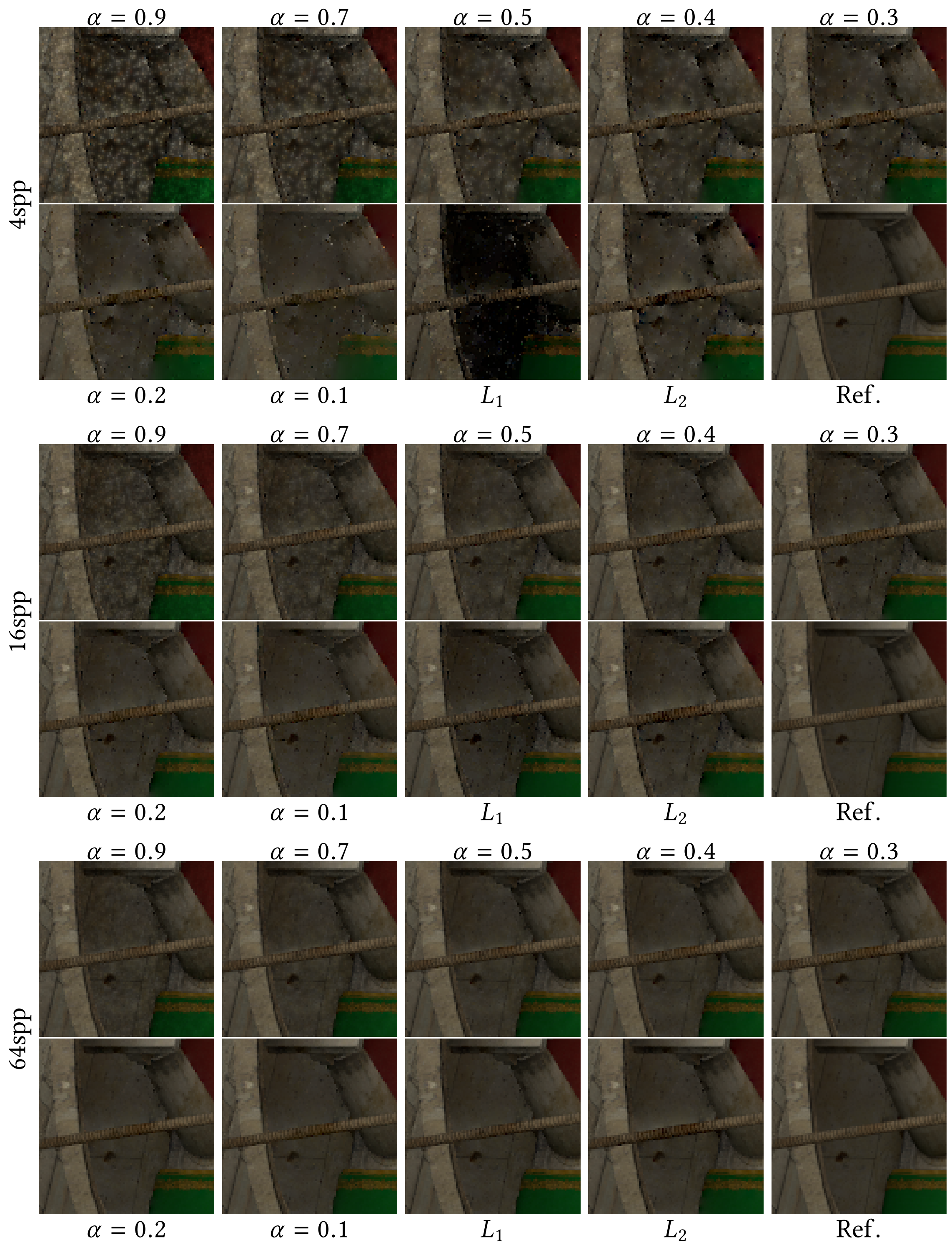


Figure B3: α comparison for the Crytek Sponza scene.

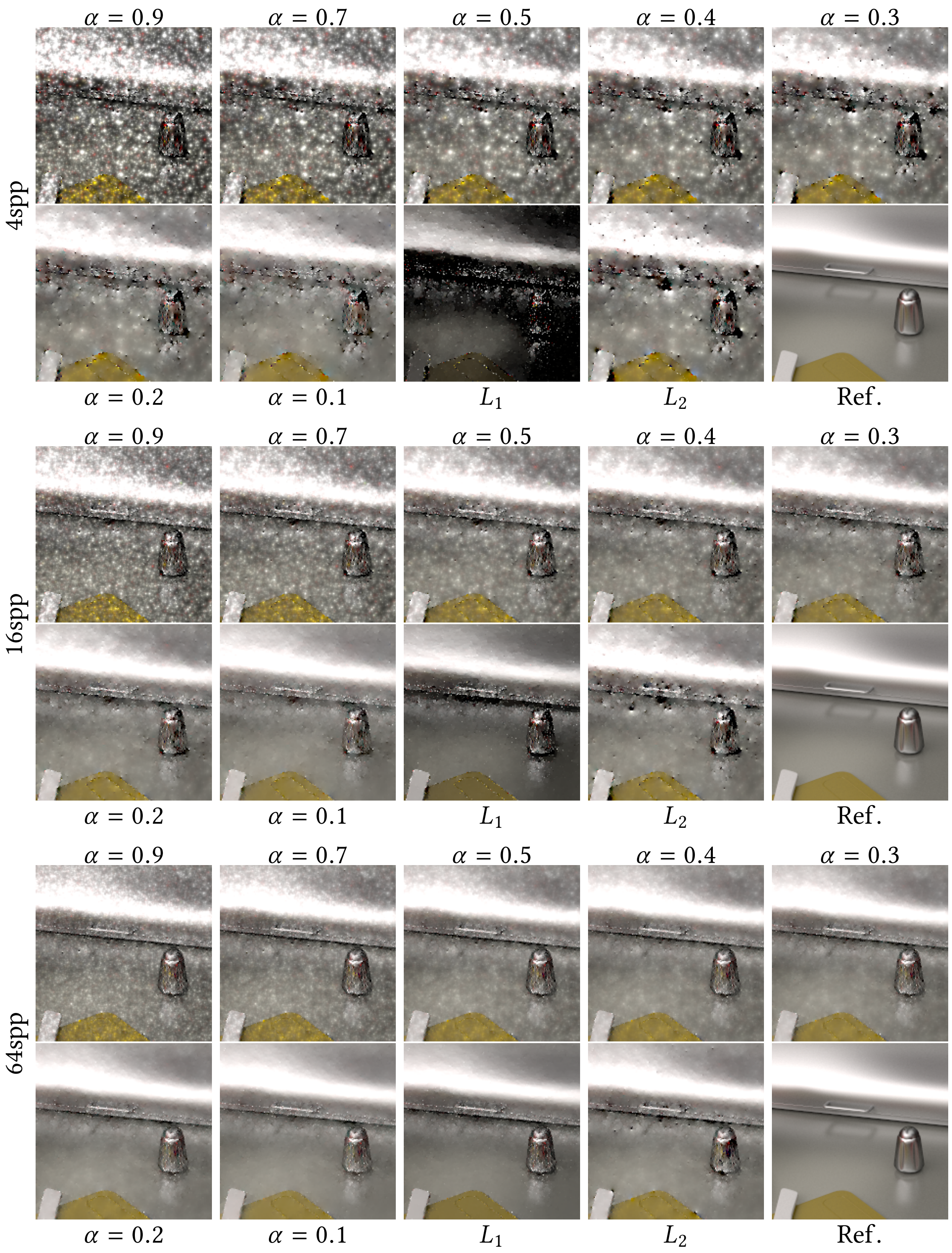


Figure B4: α comparison for the Kitchen 1 scene.

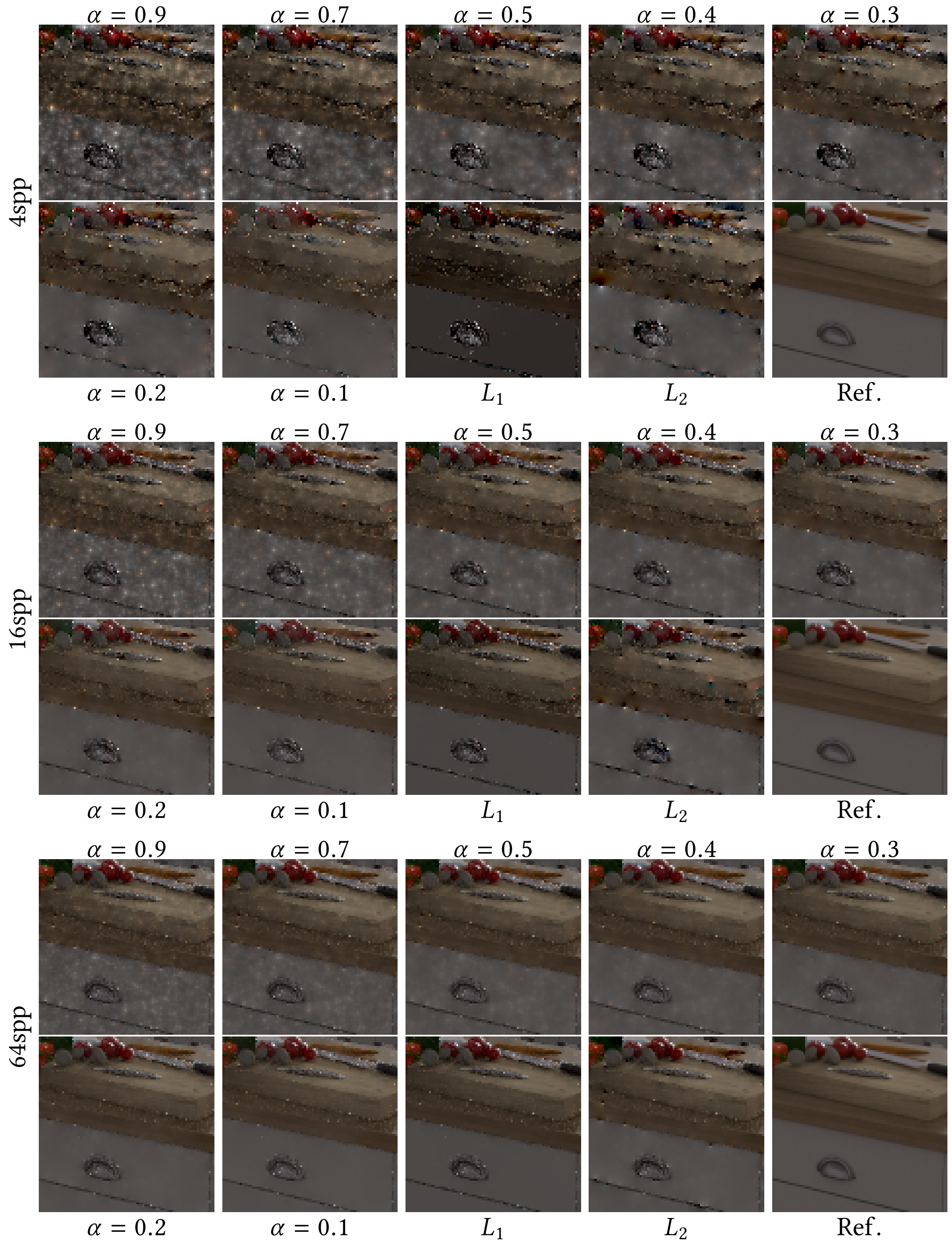


Figure B5: α comparison for the Kitchen 2 scene.

C Reference Images

This chapter is a collection of reference renders of the scenes used in this thesis. The references are used to compare the results of a given method to the ground truth – the goodness of the method is measured in terms of how little it deviates from the reference. The samples in the data set used to train our model consists of renders of these scenes with the same camera angles rendered using the gradient-domain path tracing algorithm at different sample counts. All reference images shown in the following were rendered using the Mitsuba implementation of bidirectional path tracing (Jakob 2015). Many of the scenes were downloaded from Bitterli (2016).

Table C1: Scene sources.

Bathroom 1	Blend Swap user Mareck
Bathroom 2	Blend Swap user nacimus
Bathroom 3	Ported to Mitsuba by Tiziano Portenier
Bookshelf	Ported to Mitsuba by Tiziano Portenier
Cornell Box	Ported to Mitsuba by Wenzel Jakob
Classroom	Blend Swap user NovaZeeke
Crytek Sponza	Frank Meinl
Kitchen 1	Licensed from Evermotion
Kitchen 2	Blend Swap user Jay-Artist
Living Room 1	Blend Swap user Wig42
Living Room 2	Blend Swap user Jay-Artist
Sponza	Marko Dabrovic
Staircase 1	Blend Swap user Wig42
Staircase 2	Blend Swap user NewSee2l035
Veach Door	Miika Aittala, Samuli Laine, and Jaakko Lehtinen

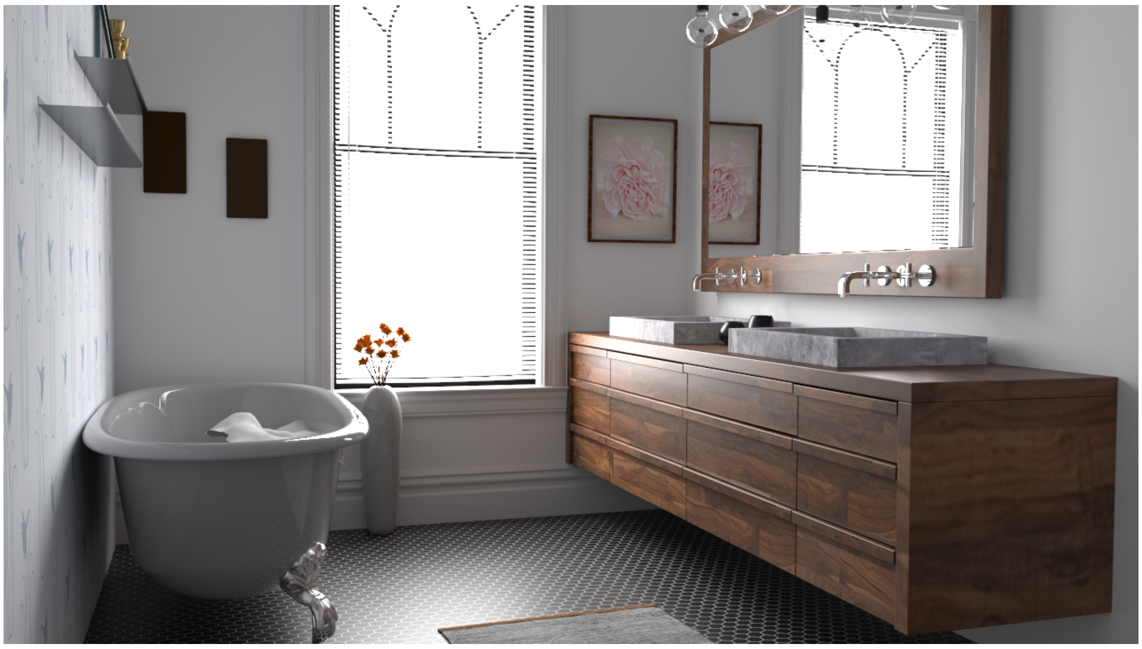


Figure C1: Reference image of the BATHROOM 1 scene, rendered at 1280 × 720 resolution and 32K spp.

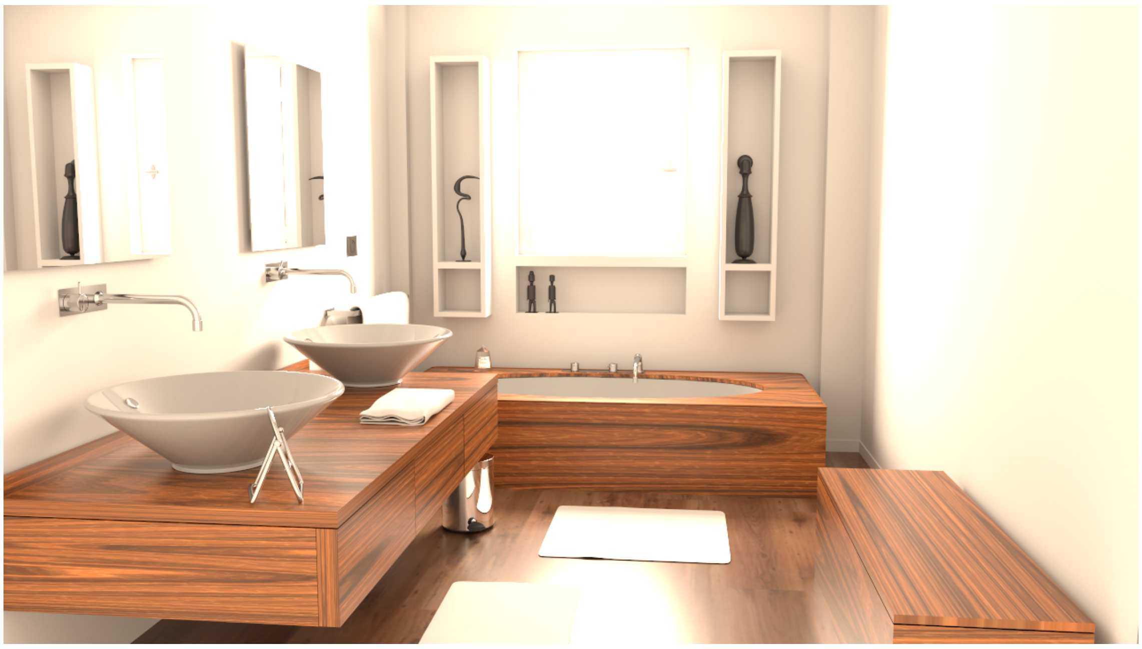


Figure C2: Reference image of the BATHROOM 2 scene, rendered at 1280 × 720 resolution and 32K spp.

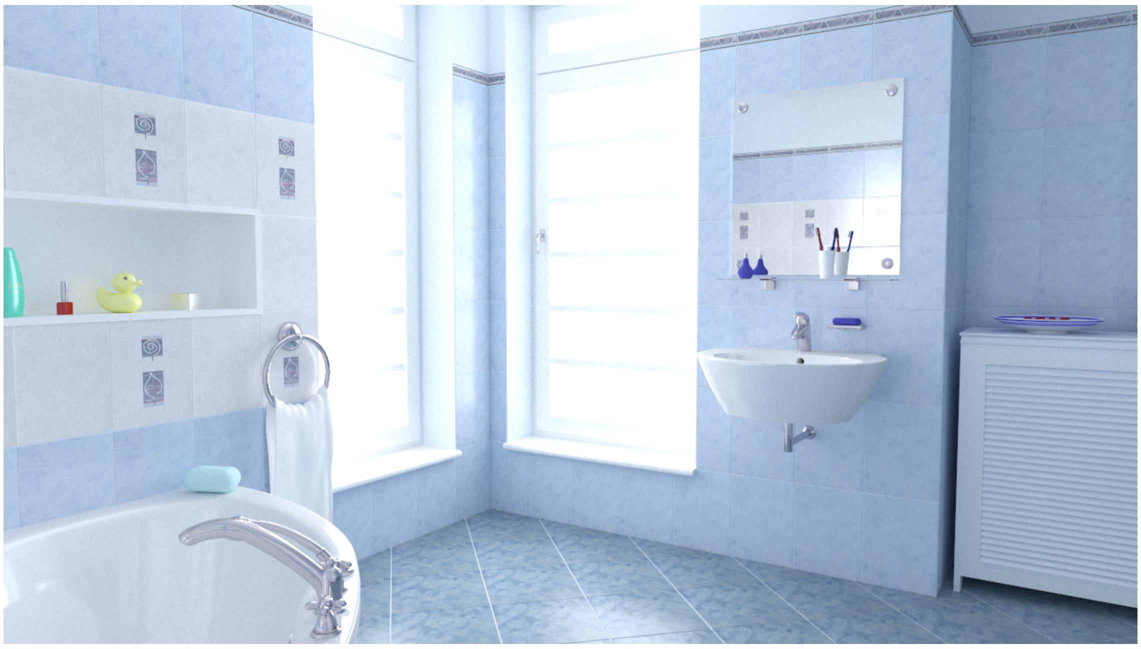


Figure C3: Reference image of the Bathroom 3 scene, rendered at 1280 × 720 resolution and 40k spp.

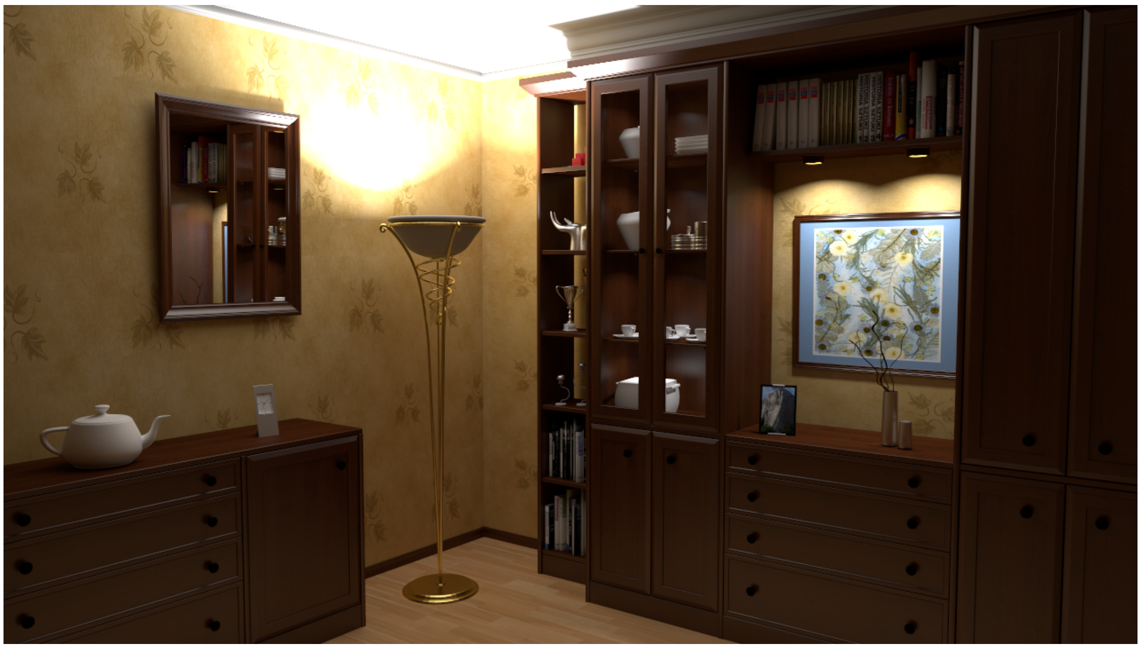


Figure C4: Reference image of the Bookshelf scene, rendered at 1280 × 720 resolution and 18k spp.

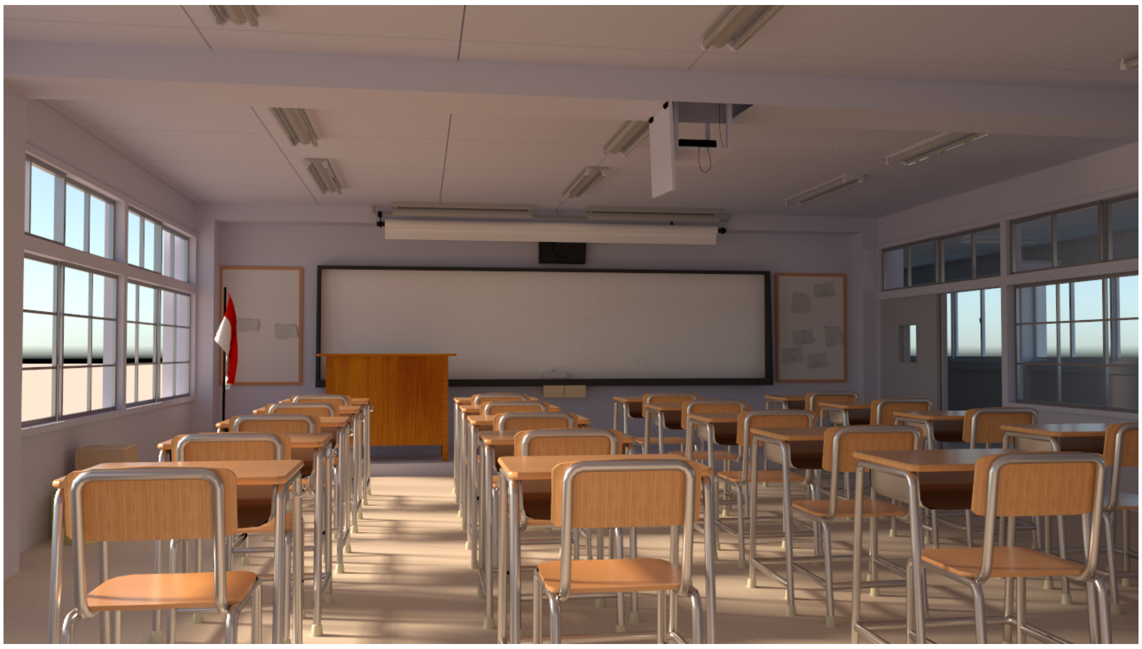


Figure C5: Reference image of the Classroom scene, rendered at 1280 × 720 resolution and 32k spp.

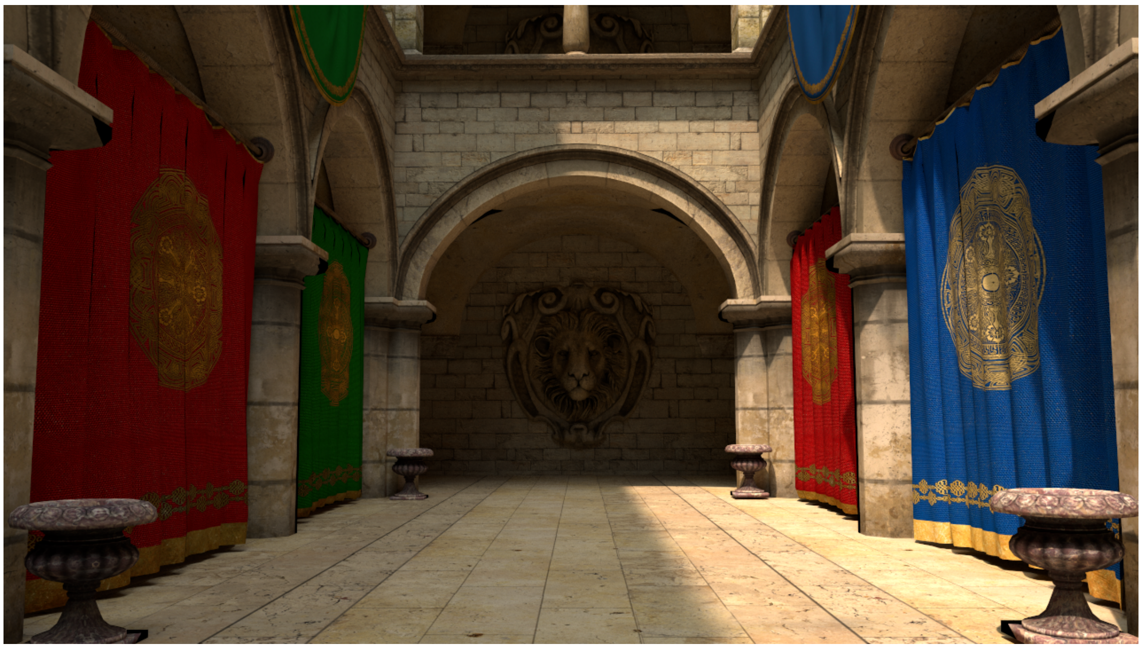


Figure C6: Reference image of the Crytek Sponza scene, rendered at 1280 × 720 resolution and 24k spp.

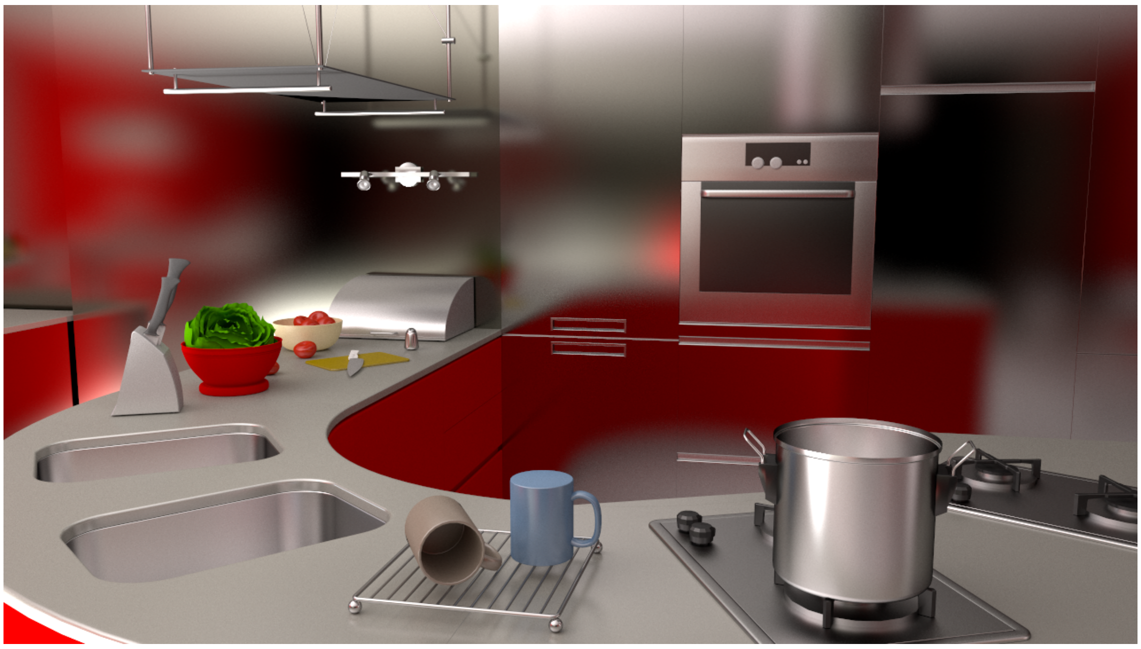


Figure C7: Reference image of the Kitchen 1 scene, rendered at 1280 × 720 resolution and 32k spp.

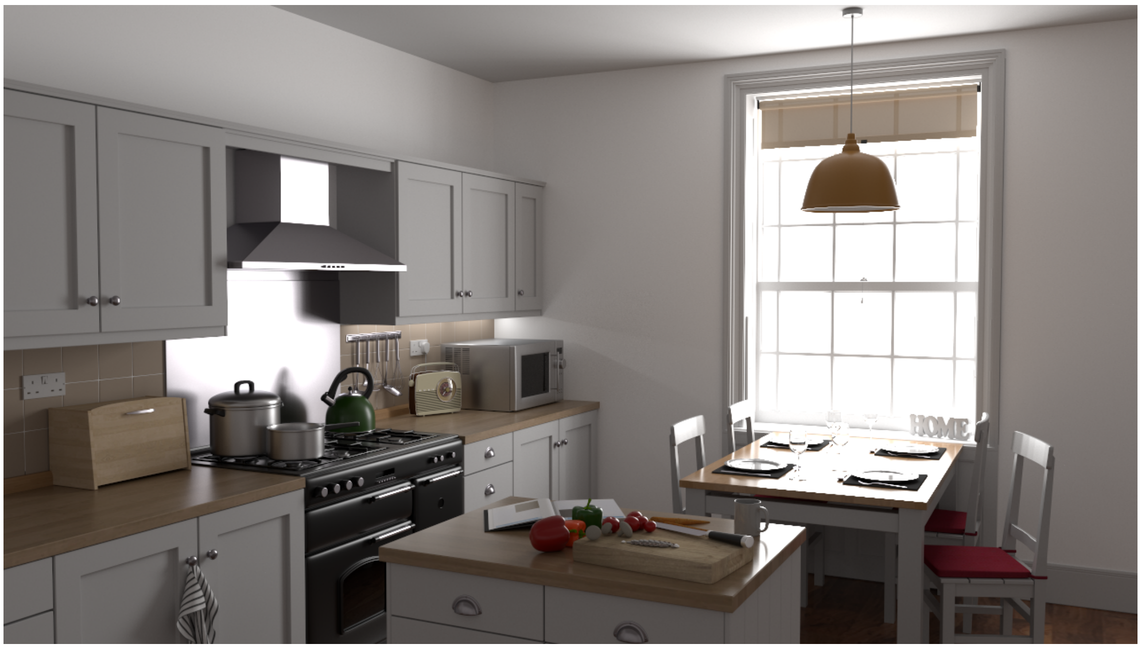


Figure C8: Reference image of the Kitchen 2 scene, rendered at 1280 × 720 resolution and 32k spp.

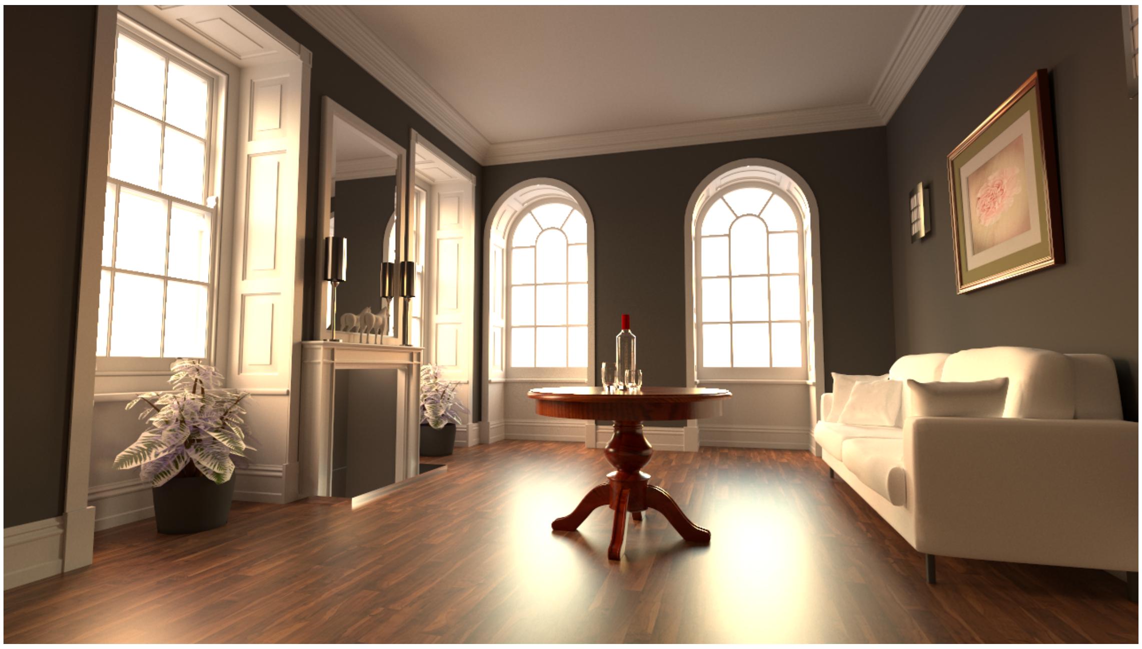


Figure C9: Reference image of the Living Room 1 scene, rendered at 1280×720 resolution and 32k spp.

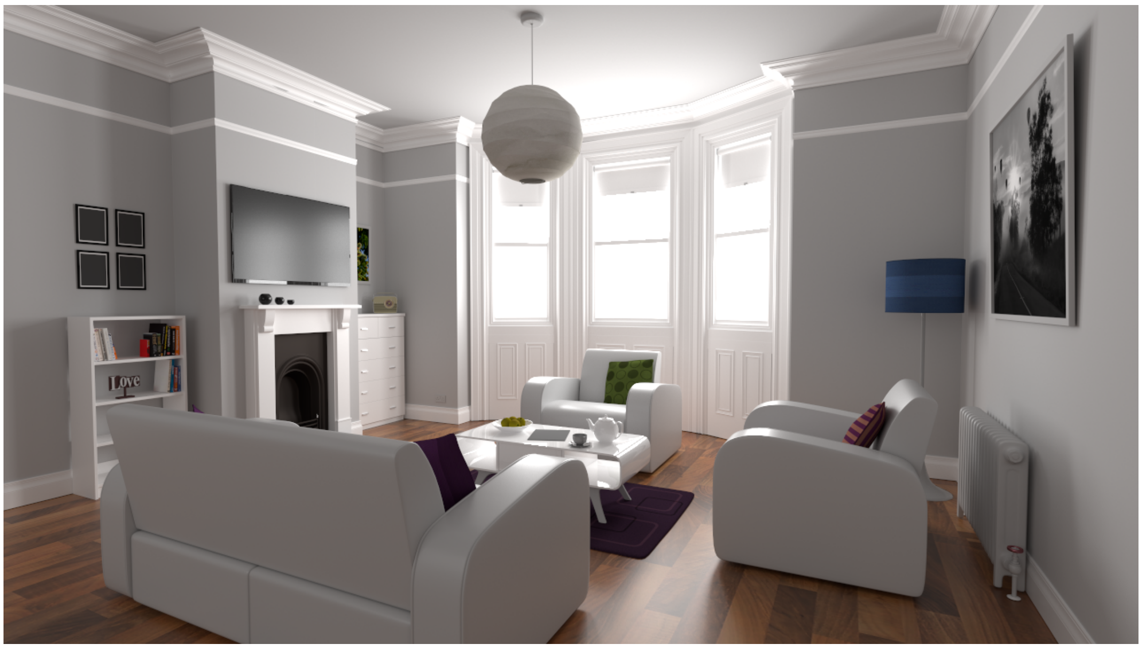


Figure C10: Reference image of the Living Room 2 scene, rendered at 1280 × 720 resolution and 32k spp.

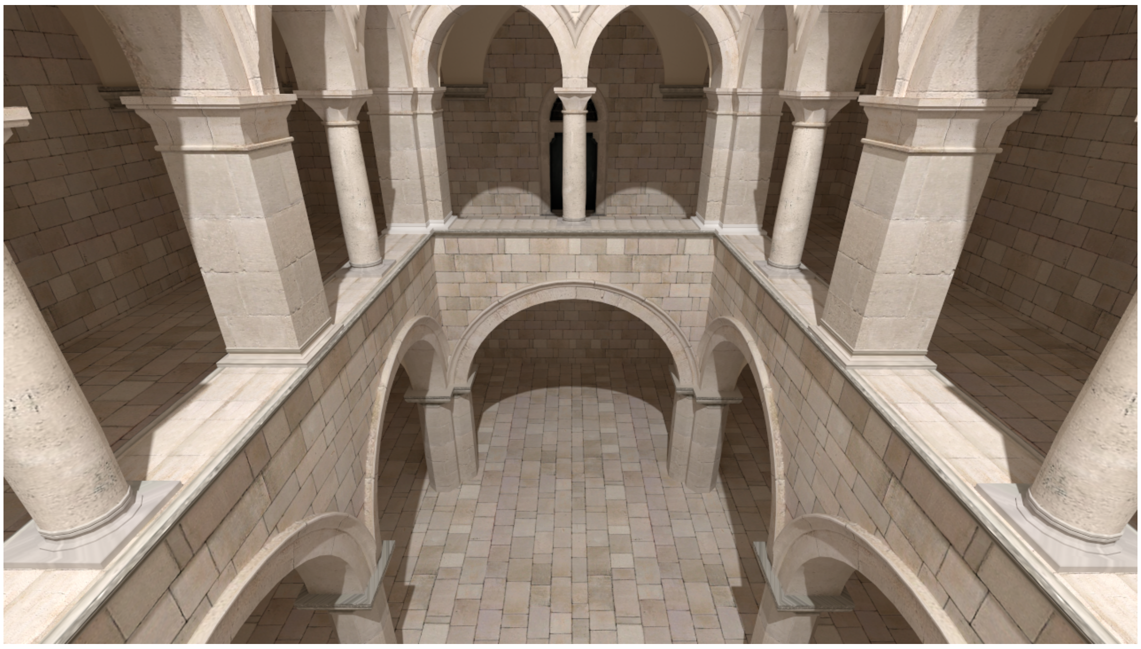


Figure C11: Reference image of the Sponza scene, rendered at 1280 × 720 resolution and 24k spp.

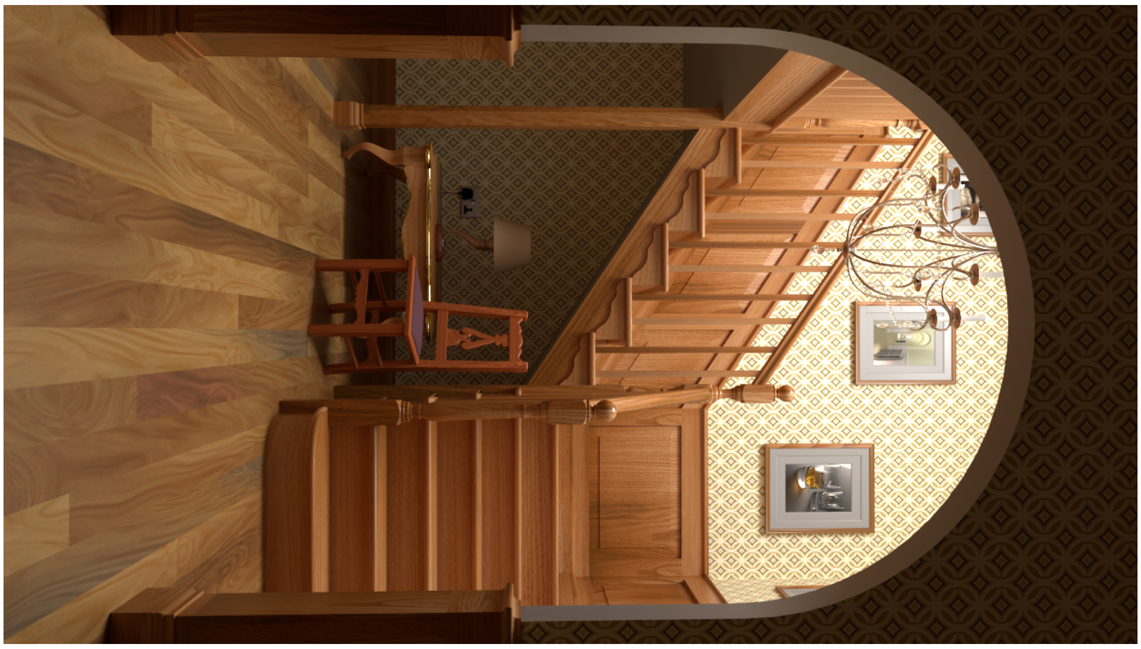


Figure C12: Reference image of the Staircase 1 scene, rendered at 720 × 1280 resolution and 32k spp. Rotated 90° clockwise.

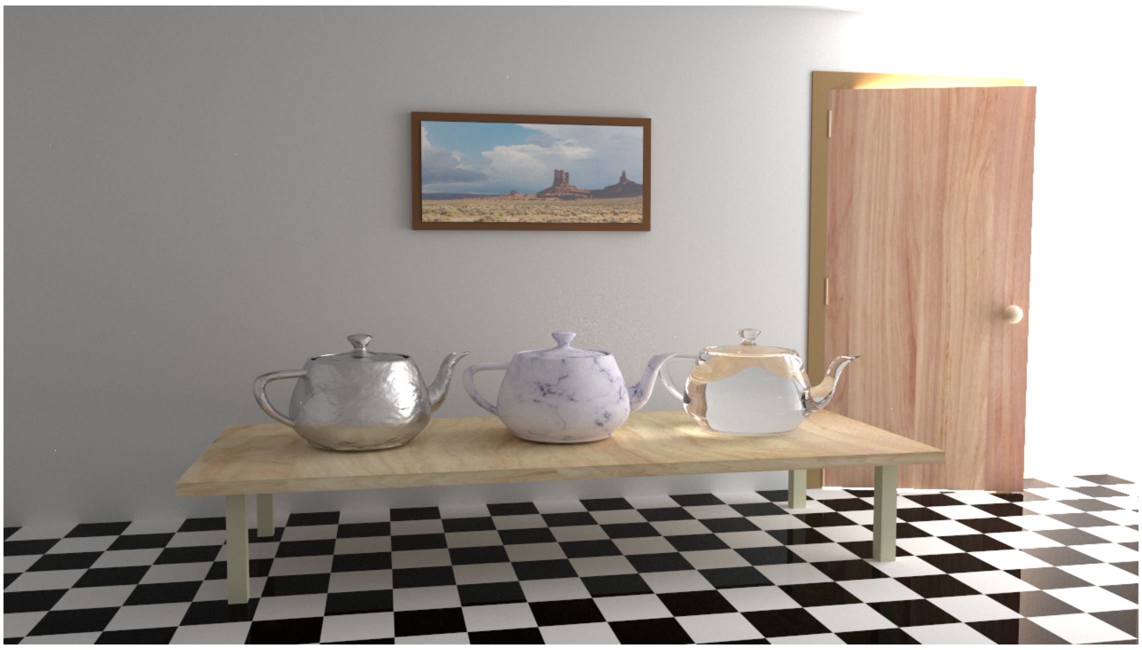


Figure C13: Reference image of the Veach Door scene, rendered at 1280×720 resolution and 32k spp.

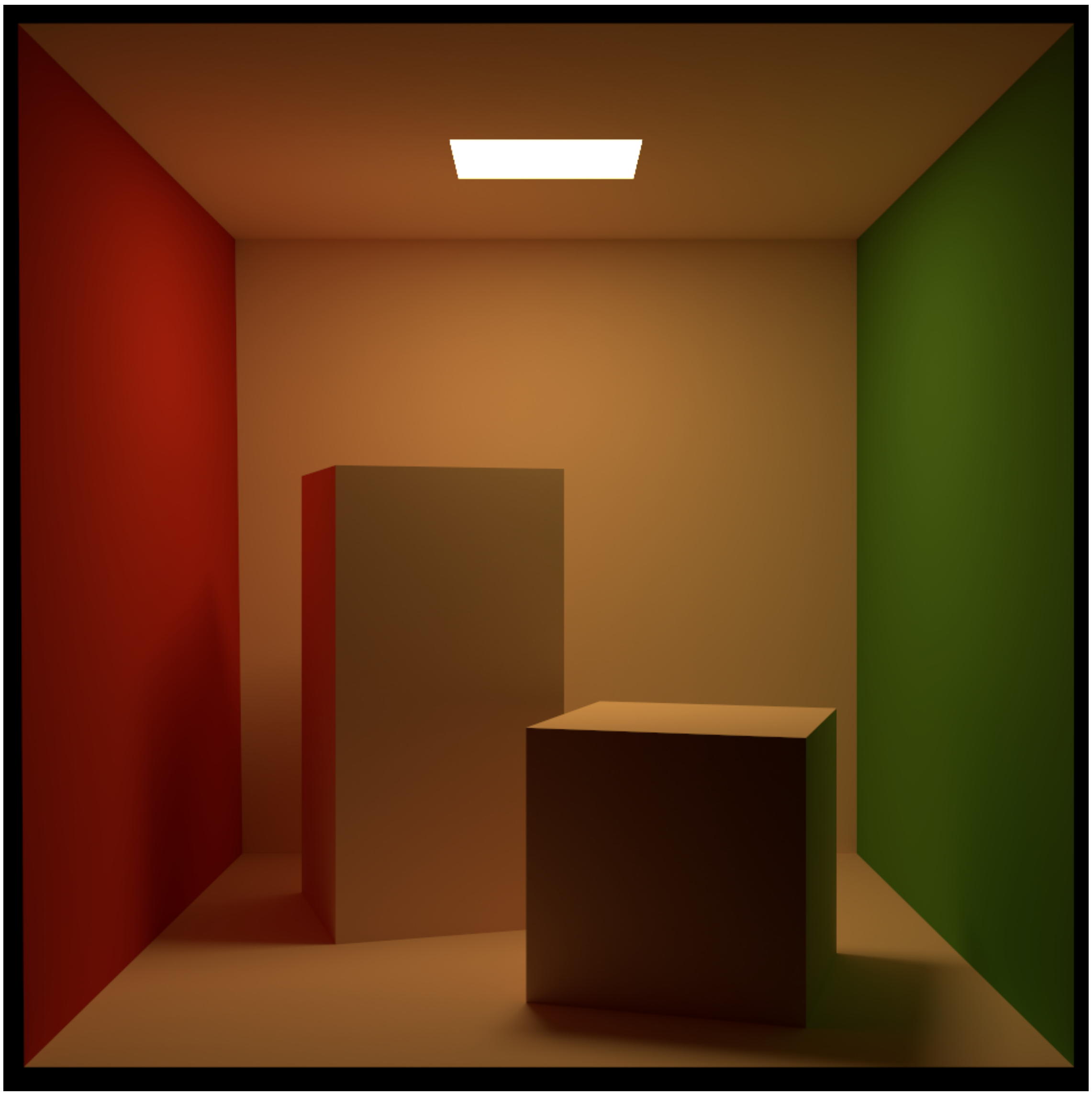


Figure C14: Reference image of the Cornell Box scene, rendered at 960×960 resolution and 12k spp.

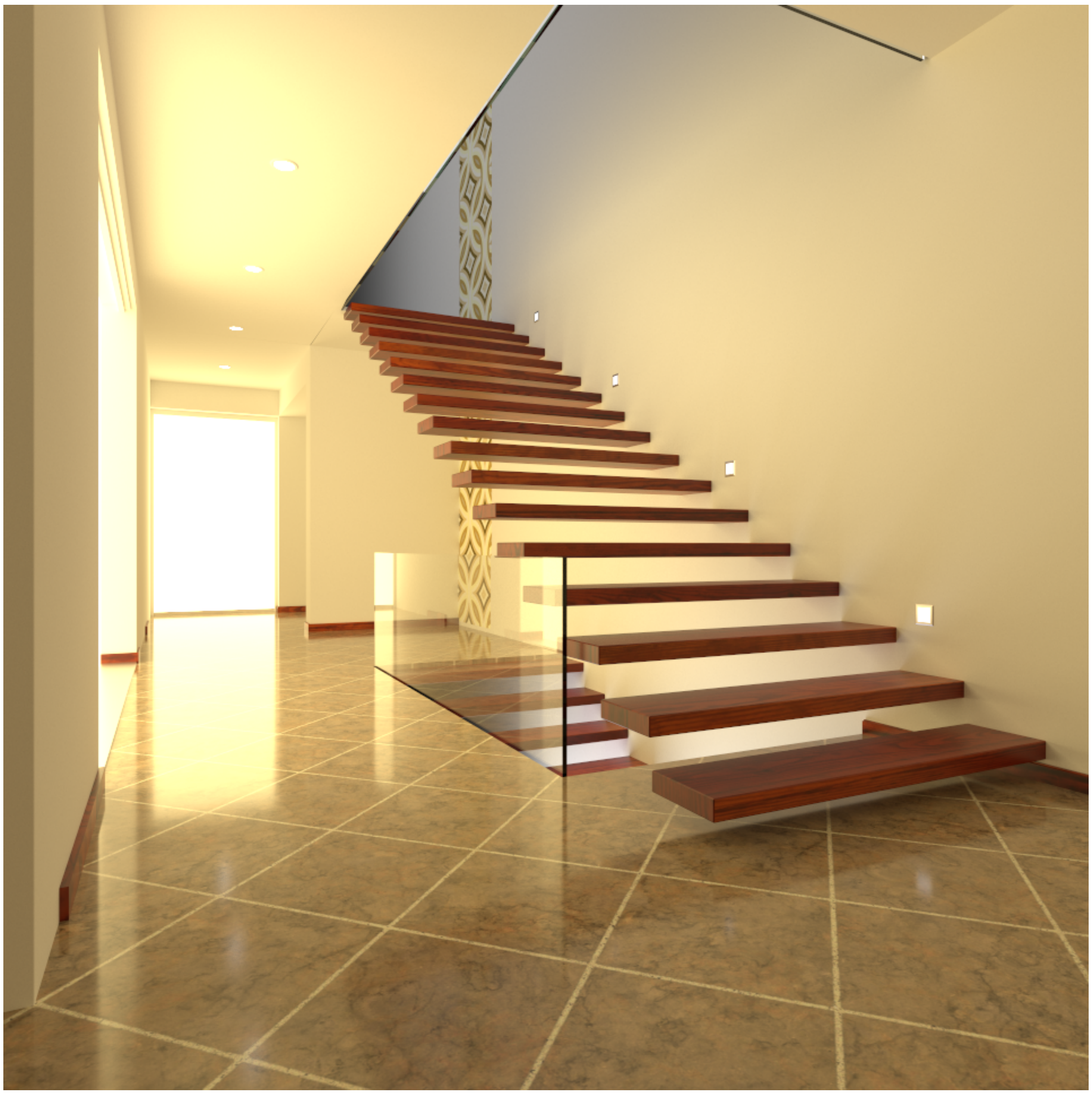


Figure C15: Reference image of the STAIRCASE 2 scene, rendered at 960 × 960 resolution and 32K spp.