Bachelor Thesis

Translucent Material Parameter Estimation

Author

Saip Can Hasbay

desired academic degree

Bachelor of Science (BSc)

Vienna, 2022

Studienkennzahl lt. Studienblatt:	UA 033 661
Fachrichtung:	Informatik - Medieninformatik
Betreuerin / Betreuer:	Projektass.(FWF) PhD David Hahn

Contents

Acknowledgments

The author would like to thank the following people who directly or indirectly influenced the completion of this project. First and foremost, special thanks to David Hahn, for his support and patience with this author. Without his help and *compression* algorithm, neither the paper nor thesis version of this project would be possible. Many thanks also to Ass.Prof. Dipl.-Ing. Dr.techn. Peter Ferschin for his support and suggestions during the experimental stage of this project. Another special thanks to Prof. Torsten Möller, Ph.D., for his help during the topic discovery phase. This endeavor would not have been possible without the Mitsuba team at EPFL in Switzerland, which provided an incredible tool for this author and the whole computer graphics community. Thanks should also go to Siemens Mobility GmbH Austria, specifically to the project leads and members at project CARMEN and CIRCE for their understanding throughout this period. Words cannot express this author's gratitude to Robert Kollruss, who had been an excellent teacher and colleague. Lastly, but most importantly, the author would like to thank his family for the hardships they had to endure and for fanning the flames of education throughout these years for the love of their son and brother.

Abstract

In this project, we present a workflow for optical material parameter estimation based on real-world images, numerical optimization, and differentiable rendering (building upon the publicly available *Mitsuba 3* code). We primarily focus on *translucent* materials and demonstrate our approach's applicability to both synthetic and real-world test cases. Specifically, we estimate parameters for two well-known material models: *Principled BSDF* (a surface model based on Burley's Disney BSDF), or *Rough dielectric BSDF*, describing volumetric *homogeneous* participating media.

To solve the inverse rendering problem of acquiring suitable material properties from a (set of) given reference image(s), we present a software tool that handles the entire material reconstruction workflow. Given a virtual 3D scene description and multiple images, our tool implements a gradient-based optimization algorithm, provides a user interface to select the optimization parameters and different formulations of the optimization objective, and allows users to visualize and export the material reconstruction results of a given (set of) reference image(s).

Finally, we also propose various experimental workflows to acquire the necessary reference images and potentially the 3D scene geometry itself. Our results show that our approach works on well-known materials, such as acrylic glass, and newly designed materials, such as alginate, from experimental materials research.

1 Introduction

In this section, we summarize the underlying ideas for this project. Firstly, we introduce a *relatively* well-known field of computer graphics: physics-based rendering. Next, we explore inverse rendering—a growing branch of computer graphics—where we aim to find the parameters of material models. The task of inverse rendering is *often* defined as an optimization problem. To solve this high-dimensional optimization problem efficiently, we introduce differentiable rendering, which provides derivatives and therefore allows gradient-based optimization techniques to successively improve the parameters with respect to a specified objective. Lastly, borrowing from the ideas we introduced, we will summarize our work and list our contributions.

Physics Based Rendering. Early work in computer graphics relied on phenomenological, often physically implausible, shading models and focused primarily on rendering images under direct, or purely diffuse illumination [48, 4, 49]. However, with more powerful hardware supporting modern ray tracing algorithms [43, 23], photo-realistic and physically accurate rendering has become not only feasible but de-facto standard in many fields, both within and outside of computer graphics.

The purpose of physics-based rendering is to generate visually accurate depictions of the world—capturing fascinating and often overlooked details that are bound by the physical laws of our daily experiences—by computational means using a virtual scene description. Photorealistic rendering is often easier said than done. However, with the research work accomplished in the last fifty years by the scientific community and industry, the created imagery is hard to separate from reality (Figure 1).

Figure 1: Reconstruction results for a surface BSDF model from synthetic data (left) and real-world alginate material (right). Dragon model by the user Delatronic [8], License: CC-BY.

Ray Tracing. Ray tracing [49, 47] is a well-known method to generate photo-realistic images. The idea is to send out rays from each pixel of a virtual camera and trace them throughout the scene. In their journey, rays get scattered by the objects defined in the scene. Eventually, every ray present in the scene provides color information for its corresponding pixel location. The combination of these pixels results in the final rendered image. This process is shown in Figure 2.

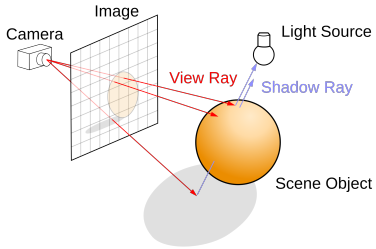

Figure 2: Ray Tracing Diagram by the user Henrik [58], CC BY-SA 4.0, via Wikimedia Commons [57]

The BSDF. The introduction of ray tracing brings another critical point that one must consider: how rays are scattered around in a scene. The ability to create images that are hard to separate from reality relies on accurate knowledge of the optical material parameters of objects in the scene.

For a moment, if we stop to observe our surroundings, we notice that light is scattered by objects in our environment. Consequently, if we could model scattering occurring at the surface of objects in mathematical terms, then we could apply those rules to the ray tracing procedure. This abstraction is often modeled through the *bidirectional scattering distribution function* (BSDF). Other than encoding the material's optical properties, the BSDF tells us how much energy is reflected from an incoming direction ω_i to an outgoing direction ω_o at the surface point p (Figure 3)[47]. The BSDF is denoted by $f(p, \omega_i, \omega_o)$.

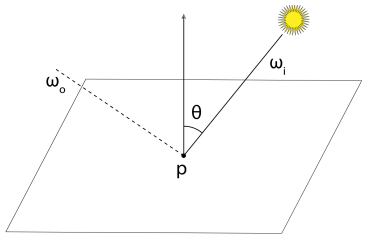

Figure 3: The Geometry of Surface Scattering, PBRT Chapter 1.2.5 Surface Scattering [47]

Microfacet models Modeling surface reflection and transmission often originates from the idea that rough surfaces can be represented as a collection of small microfacets. The scatter of light from a group of microfacets is statistically modeled by the microfacet-based BRDF models [47]. These models consist

of a representation of the *distribution* of facets and a BRDF that describes the scattering from microfacets. A well-known microfacet distribution function that is based on a Gaussian distribution of microfacet slopes is provided by Beckmann and Spizzichino [45]. Trowbridge and Reitz [52] also provided another microfacet distribution function, which is also then independently derived by Walter et al. [56] [1]. Microfacet models not only require the distribution of facets, but it also needs to account that some microfacets will be blocked by others (also known as *masking*) and some will be *shadowed* and therefore be hidden. Smith's masking-shadowing function [47] addresses these effects for microfacet models. Oren and Nayar [44] introduced a reflection model to describe rough surfaces by V-shaped microfacets using spherical Gaussian distribution. Another early model was developed by Torrance and Sparrow [51], in which they described surfaces as collections of perfectly smooth mirrored microfacets [47]. The Torrance-Sparrow model provides advantages since it is independent of the microfacet distribution and fresnel function and therefore it could be used for both conductors and dielectrics. More recently Walter et al. [56] reviewed microfacet theory and extended it to simulate transmission through rough surfaces. *Mitsuba 3* [15] renderer—which we utilize in this project—employs the work accomplished by Walter et al. [56] in its Roughdielectric BSDF implementation, which we make use of in this project.

Disney BSDF with integrated subsurface scattering In 2012, Burley et al. [5] proposed a general-purpose BRDF (also known as Disney Principled BRDF). They compared their new model with measured materials in terms of the microfacet models [51, 7, 56]. The Disney BRDF was originally based on the measured materials of Mitsubishi Electric Research Laboratories (MERL) [31]. Their BRDF blended metallic and dielectric BRDF models and included a microfacet reflection lobe with anisotropic roughness and an optional clearcoat reflection lobe. In 2014, Disney switched from ad-hoc lighting and shading to path-tracing where global illumination plays a vital role. Since path-traced global illumination mandates energy conversation, they extended their BRDF to a unified BSDF model where refraction and subsurface scattering effects were addressed. The unified Disney BSDF [14], extends the dielectric BRDF with integrated subsurface scattering and blends specular BSDF based on a new specular transmission parameter. For the specular BSDF, they follow the derivation of Walter et al. [56] and extend the microfacet reflection lobe to refraction. To include subsurface scattering in their dielectric BRDF, first, they changed the diffuse lobe into directional microsurface effects and non-directional (i.e., Lambertian) subsurface effects, then, they replaced the Lambertian portion of the diffuse lobe with either a diffusion or volumetric scattering model [14]. The original Disney BRDF was described with ten parameters. The extended version of their BRDF with integrated refraction and subsurface scattering is now represented with 12 parameters. *Mitsuba 3* [15] employs the work accomplished by Burley et al. [14] in its Principled BSDF implementation, which we also use

[1] In 2007, Walter et al. [56] introduced a microfacet distribution function, named GGX. Later it became clear that GGX is equivalent to a microfacet distribution that Trowbridge and Reitz [52] introduced in 1975. See also [46].

in this project.

The Rendering Equation Physics-based rendering has its origins in the seminal paper by Kajiya [19], formulating the now well-known rendering equation. Veach [54] introduced the path-integral form, which provides the theoretical foundation of modern Monte Carlo (MC) ray tracing methods [15, 42, 6, 47]. The rendering equation [19], also known as the light transport equation (LTE), is a recursive mathematical formulation of balanced light-energy distribution. We concentrate on the surface-only and spherical form of the LTE [19, 47, 18]:

$$L(p,\omega_o) = \underbrace{L_e(p,\omega_o)}_{Emission} + \underbrace{\int_{S^2} f(p,\omega_i,\omega_o) L_i(p,\omega_i) d\omega_i}_{Scattering} . \tag{1}$$

Here $L(p,\omega_o)$ describes the radiance (power per area, solid angle, and color channel) leaving a point p in direction ω_o due to light *emitted* at or *scattered* through p. The incident radiance results from recursively applying the LTE such that $L_i(p,\omega_i) = L(p',-\omega_i)$, where p' is a point directly visible from p in direction ω_i. Monte Carlo (MC) ray tracing approximates this recursive integral as a sum of randomly sampled scattering rays. Note that for convenience of notation, we assume the cosine term accounting for the projected solid angle is included in the BSDF, i.e, $f(p,\omega_i,\omega_o)$. The same concept has been extended to volumetric rendering, including scattering and attenuation in participating media; please refer to PBRT [47] for details.

Physics Based Inverse Rendering. As mentioned previously, physics-based rendering is a relatively mature branch of computer graphics. Physics-based rendering aims to generate photorealistic images from a well-defined scene description. On the other hand, inverse rendering is a relatively young branch of computer graphics. The recent developments in computer graphics and hardware have awoken significant interest and opened up the field of inverse rendering.

So far, we have summarized how a virtual scene, including material descriptions (BSDFs), can be rendered via ray tracing. As the name suggests, in inverse rendering, we have to think in the opposite manner. In inverse rendering, we aim to find scene parameters, such that the resulting image fulfills certain requirements. Specifically, our problem is finding the parameters of the material models (BSDFs), such as to match given reference images. Intuitively, one could maybe think this is not a complicated problem. For example, if we have an image of a red book, one would instinctively argue that the scene description should contain a book object with red material. However, we should remember that our goal is to describe reality in a physically accurate manner. The red book in the image is actually a combination of red pixels, and there could be several reasons why the image contains red pixels. In the simplest case, the red pixel might result from the red-colored diffuse material. However, it might also be the result of an indirect global illumination effect, such as a reflection of a red object on the computer. Figure 4 illustrates the complexity.

We describe the rendering of an image as a function of $I(x) = y$, where y

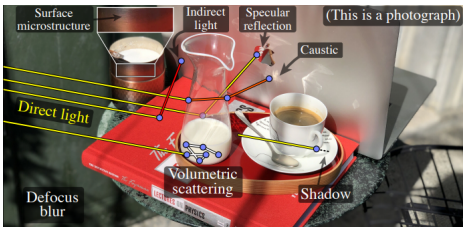

Figure 4: Image from *Introduction Physics-Based Differentiable Rendering*, by Zhao et al. [62].

is the rendered image, and x is the scene description. In inverse rendering, we aim to invert this function and estimate x.

Optimization Problem and Differentiable Rendering. In inverse rendering, our goal is to find some possible scene parameters x_{ref} from a rendered image y_{ref}. Since the given scene description x_{ref} is high-dimensional and complex—x_{ref} contains multiple parameters such as the objects, materials, and lights—and the generated image y_{ref} depends on the complexity of x_{ref}, there is *often* not a direct way to find the inverse function I^{-1} to retrieve parameters x_{ref} from an image y_{ref}.

Consequently, the task of inverse rendering is *often* defined as an optimization problem. We also make use of this approach. Consequently, the other available techniques are beyond the scope of this project [21, 38, 39]. Subsequently, defining the task as an optimization problem requires an objective function that quantifies the quality of the solution:

$$\min g(y(x)), \text{ s.t. } h(x) \leq 0, \tag{2}$$

where g defines the optimization objective (or loss) function on the rendered image y given scene parameters x, and h defines additional constraints (e.g., min/max parameter values). In order to solve this optimization problem more efficiently, differentiable rendering provides derivatives ($\delta g/\delta x$), which allows gradient-based optimization techniques to successively improve the parameters with respect to specified objective, see also [62]. In the simplest form, the objective takes the (scaled) squared L2 norm of the difference between the rendered and the reference image (i.e. mean square error; interpreting the sequence of N pixel values as a vector), thus $g(y) = ||y - y_{ref}||^2/N$.

The *dual buffer method* by Deng et al. [9] proposes an alternative objective function that is better suited to handle the noise introduced by MC rendering. Although we work with a slightly different BSDF model than their work, we still find this loss formulation useful in practice. Rather than taking one differentiable rendering step, the dual buffer method takes two independent rendering steps y_1, y_2, per iteration. Instead of the standard L2 loss, we now compute the loss in each iteration as $g(y_1, y_2) = (y_1 - y_{ref}) \cdot (y_2 - y_{ref})/N$.

We presented in Eq. (1) the Rendering Equation [19], where $L(p, \omega_o)$, L_e, and L_i referred to outgoing, emitted, and scattered radiance leaving a point p

in direction ω_o. The product of the BSDF f_s, and the scattered radiance L_i is integrated over the projected solid angles. In order to acquire the gradients required for optimization, we take the derivative of Eq. (1) with respect to the scene parameters x [41, 62]:

$$\begin{aligned}\delta_x L_o(p,\omega_o) = &\underbrace{\delta_x L_e(p,\omega_o)}_{Emission} \\ &+ \int_{S^2} [\underbrace{\delta_x L_i(p,\omega_i) f(p,\omega_o,\omega_i)}_{Transport} \\ &+ \underbrace{L_i(p,\omega_i)\delta_x f_x(p,\omega_o,\omega_i)}_{Material}] d\omega_i.\end{aligned} \tag{3}$$

This equation describes the scattering of derivatives analogous to the LTE. The individual terms are:

- Emission: differential radiance is emitted when the emitted radiance L_e depends on x.
- Transport: differential radiance *scatters* in the same way as normal radiance, according to the BSDF f.
- Material: surfaces with a parameter-dependent BSDF emit differential radiance proportional to incident radiance.

Note that along each ray, we find $\delta L_i(p,\omega_i) = \delta L(p',-\omega_i)$, analogous to the forward rendering. While in an abstract sense, the gradient of the objective function could be computed as $\delta g/\delta x = (\delta g/\delta y)(\delta y/\delta x)$, doing so would be computationally expensive due to the large size of $\delta y/\delta x$ (which can be thought of as a matrix of derivatives for each image pixel with respect to each scene parameter). Instead, the back-propagation method [41, 55] uses the (partial) derivative of the objective function $\delta g/\delta y$ as a source of "adjoint radiance" which is then traced from the virtual camera into the scene, scattered according to Eq. (3) and "collected" at the scene parameters.

In this work, we rely on the ADAM optimizer [22] to solve the parameter estimation problem. ADAM is a general-purpose method that applies gradient descent with momentum and an adaptive strategy to estimate the learning rate per parameter. As such it is well suited to non-linear optimization problems that may contain spurious local minima. Using a momentum strategy can prevent getting stuck in these local minima and increase the chances of finding a good solution.

So far, we have accumulated background information from the field of computer graphics. In the remainder of this thesis, we will mainly focus on material parameter estimation. For translucent materials, ray tracing methods, as well as material descriptions, have been extended to cover *volumetric scattering* in addition to surface effects. While both surface and volumetric BSDF parameters can be obtained from careful measurements of real-world materials, or instead formulated in a data-driven (rather than a parametric) framework, these approaches heavily rely on specialized laboratory equipment. However, this precise approach might not always be feasible or cost-effective. For example, research in

materials engineering might produce new compounds for specific applications, whereas aging and weathering effects might alter a material's appearance over time.

Recent work on differentiable and inverse rendering, on the other hand, has enabled the identification of optical material parameters by applying (gradient-based) optimization methods in order to match the rendered result to an object's real-world appearance. This optimization approach is particularly suited to estimating parameters from only a few reference images. In this work, we rely on the *Mitsuba 3* renderer [16], a publicly available, state-of-the-art differentiable rendering system, to solve material parameter estimation problems. However, as a research-oriented system, Mitsuba 3 requires a lot of domain-specific knowledge to operate. To facilitate the parameter estimation workflow, we implement a software tool that combines a gradient-based optimization algorithm with the necessary data management and rendering interface.

In the rest of this thesis, we first provide an overview of previous work on physics-based (inverse) rendering. The subsequent sections then detail the experimental and computational aspects of our proposed approach, discuss the implementation details, and present and analyze our results from both synthetic and real-world data. In summary, our contributions are:

- A software tool that incorporates the full power of the Mitsuba 3 renderer and various features to accomplish the task of inverse rendering.
- A gradient-based optimization procedure to acquire material properties from an image, including a careful evaluation of how optimization hyperparameters and rendering quality affect the results based on synthetic ground-truth comparisons.
- A workflow to increase the reproducibility of our results, encompassing both synthetic as well as real-world examples.

2 Related Work

The roots of Physics-based rendering can be traced back to Kajiya's groundbreaking paper [19], which presented the rendering equation that is now widely recognized. Later on, Veach [54] contributed to the field by developing the path-integral form, which established the theoretical basis for contemporary Monte Carlo (MC) ray tracing techniques [15, 42, 6, 47]. In order to arrive at a photorealistic rendering, an accurate description of how various materials interact with light must be formulated. Different material models have been introduced to account for the light-scattering effects from surfaces [56, 14, 33]. Similarly, models for participating media describe volumetric scattering effects as light passes through the material [47, 33]. Physics-based rendering itself (which we also refer to as *forward* rendering in this context) is concerned with computing accurate solutions to the rendering equation, given a scene description, including all materials.

Conversely, the question for *inverse* rendering is how to find an appropriate scene description in order to obtain a desired result. In particular, finding

the parameters of a scattering model corresponding to a real-world material, has been a long-standing research problem. While some previous work has addressed this problem in a data-driven way [31, 24, 50], inverse rendering methods in contrast, seek to determine the parameters of established material models, resulting in a more constrained search space that ensures the use of physically sound models. Early work on inverse rendering for material parameter estimation [11] used material dictionaries, aiming to approximate the appearance of real-world materials as a combination of dictionary entries. More recently, advances in differentiable rendering [41, 55, 40, 25, 61] have enabled inverse rendering applications by applying gradient-based optimization directly on the material parameters.

Initially, differentiable renderers have applied code-level automatic differentiation to find gradients of either the rendered image, or an objective function defined on that image, with respect to input parameters. In order to reduce the computational cost and memory footprint of automatic differentiation, analytic back-propagation formulations have been developed [41, 55]. One issue that has received a lot of attention in recent research, is that derivatives of pixel values (which are integrated according to the rendering equation) can contain discontinuous integrands if a silhouette edge moves across the pixel due to a change of scene parameters [25, 60, 28, 62]. As we are primarily concerned with material parameters of standard material models, these discontinuities do not *often* affect our results. Note that, however, this is not the case for geometry reconstruction-related experiments (shown in §5.2).

Modern ray tracing renderers use importance sampling to reduce noise due to MC sampling, based on either the material's scattering behaviour, the light sources, or combining both through multi-importance sampling. Consequently, similar sampling strategies have also been investigated for differentiable ray tracing. Zeltner et al. [60] analyzed numerous potential approaches and mathematically classified various estimators for differential light transport. Prior to the work by Vicini et al. [55], many techniques used statistically biased gradient estimation methods. Their work proposed a new back-propagation algorithm that runs in constant memory and linear computation time. Additionally, this method provides a way to handle highly specular materials such as smooth dielectrics and conductors. Our work builds upon their method, which is implemented in Mitsuba 3 [16].

Inverse Rendering for translucent material reconstruction. The work from Gkioulekas et al. [11] is one of the initial inverse rendering research regarding the reconstruction of homogeneous translucent materials. They introduced an optimization framework to measure the bulk scattering properties of homogeneous materials. Their primary focus was on the parameters in which the homogeneous materials are described by two scalar values and one angular function: scattering coefficient, absorption coefficient and phase function. Similar to our project, Gkioulekas et al. [11] also incorporated gradient-based optimization with MC rendering. However, they specifically used stochastic gradient descent for the optimization. In their approach, they define the material parameters as a convex linear combination of a *material dictionary.* The material

dictionary contained a variety of common materials, including liquid, solid, and publicly-available collections of tabulated phase functions. To recover scattering parameters of an unknown material using images, they define the problem of acquiring material properties as an appearance-matching framework. Subsequently, they differentiate the appearance-matching error with respect to the mixing weights and derive an efficient optimization scheme based on stochastic gradient descent and MC rendering. Note, however, unlike our approach, they do not use differentiable rendering.

Yang et al. [59] proposed another inverse rendering approach for heterogeneous translucent materials from a single input photograph. The proposed method can obtain the material distribution and estimate heterogeneous material parameters to render images similar to the provided reference image. Additionally, as in our second approach using Roughdielectric BSDF with homogeneous participating media (§3.1), they introduce an iteration process for optimizing the scattering coefficient and absorption coefficient, which they describe the sum as the extinction coefficient. Deng et al. [9] introduced another approach to reconstruct translucent objects using differentiable rendering. They extended previous methods using a bidirectional scattering-surface reflectance distribution function (BSSRDF) for translucent materials. Moreover, to handle the noise introduced by the BSSRDF integral, they proposed a dual-buffer method for evaluating the loss during optimization. In this work, we make use of the dual-buffer method and observe improved optimization convergence over standard error metrics (see our results in §5).

Geometry and material reconstruction. Mildenhall et al. [32] proposed a groundbreaking approach to reconstruct novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views [32]. Their proposed approach only requires a set of images with known camera positions. The proposed method (*Neural Radiance Fields (NeRF)*) offers to represent a scene using a fully connected neural network, where the input is a 5D coordinate spatial location (x, y, z) and viewing direction (θ, ϕ) and whose output is the volume density and view-dependent emitted radiance at that spatial location.

Many methods make use of the power of multi-layer perceptrons (MLPs) to model complex spatially-varying functions such as images and 3D scenes. However, the use of MLPs often comes at the cost of long training and inference times. On the contrary, voxel grid representations of scenes can give fast training and inference times; however, it is not able to reach the quality of MLPs without significant memory consumption. Karnewar et al. [20] proposed a method that allows fixed non-linearity (ReLU) on interpolated grid values that represents complex signals such as images or 3D scenes on regularly sampled grid vertices. Essentially, their method can be interpreted as NeRF-like [34, 32], however, without using any neural networks or sparse data structures. Their method represents the scene after an emissive volume with directionally varying emissions and, as in NeRF [32], the outgoing radiance is evaluated using ray marching model [20]. By differentiating the ray marching routine and optimizing spherical harmonic coefficients and volume density parameters, Karnewar

et al. [20] provides another method for 3D scene reconstruction.

In 2020, NeRFs [32] provided an extremely innovative approach, but its most significant problem was the costly training time of the underlying neural network. Mueller et al. [36] extended prior work in NeRFs [32] by reducing the costly training time with a versatile new input encoding scheme that allows the use of smaller neural networks. Consequently, their approach significantly reduce memory access operations during optimization. In essence, NeRF is a different approach to how we deal with geometry and material reconstruction, but with their award-winning SIGGRAPH 2022 paper, Mueller et al. [36] have made NeRFs more easily accessible and allowed to export acquired geometry. Munkberg et al. [37] proposed another efficient method for joint optimization of topology, materials, and lighting from multi-view image observations. However, their work provides an approach where the output is triangle meshes with spatially-varying materials and environment lighting instead of 3D representation encoded in neural networks. Their work also indicate an approach where constructed volumetric representation by NeRFs can be extracted in 3D models to use in 3D engines. Munkberg et al. [37] suggested a pipeline that has three steps. First they train a NeRF [32] model and extract the topology using Marching Cubes [27]. Then, they tweak the extracted mesh and learn material parameters using a differentiable renderer. Their approach connects the two worlds of NeRFs and differentiable rendering. Consequently, the proposed ideas in this project can be a part of the suggested pipeline to reconstruct geometry and materials.

3 Parameter estimation

Having established a foundational understanding of rendering and inverse rendering, our initial focus will be directed towards examining the *relevant* optical material parameters *crucial* for the reconstruction of translucent materials. Next, we will provide a brief overview of the objectives of this project and outline the essential requirements that our tool must satisfy. Lastly, we will introduce our method and specifically explore the experimental and computational parts of our proposed approach.

3.1 Material parameters

For surface-only materials, we use the *Disney Principled BSDF* implementation as described by Burley [14], focusing on the following parameters:

- base colour (albedo) c (can be textured),
- surface roughness α,
- specular transmission σ_s (blending between the BRDF and BTDF lobes), and the
- index of refraction η.

Note that we do not use secondary reflective lobes such as clear-coat materials or metallic reflections.

For volumetric rendering, we use Mitsuba's *Rough Dielectric BSDF* with a *homogeneous participating medium*. This model extends the surface BSDF by

- a volumetric albedo colour (texture) and
- an extinction coefficient σ_t describing the absorption as light traverses the material.

3.2 Problem statement

In this project, our goal is to estimate optical material parameters $x = \{c, \alpha, \sigma_s, \eta, \sigma_t\}$ in a given 3D scene, such that the rendered result best approximates one (or more) reference image(s).

As mentioned previously, we define this task as an optimization problem and introduce an objective function to quantify the quality of our solution. For completeness—as shown in Eq. (2)—we denote this task as:

$$\min g(y(x)), \text{ s.t. } h(x) \leq 0, \tag{4}$$

where g defines the optimization objective function on the rendered image y with given material parameters x, and h defines the additional constraints such as minimum and maximum parameter values.

In simplest scenarios we use the L2 norm of the difference between the rendered and the reference image, thus $g(y) = ||y - y_{ref}||^2/N$. As an alternative, we also utilize the dual buffer method introduced by Deng et al. [9] that better handles the noise introduced by MC rendering, therefore $g(y_1, y_2) = (y_1 - y_{ref}) \cdot (y_2 - y_{ref})/N$, where y_1 and y_2 are two independent rendering steps per iteration.

In order to achieve our goal, we require

- a (set of) reference image(s) and
- a Mitsuba scene file, including the initial guess for the material parameters, and virtual cameras corresponding to the reference images as input to our system.

Finally, we address this task by combining a differentiable renderer (Mitsuba 3 [16]) with a gradient-based optimization procedure, which we describe in the following section.

3.3 Method

We now direct our attention to solving the defined problem in Section 3.2. This section is divided into two parts: *experimental* and *computational*. In the experimental part, we are going to introduce our workflow to meet the requirements defined in Section 3.2. In the computational part, we will show our procedure to reach the goals that we defined in Section 3.2.

3.3.1 Scene and image acquisition

In this section, we discuss scene and image file acquisition. For validation tests where the Bunny [53] model is in use, we utilized the readily available Mitsuba

scene from their documentation [34] and modified the parameters we introduced in Section 3.1. However, for synthetic data, users may also construct a scene in a 3D computer graphics software tool such as Blender [6]. For example, for the Dragon [8] model test case in Section 5.1, we constructed a scene in Blender and exported it using Mitsuba's Blender Add-on [3]. For validation tests, we then rendered (using Mitsuba) acquired scenes to obtain corresponding reference images.

The real-world case is more complicated. In the imaging phase, it is important to have a controlled environment where all the scene parameters that could impact the resulting image, especially the scene geometry and light position(s), are known accurately.

We suggest two approaches, *first*, users may reconstruct the imaged environment in Blender. This task can get seriously complicated and requires relatively adequate skills in modeling. For the alginate [29] material test cases that we acquired from Prof. Stavric and colleagues at TU Graz, we only partially had this difficulty. We were provided by Prof. Ferschin (TU Wien) and his former masters' student Cheng Shi a "material scanner"—a small light-proof container with fixtures for camera and light placement—where we photographed the alginate samples. Fortunately, they also provided us with a corresponding 3D model of the material scanner that we used in Blender. However, we still had to approximate the model of the sample alginate materials, camera positions, and the radiance values of the lights in Blender. Please note, for the alginate test cases we used only one reference image per alginate specimen, and then we further processed those images using a Photoshop [1] script to remove the background [2].

Our *second* approach utilizes a photogrammetry tool Metashape [2]. Using Metashape, we not only acquired the geometry of the imaged bird-statue (first and third row in Fig. 12) but also the camera positions from photographs. Using Metashape [2], (1) we first loaded images and used the *Align Images* functionality, (2) then used the *Build Mesh* functionality, (3) next, exported mesh and camera locations with the X3D format, (4) and lastly, imported the X3D file into Blender. Once we had a scene in Blender, we utilized the Mitsuba Blender Add-on [3] to export and use it in our tool. We found our second approach useful specifically for *opaque* materials. As in our previous approach, we found background removal from the reference images useful. Please note, using this approach, we had to approximate light positions and radiance values using Mitsuba's constant emitter [33]. In the future, we would like to test our second approach using an environment map, which it could be acquired by photographing the environment of the object of interest.

Another technique we employed involved *naive* vertex position optimization that is supported by Mitsuba 3 and our tool. As shown in the second and fourth row of Fig. 12, from a scaled-sphere object the optimization recovers a *rough* bird statue. Geometry reconstruction on its own is a fascinating and challenging

[2]The interested reader may find the script with the name *background-remover-ps.js* file in our repository.

research topic, and we will omit the details in this work. However, the interested reader may find more about our naive approach in Appendix A. Lastly, although we did not utilize this approach, please note that the geometry reconstruction step can also be achieved with different NeRF procedures [32, 36, 20, 30].

3.3.2 Optimization

We are now ready to introduce the computational part of our approach, combining the differentiable renderer Mitsuba 3 [16] with an ADAM [22, 17] optimizer to estimate the material parameters.

Using our software tool (Fig. 5), we first load a virtual 3D scene file, which includes the initial material parameters x_0. We also load the (set of) reference image(s), y_{ref}, which will be used in the objective function (selecting either the L2 norm or the dual buffer method as introduced earlier). Next, we select the material parameters of interest (x_0), which get assigned to a newly initialized ADAM optimizer by our tool. Optionally, we also select the following optimization hyper-parameters (default values in braces):

- the *maximal number of iterations* for the optimization (100),
- the number of *samples per pixel (spp = 4)* for each rendered image,
- *loss tolerance* ϵ, where optimization stops if the loss $g < \epsilon$ (0.001),
- the *learning rate*, which adjusts the step size at each iteration (0.03),
- *minimum and/or maximum clamp values*, which define box constraints for specific parameters. For example, an RGB color value must be in the interval $[0, 1]$; by default, parameters are constrained, i.e. in $[0, 1]$.

Next, initializing $x_i = x_0$, we then run the following optimization loop for each camera pose (i.e. reference image):

1. Perform a differentiable rendering step with respect to x_i resulting in an image y_i.
2. Evaluate the objective function $g(y_i)$.
3. Back-propagate $\delta g/\delta y$ using Mitsuba 3, to obtain $\delta g/\delta x_i$.
4. Take an ADAM optimization step to find updated parameters $\tilde{x}_{i+1}$.
5. Ensure legal values for x_{i+1} by clamping $\tilde{x}_{i+1}$ using box constraints.
6. Update the scene with x_{i+1}.
7. Repeat until either the loss tolerance or the maximal iterations is reached.

To streamline the above-mentioned process, we propose a mini-tool with a GUI [3]. In its simplest form, users only need to load a *Mitsuba 3* [16] scene and a reference image and pick the proper material parameters to execute the optimization. After loading a scene, our tool picks the qualified integrator (see also §4.3), scene resolution, provides default hyperparameters, and presents supported differentiable scene parameters for optimization. At the end of the optimization, our tool not only provides visualizations similar to the Figures in §5 but also allows to export the obtained optimized parameter results.

[3] We expect the use of this tool for academic purposes. Consequently, the design of the GUI is constructed with simplicity. Thus the design and human-computer interaction related intricacies are beyond the scope of this project.

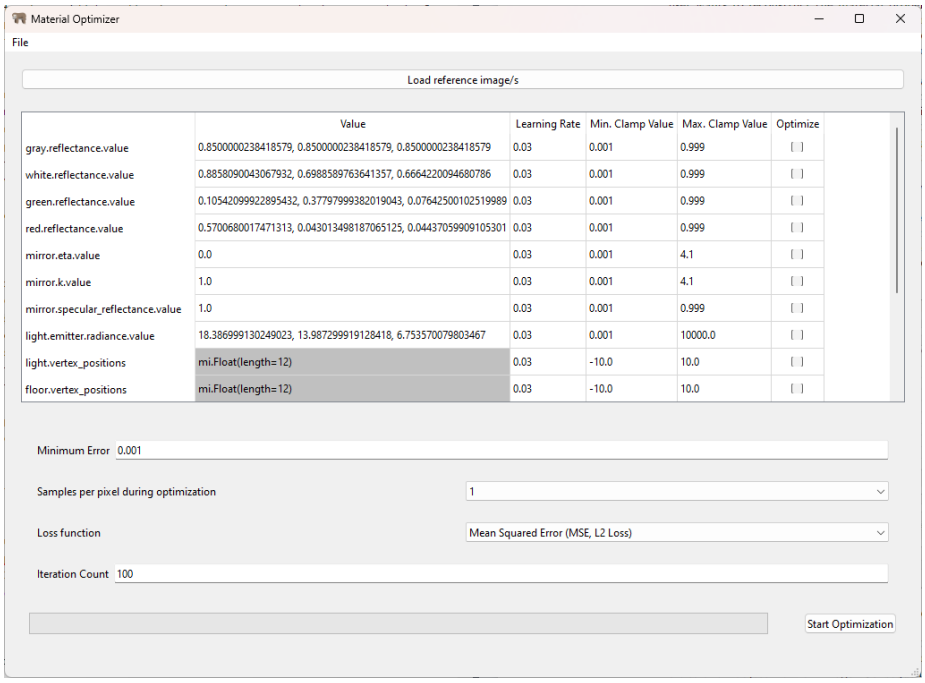

Figure 5: The main interface of our tool. The user first loads a scene (File menu), then proceeds from top to bottom: loading reference images, selecting optimization parameters, setting constraint values, choosing hyper-parameters, and finally executes the optimization process.

4 Implementation Details

In this section, we discuss the implementation details of our tool. First, we examine the architectural details of our system and discuss the input and output (I/O) capabilities of our tool. Lastly, we show the implementation of the computational part of our approach, discussed in Section 3.3.2.

4.1 Architecture

In this project, our focus primarily leaned on material reconstruction. Therefore design topics, which encompass not only architectural but also human-computer interaction related details, were given a lower priority during the development process. However, since we wanted to have a reusable and flexible system, we made sure that the components of the system were well divided. Consequently, the actions taken by the user are processed and visualized separately. Additionally, whenever a new finding emerged, the flexible design allowed easy changes without large refactorings.

Our tool implements a well know Model-View-Controller pattern [10] (Fig.6). The Model-View-Controller pattern divides the system into three components,

where the Model is the system's central component and contains the system's business logic. In our tool, the Model refers to *MaterialOptimizerModel*, and it contains the appropriate logic to accomplish the task of inverse rendering. The main tasks of *MaterialOptimizerModel* are the implementation of the necessary data management, and the optimization procedure. The *MaterialOptimizerView* utilizes appropriate visual components (e.g., PyQt6 components [26]) to effectively illustrate the data being received from the Model. The *MaterialOptimizerController* is essentially the coordinator between *MaterialOptimizerModel* and *MaterialOptimizerView*. Its main goal is to convert user input for the *MaterialOptimizerModel* and *MaterialOptimizerView*.

Figure 6: The UML class diagram of our tool, which implements a well know Model-View-Controller pattern [10], where the *MaterialOptimizerModel* contains the business logic, *MaterialOptimizerView* implements the visual components and *MaterialOptimizerController* coordinates interaction between the Model and View.

4.2 Input and Output Description

As mentioned previously (Fig. 3.2), we abstract the inputs of the system as a Mitsuba scene and a (set of) reference image(s). The system's output can be summarized into a (set of) file(s) containing the results from the optimization procedure.

The Mitsuba scene specification follows an XML file structure and the semantics of a scene description (e.g., objects, light positions, material properties, integrator, etc.) [35]. Using Mitsuba, our tool traverses nodes of the provided scene graph and returns a dictionary-like object, and prepares the appropriate configuration for the system—more on this in Section 4.3. The provided (set of) reference image(s) plays a vital role in the optimization procedure. The

reference image that corresponds to the sensor defined in the scene will be in use when evaluating the objective function. At the moment, our tool allows two formats for reference images: PNG and JPEG.

Currently, our tool supports four output formats. The (default) JSON file contains multiple key-value pairs of optimized scene parameters. An example JSON file output from optimizing the color of a white wall in a scene is shown in Listing 1.

```
{
    "gray.reflectance.value": "[[...]]",
    "white.reflectance.value": "[[0.94, 0.73, 0.68]]",
    "green.reflectance.value": "[[...]]",
    "red.reflectance.value": "[[...]]",
    "mirror.eta.value": [...],
    "mirror.k.value": [...],
    "mirror.specular_reflectance.value": [...],
    "light.emitter.radiance.value": "[[...]]",
    "light.vertex_positions": [...],
    "floor.vertex_positions": [...],
    "ceiling.vertex_positions": [...],
    "back.vertex_positions": [...],
    "greenwall.vertex_positions": [...],
    "redwall.vertex_positions": [...]
}
```

Listing 1: Exampe JSON file output from optimizing reflectance value of a white wall object. Note that each row refers to a scene parameter, where the first word is associated with an object, and the remaining words are associated with a specific scene attribute. E.g., white refers to the white wall in the scene and reflectance.value refers to the diffuse BSDF color assigned to it. For conciseness, we added dots for the values of other rows. The interested reader may find more about the scene parameters in the mitsuba documentation [33].

The other types of output formats provided by our tool correspond to more specific scene parameters. The output formats and their description is shown in Table 1 and an example output is shown in Figure 7.

Format	Description
`JSON`	Optimized scene parameters (default).
`PNG`	Optimized bitmap texture data.
`Volume (.vol)`	Grid-based volume data [33].
`NumPy Array (.npy)`	Multidimensional floating point arrays (e.g. vertex colors) [13].

Table 1: Provided output formats from our tool.

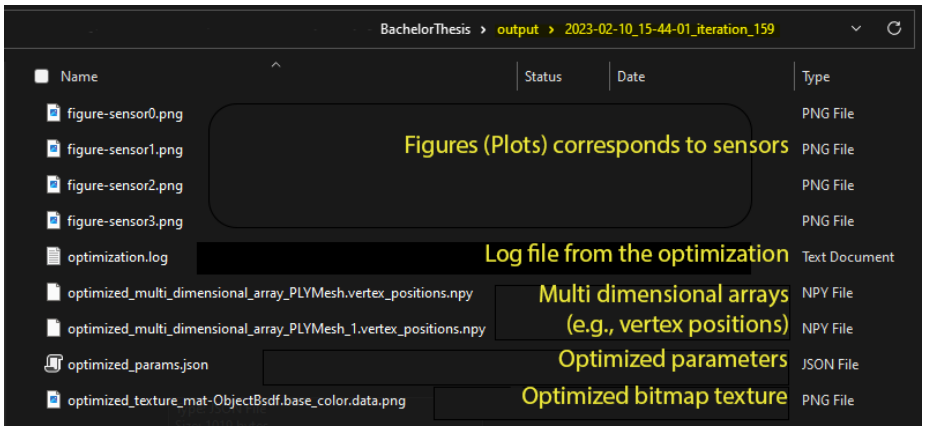

Figure 7: An example output from our tool. Note that the output directory contains the outputs from our tool named after the corresponding date and time of the optimization.

4.3 Implementation

In this section, we discuss the implementation details behind our tool. Please note that it is not feasible to examine the full implementation of our tool. Consequently, we mainly discuss the implementation of the optimization procedure we presented in Section 3.3.2. For the full implementation, we invite the reader to explore the attached ZIP folder or provided GitHub/GitLab repository.

Load a scene file. Whenever users click the "Import File" button, the View (*MaterialOptimizerView*) notifies the Controller (*MaterialOptimizerController*). The Controller then calls the following method:

```
1  def loadMitsubaScene(self):
2      try:
3          fileName = self.view.showFileDialog(XML_FILE_FILTER_STRING)
4          self.model.loadMitsubaScene(fileName)
5          self.model.resetSensorToReferenceImageDict()
6          self.model.setSceneParams(self.model.scene)
7          self.model.setInitialSceneParams(self.model.sceneParams)
8          self.model.setDefaultOptimizationParams(
9              self.model.initialSceneParams)
10         self.updateTable(self.model.initialSceneParams,
11             self.model.optimizationParams)
12     except Exception as err:
13         self.model.loadMitsubaScene()
14         self.view.showInfoMessageBox(...)
```

The *loadMitsubaScene(self)* method can be summarized in the following way: first, the Controller instructs the View to show a Window so the user can pick a scene file (*line 3*). Then, the Controller passes the selected scene file to the Model. The Model parses the XML file and selects an Integrator depending on the patterns found (*line 4*). For example, if the file contains a scene parameter that might introduce discontinuities, then our tool selects the *'prb_reparam'* integrator [33]. We achieve this by following the mitsuba documentation [33],

where they note the scene parameters that might introduce discontinuities. Consequently, our tool keeps a constant list of those values[4], and picks the appropriate integrator depending on the matched pattern of the available parameters. Next, the Controller instructs the Model to reset the data structure that contains a list of key-value pairs, where the key refers to sensors (i.e., camera positions) and the value to corresponding reference images (*line 5*). In *line 6*, the Model saves the scene parameters. Next, the Model creates a new data structure that contains the scene parameters supported in our tool (*line 7*). In *line 8*, the Model sets the default optimization parameters (e.g., the learning rate). Lastly, the Controller passes the new scene parameters and instructs the View to update its table (*line 10*). If an error occurs during these steps, the Controller instructs the View to show the affiliated error message (*line 16*) and instructs the Model the load the default mitsuba scene (i.e., the Cornell Box Scene [12], *line 15*).

Load s (set of) reference image(s). When the "Load reference image/s" button is clicked, the Controller is triggered by the View for the selection of reference images:

```
1  def loadReferenceImages(self):
2      try:
3          refImgFileNames = self.view.showFileDialog(
4              IMAGES_FILE_FILTER_STRING, isMultipleSelectionOk=True
5          )
6          # check length of reference images is equal to # of sensors
7          if len(refImgFileNames) != len(self.model.scene.sensors()):
8              raise RuntimeError(...)
9          readImgs = [
10             self.model.readImage(refImgFileName)
11             for refImgFileName in refImgFileNames
12         ]
13         # Create a dictionary where key: sensor, value: refImage
14         self.model.setSensorToReferenceImageDict(readImgs)
15         self.view.showInfoMessageBox(...)
16     except Exception as err:
17         logging.error(...)
18         self.view.showInfoMessageBox(...)
```

First, the View shows a dialog box to choose reference images (*line 3*). Then, the number of chosen images is checked whether they are equal to the sensors loaded in the scene (*line 7*). If not, an error is displayed to the user (*line 8*). Next, from the selected files, the Model constructs Bitmap Objects according to aspect ratio of the loaded images (*line 9*). The new resolution is computed as $(256 \times AspectRatio, 256)$. This restriction originates mainly from memory limitations. Next, from the constructed Bitmap objects, the Model assembles a new data structure that contains a list of key-value pairs, where the keys refer to sensors (i.e., camera positions) and the values to a corresponding reference image (*line 14*). Please note that currently, the order of the loaded reference images matters. Therefore, each reference image is assigned to a sensor sequentially (e.g., first sensor is mapped to the first reference image). Lastly, the Controller instructs the View informing the user that the reference images are loaded successfully (*line 15*). Similar to the *loadMitsubaScene(self)* method,

[4]The interested reader may find them in *constants.py* file in our repository.

the View displays the corresponding error message to the user in case of failure.

Optimization preparation. Whenever the "Start Optimization" button is clicked, the Controller calls the *optimizeMaterials()* method, which is shown here:

```
1  def optimizeMaterials(self):
2      # Precondition: (1) reference image/s is/are loaded, and
3      #               (2) at least one checked scene parameter
4      checkedRows = self.getCheckedRows()
5      if (self.model.sensorToReferenceImageDict is None
6          or len(checkedRows) <= 0):
7          self.view.showInfoMessageBox(...)
8          return
9
10     ... # for now we omit the details, please refer to next steps
```

First, the checked scene parameters (*line 3*) are gathered. Then it is controlled so that at least one reference image is loaded and one scene parameter is selected (*line 5*). If this is not the case, the optimization cannot begin, and the user is informed correspondingly.

If the preconditions are met, the Controller instructs the Model to prepare for optimization.

```
1  def optimizeMaterials(self):
2      # Precondition: (1) reference image/s is/are loaded, and
3      #               (2) at least one checked scene parameter
4      # The implementation is as in the previous sections
5      ...
6      # Omitting View dependent details...
7      opts, initImg = self.model.prepareOptimization(checkedRows)
8      # for now we omit the details, please refer to next steps
```

The Model prepares for the optimization by receiving selected scene parameters and, from them, initializes the ADAM [22, 17] optimizer. From the initial scene parameters, the Model makes the initial render of the loaded scene (*line 7*). The initialization of the ADAM optimizer occurs as follows:

```
1  def initOptimizers(self, params: list) -> list:
2      opt = mi.ad.Adam(   # note: imported mitsuba as mi
3          lr=0.2, params={k: self.sceneParams[k] for k in params}
4      )
5      opt.set_learning_rate(
6          {
7              k: self.optimizationParams[k][COLUMN_LABEL_LEARNING_RATE]
8              for k in params
9          }
10     )
11     return [opt]
```

First, the Model initializes the Mitsuba Adam object with an arbitrary learning rate and selected scene parameters (*line 2*). Next, the Model assigns the custom learning rates with the configured values (*line 5*).

As mentioned previously, the user is always provided with the default optimization function mean squared error:

```
1  def mse(self, image, refImage):
2      """L2 Loss: Mean Squared Error"""
3      return dr.mean(dr.sqr(refImage - image))
```

However, as shown in Figure 5, the user is provided with a dropdown menu where the following optimization functions can be chosen: mean squared error,

brightness independent mean squared error, dual buffer method [9], mean absolute error, and mean bias error. In the following, we show the dual buffer method implementation (other implementations can be found in the *MaterialOptimizerModel*):

```
    def dualBufferError(self, image, image2, refImage):
        """
        Loss Function mentioned in: Reconstructing Translucent Objects Using Differentiable Rendering, Deng et al.
        ...
        """
        return dr.mean((image - refImage) * (image2 - refImage))
```

We omit the details of the implementation hyperparameters. However, the reader may think that the implementation has many similarities to the *loadReferenceImages()* method. Whenever an action in the View occurs, then the Controller is triggered; thus, the Model is updated with the appropriate hyperparameter values. The interested reader may find the implementation details in the *onMinErrLineChanged(), onIterationCountLineChanged(), onSamplesPerPixelChanged()* methods.

Optimization loop. When the "Start Optimization" button is clicked, the following method is called by the Controller:

```
def optimizeMaterials(self):
    # Precondition: (1) reference image/s is/are loaded, and
    #               (2) at least one checked scene parameter
    # The implementation is as in the previous sections
    ...
    lossHist, sceneParamsHist = self.model.optimizationLoop(
        opts, lambda x: self.view.progressBar.setValue(x)
    )
    # Omitting View dependent details...
```

If the preconditions are met, the Controller instructs the Model to prepare for optimization. Next, the Model is called for the optimization loop:

```
def optimizationLoop(self, opts: list, setProgressValue: callable = None):
    lossHist = []; sceneParamsHist = []
    for it in range(self.iterationCount):
        if setProgressValue is not None:
            setProgressValue(int(it / self.iterationCount * 100))
        total_loss = 0.0
        for sensorIdx, sensor in enumerate(self.scene.sensors()):
            loss = self.computeLoss(sensor=sensor, seed=it)
            # Backpropagate through the rendering process
            dr.backward(loss)
            self.updateAfterStep(opts, self.sceneParams)
            total_loss += loss[0]

        # update loss and scene parameter histograms
        sceneParamsHist.append(
            self.createSubsetSceneParams(
                self.sceneParams,
                SUPPORTED_MITSUBA_PARAMETER_PATTERNS,
            )
        )
        lossHist.append(total_loss)
        if total_loss < self.minError:
            break
    return lossHist, sceneParamsHist
```

The *optimizationLoop()* method receives two inputs, the *opts* refers to a list of ADAM [22] optimizers initialized previously and a function *setProgressValue*

that indirectly updates the progress. The output of this function is two lists; *lossHist* contains the loss during optimization, and *sceneParamsHist* contains scene parameters at each iteration step. The optimization loop begins with the selected amount of iteration counts (*line 3*). Then the progress bar is updated with the *setProgressValue* (*line 5*). Next, for each defined sensor, the Model computes the loss with the chosen loss function (*line 8*), where the optimization function receives two inputs, firstly the noisy differentiable rendering of the scene and, secondly, the reference image. Then, Dr.Jit [17], which is the numerical foundation of Mitsuba 3, backpropagates through the rendering process with the computed loss (*line 10*) and gradients. Note that Dr.Jit backpropogates gradients from the provided differentiable Dr.Jit Array. Next, the ADAM optimizer takes an optimization step, then the Model ensures legal values for the updated parameters using box constraints and updates the scene with the new values (*line 11*). Additionally, during these steps, we keep track of the loss (*line 12*). After completing the loop for each camera sensor, we update the loss and scene parameter histograms (*line 14, 21*). The optimization repeats these steps until either the loss tolerance or the maximal iterations are reached.

5 Evaluation

In this section, we examine results produced using our software tool and optimization approach. In the first part of this section, we evaluate results from synthetic data, where ground-truth solutions are available for comparison. In the second part, we show results for real-world data. Please note that in Appendix B, we provide a *comprehensive* presentation of synthetic data results and analysis to illustrate the thought process that led to our conclusions on the reconstruction of translucent materials.

5.1 Validation tests

In this section, we examine material reconstruction from synthetic data. These tests start from a virtual scene, with given ground-truth parameters. The optimization must then recover the correct material parameters from a dark and opaque initial guess. First, in Fig. 8, we show successful results using the method described in Section 3.3.2. Table 2 also shows the corresponding initial and optimized parameter values and optimization hyperparameters.

In all cases, we observe that the dual buffer method [9] is extremely useful in reconstructing the object's material properties, even when provided with *less* restrictive constraints. Conversely, using the standard (single-image) L2 error, we *often* require more restrictive constraints to prevent some parameters from diverging. Consequently, in this project, we *primarily* focus on translucent material reconstruction using the dual buffer method.

Furthermore, we observe that noise inherent to MC sampling affects the optimization procedure, and a sufficient number of samples per pixel (*spp*) is required to obtain good convergence. We also use box constraints on certain

parameters, i.e. clamping to minimum or maximum values to prevent some parameters from moving to physically implausible values[5]. Especially the index of refraction (η) often suffers from these issues. We believe this is due to total internal reflection introducing discontinuous jumps in light propagation paths. Another useful strategy to resolve these issues is to split the optimization process into two parts: first we optimize a group of parameters that work well together (for instance excluding η). We then continue from the optimized values from the first part and include all parameters. Results using this procedure are shown in the last two rows of Fig. 8.

Having established that our method is capable of recovering ground-truth material parameters, we now show additional comparisons and discuss potential pitfalls during translucent material reconstruction in a short ablation study, Fig. 9 and Table 3. We compare the dual buffer method (2×8 spp) to the single-image L2 error metric (1×16 spp) on the Stanford bunny using the Principled BSDF (first row in Fig. 8 and 9 respectively). While both approaches result in acceptable visual and numeric results, the single-image version, however, required additional constraints. Most notably, we clamp η to a maximum of 1.55 to enable convergence. This restriction was *not* necessary when using the dual buffer method. Note that the minimum loss in the L2 error was 0.005608 (in iteration 121), whereas the dual buffer version obtained 0.0000401 (in the 95th iteration).

On our second test case, the Dragon using a surface BSDF, we compare the optimization behaviour for different numbers of reference images (4 vs. 1) and samples per pixel (4×16 spp vs. 1×8 spp), as well as relaxing the minimum clamp value for the specular transmission (σ_s) parameter. As shown in Fig. 9 (second row), the lower sample count results in a visually slightly worse appearance after parameter optimization. However, as shown in Table 3 (second row), the numerical results are not satisfactory. Additionally, as shown in Fig. 10, the single-image optimization encodes a lot of appearance details into the texture

[5]The corresponding minimum and maximum clamp values can always be found in the last column (*opt. params*) of resulting Tables.

Case	parameters	initial	optimized	reference	hyperparam.	opt. param.
Bunny (Principled)	c	0.01, 0.01, 0.01	0.4192, 0.8531, 0.9990	0.412, 0.824, 0.999	*spp*=8, DBM	Default
	α	0.5	0.001	0.01		
	σ_s	0.02	0.9184	0.9		
	η	1.54	1.4737	1.49		
Dragon (Principled)	c	bitmap	bitmap	bitmap	*spp*=16, ic = 200, DBM	Default
	α	0.7	0.0022	0.001		
	σ_s	0.1	0.9749	1		min=0.11
	η	1.64	1.5135	1.49		Default
Bunny (Rough dielectric)	c	0.01, 0.01, 0.01	0.4810, 0.8495, 0.9990	0.412, 0.824, 0.999	*spp*=16, min. err.=0.0001 DBM	Default
	α	0.5	0.001	0.01		
	σ_t	0.98	0.4841	0.4		
	η	1.544	1.5169	1.49		max=1.55
Dragon (Rough dielectric)	c	varying volume	varying volume	varying volume	*spp*=4, DBM	Default
	α	0.7	0.0098	0.01		
	σ_t	0.98	0.349	0.4		
	η	1.544	1.4869	1.49		

Table 2: Results corresponding to Figure 8; DBM = dual buffer method, ic = iteration count. Unless specified otherwise, default values are used. Please note that the optimization results that are shown in the last row are achieved by separating the optimization procedure into two parts.

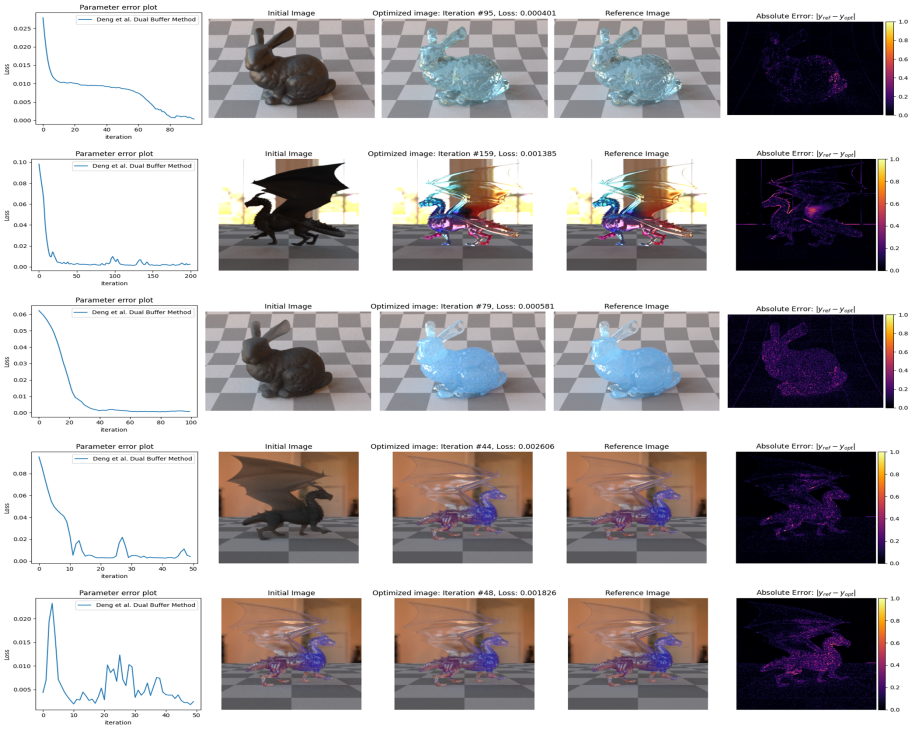

Figure 8: We successfully optimize material parameters for the Principled (first and second row) and Rough dielectric BSDF (third to fifth row). Columns: (1) convergence plot, (2) initial state, (3) optimized result, (4) reference image, and (5) the absolute error between the reference and optimized image. For corresponding parameter values please refer to Table 2.

Case	parameters	initial	optimized	reference	hyperparam.	opt. param.
Bunny (Principled)	c	0.01, 0.01, 0.01	0.4119, 0.8450, 0.9990	0.412, 0.824, 0.999	*spp*=16, ic=200	Default
	α	0.5	0.001	0.01		
	σ_s	0.02	0.9002	0.9		
	η	1.54	1.4973	1.49		max=1.55
Dragon (Principled)	c	bitmap	bitmap	bitmap	*spp*=8, DBM	Default
	α	0.7	0.1266	0.001		
	σ_s	0.1	0.001	1		
	η	1.64	1.8079	1.49		
Bunny (Rough dielectric)	c	0.01, 0.01, 0.01	0.5011, 0.8844, 0.9990	0.412, 0.824, 0.999	*spp*=8, DBM	Default
	α	0.5	0.0089	0.01		
	σ_t	0.98	0.5991	0.4		
	η	1.544	1.6042	1.49		
Dragon (Rough dielectric)	c	varying volume	varying volume	varying volume	*spp*=16, DBM	Default
	α	0.7	0.3065	0.01		
	σ_t	0.98	0.2705	0.4		
	η	1.544	1.5499	1.49		max=1.55

Table 3: Results corresponding to Figure 9; DBM = dual buffer method, ic = iteration count. Unless specified otherwise, default values are used.

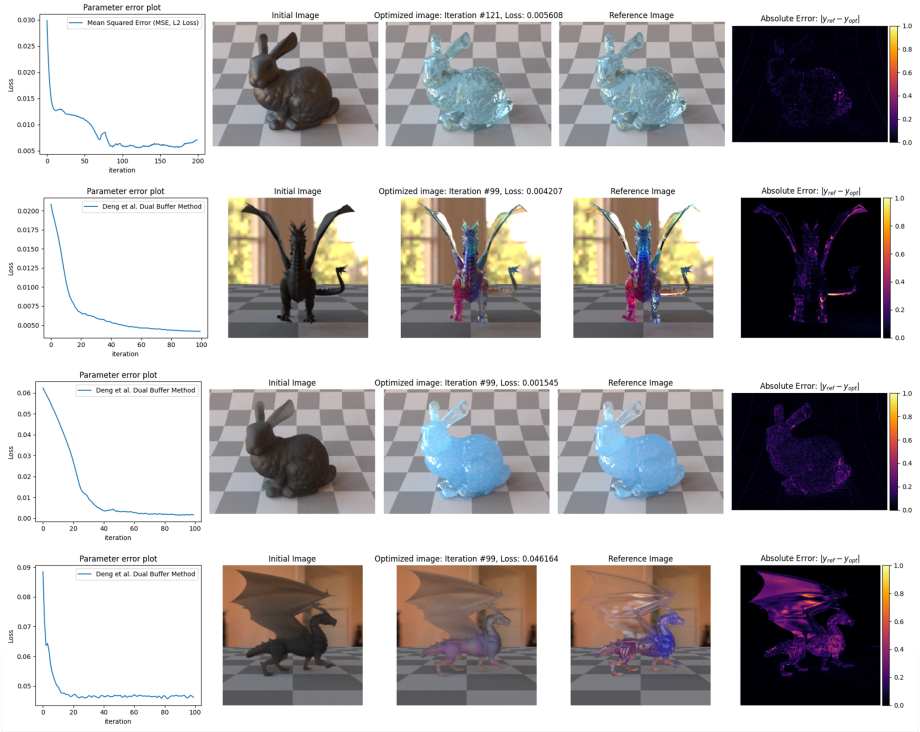

Figure 9: Results from our comparisons with different optimization strategies and hyper-parameters for the Principled (rows (1), (2)) and Rough dielectric (rows (3), (4)) BSDF Please see also Table 3.

image. Using multiple reference images, on the other hand, allows for improved texture recovery (note that the unused areas in the texture will always remain unchanged during optimization).

Using the volumetric material model, we again test the influence of noise on the optimization procedure. In the third row of Figure 9, we observe visually acceptable results, whereas numerical results degrade when reducing the samples from 16 spp to 8. In this experiment, the index of refraction η initially increases (moving away from the expected reference value), and subsequently, the extinction coefficient (σ_t) remains not recovered accurately.

Finally, applying the volumetric material to the more complex Dragon scene in the fourth row of Fig. 9, we attempt to optimize all parameters of interest simultaneously (as opposed to the successful case presented earlier, where we optimize in two stages). Interestingly the material's roughness (α) fails to reduce sufficiently from an initially high value, resulting in a visually noticeable mismatch between the optimization result and the reference. Note that both in this example and in our successful result, we use multiple reference images. Consequently, in some cases, we suggest separating the optimization procedure

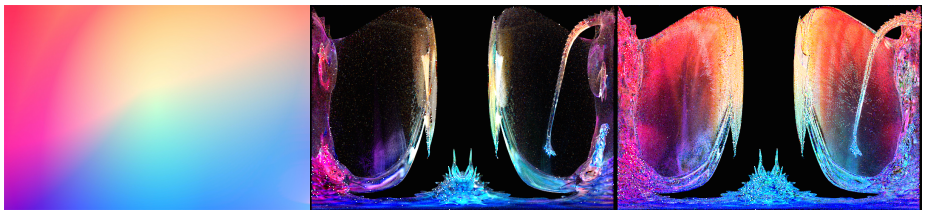

Figure 10: Texture used in the Dragon (Principled) test case. Left: reference bitmap; middle: optimized result using a single camera; right: optimized result from four cameras.

into two parts, as discussed earlier, to acquire acceptable results, as shown in the last two rows of Figure 8.

5.2 Real-World applications

In this section, we perform material reconstruction from real-world data. In particular, we are interested in acquiring optical material properties for two specimens made of a novel alginate material [29]. As these alginate specimens are relatively thin, we use the *Principled* BSDF [5, 14] material model, which has proved easier to optimize in the test cases discussed previously. Figure 11 shows results for both (green and blue) translucent alginate specimens. We again employ the strategy to split the optimization into two parts for both results. We first only allow one constant RGB colour, and in the second stage extend the optimization to a bitmap texture image. We obtain visually acceptable results, even though the virtual geometry is not perfectly matched to the real-world images, resulting in errors along the outer edge of the specimens, seen in the absolute error images in Fig. 11. Table 4 summarizes the numerical results corresponding to Fig. 11.

For both cases, we apply minimum and maximum clamp values for the α and σ_s parameters, that visually and numerically refers to a translucent material [33]. Without these additional constraints, we obtained visually identical results, albeit with physically unreasonable parameter values. Therefore we include

Case	parameters	initial	optimized	hyperparam.	opt. param.
Alginate (Green)	c	0.1 0.1 0.1	bitmap	*spp*=8, DBM	Default
	α	0.7	0.6999		max=0.7
	σ_s	0.1	0.4503		min=0.4
	η	1.54	1.2567		Default
Alginate (Blue)	c	0.1 0.1 0.1	bitmap	*spp*=8, DBM	Default
	α	0.7	0.3828		max=0.6
	σ_s	0.1	0.5132		min=0.5
	η	1.54	1.3406		Default

Table 4: Results for the real-world alginate specimens.

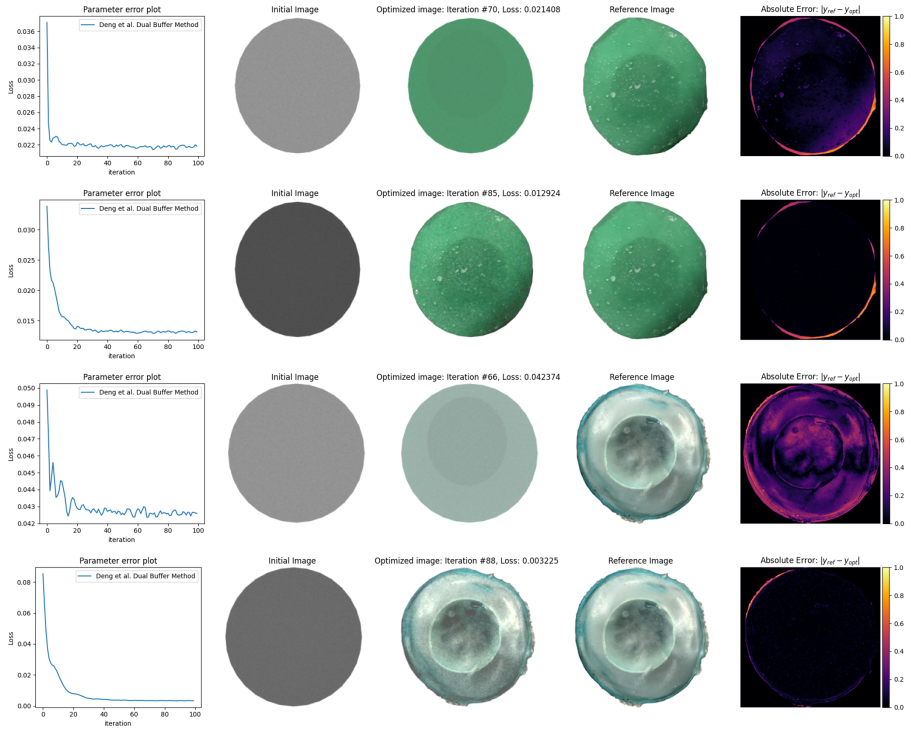

Figure 11: Results from the green (first and second row) and blue (third and fourth row) alginate [29] materials. Note that in both cases Principled [5, 14] BSDF is in use. For corresponding numerical results please refer to Table 4.

these constraints in order to obtain a plausible *translucent* material result.

Finally, in Fig. 12 and Table 5, we also show the results of a more complex opaque object. In Fig. 12, the first row showcases results utilizing the geometry obtained through Metashape [2], where only the c, *specular*, and α parameters of the object were optimized. The second row features results using a (scaled)

Case	parameters	initial	optimized	hyperparam.	opt. param.
1. Row	c	bitmap	bitmap	Default	Default
	specular	0.5	0.4279		
	α	0.5	0.0732		
2. Row	c	bitmap	bitmap	ic=200	Default
	vertex_positions	scaled sphere	shape		lr=0.0008, min/max=(-0.875, -0.933, 9.320)/(0.586, 0.723, 10.944)
3. Row	c	bitmap	bitmap	Default	Default
	specular	0.5	0.999		
	α	0.5	0.1193		
	vertex_positions	reconstructed	shape		lr=0.0008, min/max=(-0.875, -0.933, 9.320)/(0.586, 0.723, 10.944)
4. Row	c	bitmap	bitmap	ic=200	Default
	specular	0.5	0.999		
	α	0.5	0.001		
	vertex_positions	scaled sphere	shape		lr=0.0008, min/max=(-0.875, -0.933, 9.320)/(0.586, 0.723, 10.944)

Table 5: Results from Fig. 12. Note that clamp values refer to the min/max value of the object's bounding box from *(1)* in Fig. 12; lr = learning rate.

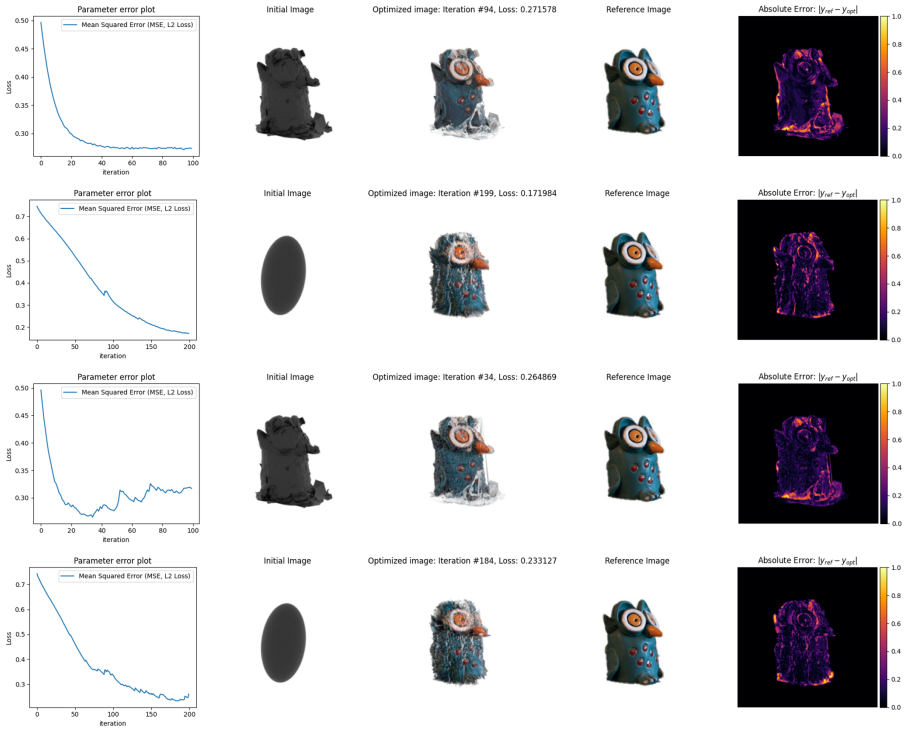

Figure 12: Results from a bird statue. Rows: (1) Material optimization using the reconstructed shape from Metashape, (2) Geometry and color optimization. (3) Material optimization as in *(1)*, and vertex position optimization. (4) Geometry and material optimization. See also 5.

sphere object, with optimization of c and *vertex_positions* parameters. The third row utilizes the reconstructed geometry as in the first row, but with the addition of *vertex_positions* optimization. The fourth row uses a (scaled) sphere object as in the second row but includes optimization of additional parameters that may cause discontinuities, such as *specular*, and α parameters.

In conclusion, we acknowledge the complexity of real-world material and geometry reconstruction and emphasize the importance of a comprehensive data acquisition process that accounts for various factors, including lighting, image acquisition, object and light positions, and geometry reconstruction. However, as shown in Fig. 1, with accurate measurements, multiple reference images, and thorough experimentation, material and geometry reconstruction can be successfully achieved.

6 Conlusion

In this project, we describe a material reconstruction pipeline built upon the Mitsuba 3 differentiable renderer. Our software tool is capable of reconstructing material properties from a scene description file and multiple images. The main focus of this project is on translucent material reconstruction, which we validated using synthetic data and demonstrated on real-world specimens. We test our approach on both a surface-only *Principled* BSDF, as well as a volumetric *Rough dielectric* BSDF with homogeneous participating media.

One difficulty we observed was the complexity of the scene and image file acquisition. The nature of real-world data acquisition can be extremely complex since the measurement process requires a controlled environment and either manual or automated, geometry reconstruction prior to material identification. Regarding translucent materials, we observed that the noise introduced by MC sampling affected the parameter estimation procedure. Employing the dual buffer method noticeably improved our results. Our analysis revealed that certain parameters that introduce discontinuities in the scattering behaviour, such as the index of refraction, caused difficulties in achieving accurate convergence. In some cases, we found that two-stage optimization was necessary. Especially when optimizing a (surface or volumetric) texture for the material's albedo, these highly localized parameters might have a stronger influence than global material parameters, which may result in inaccurate reconstruction. Furthermore, our findings indicated that in certain scenarios, additional constraints, such as imposing a maximum clamp value on some parameters, were necessary. We believe that this limitation stems from slight imprecisions in the image acquisition and modelling phase.

In the future, we aim to improve our geometric modelling and image acquisition procedures, which will allow for a more accurate representation of physics-based reality with well-established material models. Based on the evidence obtained from our results, we conclude that our approach will be beneficial for different translucent material reconstruction tasks in various applications.

References

[1] Adobe Inc. Adobe photoshop.

[2] AgiSoft. Agisoft metashape standard, 2022. https://www.agisoft.com/.

[3] <baptiste.nicolet@epfl.ch>, B. N. Mitsuba blender add-on. `https://github.com/mitsuba-renderer/mitsuba-blender`, 2022.

[4] Blinn, J. F. Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph. 11*, 2 (jul 1977), 192198.

[5] Burley, B. Physically-based shading at disney.

[6] Community, B. O. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2022.

[7] Cook, R. L., and Torrance, K. E. A reflectance model for computer graphics. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1981), SIGGRAPH '81, Association for Computing Machinery, p. 307316.

[8] Delatronic. Dragon. `https://blendswap.com/blend/15891`, August 21, 2015. [Online; accessed January 30, 2023].

[9] Deng, X., Luan, F., Walter, B., Bala, K., and Marschner, S. Reconstructing translucent objects using differentiable rendering. In *ACM SIGGRAPH 2022 Conference Proceedings* (New York, NY, USA, 2022), SIGGRAPH '22, Association for Computing Machinery.

[10] Freeman, E., Freeman, E., Bates, B., and Sierra, K. *Head First Design Patterns*. O' Reilly & Associates, Inc., 2004.

[11] Gkioulekas, I., Zhao, S., Bala, K., Zickler, T., and Levin, A. Inverse volume rendering with material dictionaries. *ACM Trans. Graph. 32*, 6 (nov 2013).

[12] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. Modeling the interaction of light between diffuse surfaces. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1984), SIGGRAPH '84, Association for Computing Machinery, p. 213222.

[13] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[14] Hill, S., McAuley, S., Burley, B., Chan, D., Fascione, L., Iwanicki, M., Hoffman, N., Jakob, W., Neubelt, D., Pesce, A., and Pettineo, M. Physically based shading in theory and practice. In *ACM SIGGRAPH 2015 Courses* (New York, NY, USA, 2015), SIGGRAPH '15, Association for Computing Machinery.

[15] Jakob, W. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[16] Jakob, W., Speierer, S., Roussel, N., Nimier-David, M., Vicini, D., Zeltner, T., Nicolet, B., Crespo, M., Leroy, V., and Zhang, Z. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org.

[17] Jakob, W., Speierer, S., Roussel, N., and Vicini, D. Dr.jit: A just-in-time compiler for differentiable rendering. *Transactions on Graphics (Proceedings of SIGGRAPH) 41*, 4 (July 2022).

[18] Jarosz, W. *Efficient Monte Carlo Methods for Light Transport in Scattering Media*. PhD thesis, UC San Diego, September 2008.

[19] Kajiya, J. T. The rendering equation. *SIGGRAPH Comput. Graph. 20*, 4 (aug 1986), 143150.

[20] Karnewar, A., Ritschel, T., Wang, O., and Mitra, N. Relu fields: The little non-linearity that could. In *ACM SIGGRAPH 2022 Conference Proceedings* (New York, NY, USA, 2022), SIGGRAPH '22, Association for Computing Machinery.

[21] Kato, H., Ushiku, Y., and Harada, T. Neural 3d mesh renderer, 2017.

[22] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization, 2014.

[23] Lattner, C., and Adve, V. LLVM: A compilation framework for lifelong program analysis and transformation. pp. 75–88.

[24] Lawrence, J., Ben-Artzi, A., DeCoro, C., Matusik, W., Pfister, H., Ramamoorthi, R., and Rusinkiewicz, S. Inverse shade trees for non-parametric material representation and editing. *ACM Trans. Graph. 25*, 3 (jul 2006), 735745.

[25] Li, T.-M., Aittala, M., Durand, F., and Lehtinen, J. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph. 37*, 6 (dec 2018).

[26] Limited, R. C. Qt for python. `https://doc.qt.io/qtforpython/index.html`, 2023. [Online; accessed 17-January-2023].

[27] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (aug 1987), 163169.

[28] Loubet, G., Holzschuch, N., and Jakob, W. Reparameterizing discontinuous integrands for differentiable rendering. *Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38*, 6 (Dec. 2019).

[29] Marjanovi, I., amec, E., Vaatko, H., and Stavri, M. Alginate in architecture : An experimental approach to the new sustainable building material. In *Art and Science Applied: Experience and Vision*, . and . , Eds., vol. 2 of *SmartArt*. , , , 2022, ch. 21, pp. 394–406. 21.

[30] Matthew Tancik*, Ethan Weber*, E. N. Nerfstudio: A framework for neural radiance field development, 2022.

[31] Matusik, W., Pfister, H., Brand, M., and McMillan, L. A data-driven reflectance model. *ACM Trans. Graph. 22*, 3 (jul 2003), 759769.

[32] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM 65*, 1 (dec 2021), 99106.

[33] Mitsuba3. Mitsuba 3: Plugin reference. `https://mitsuba.readthedocs.io/en/stable/src/plugin_reference.html`, 2023. [Online; accessed 11-January-2023].

[34] Mitsuba3. Mitsuba 3: Read the docs. `https://mitsuba.readthedocs.io/en/stable/`, 2023. [Online; accessed 14-February-2023].

[35] Mitsuba3. Mitsuba 3: Scene xml file format. `https://mitsuba.readthedocs.io/en/stable/src/key_topics/scene_format.html`, 2023. [Online; accessed 06-January-2023].

[36] Müller, T., Evans, A., Schied, C., and Keller, A. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph. 41*, 4 (jul 2022).

[37] Munkberg, J., Hasselgren, J., Shen, T., Gao, J., Chen, W., Evans, A., Mueller, T., and Fidler, S. Extracting Triangular 3D Models, Materials, and Lighting From Images. *arXiv:2111.12503* (2021).

[38] Nguyen-Phuoc, T., Li, C., Theis, L., Richardt, C., and Yang, Y.-L. Hologan: Unsupervised learning of 3d representations from natural images, 2019.

[39] Nguyen-Phuoc, T., Richardt, C., Mai, L., Yang, Y.-L., and Mitra, N. Blockgan: Learning 3d object-aware scene representations from unlabelled images, 2020.

[40] Nimier-David, M., Müller, T., Keller, A., and Jakob, W. Unbiased inverse volume rendering with differential trackers. *ACM Trans. Graph. 41*, 4 (jul 2022).

[41] Nimier-David, M., Speierer, S., Ruiz, B., and Jakob, W. Radiative backpropagation: An adjoint method for lightning-fast differentiable rendering. *Transactions on Graphics (Proceedings of SIGGRAPH) 39*, 4 (July 2020).

[42] Nimier-David, M., Vicini, D., Zeltner, T., and Jakob, W. Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38*, 6 (Dec. 2019).

[43] NVIDIA, Vingelmann, P., and Fitzek, F. H. Cuda, release: 10.2.89, 2020.

[44] Oren, M., and Nayar, S. K. Generalization of lambert's reflectance model. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1994), SIGGRAPH '94, Association for Computing Machinery, p. 239246.

[45] Petr, B., and Andre, S. *The Scattering of Electromagnetic Waves from Rough Surfaces*. Pergamon Press, 1963.

[46] Phar, M. Lets stop calling it ggx. `https://pharr.org/matt/blog/2022/05/06/trowbridge-reitz`, 2023. [Online; accessed 22-February-2023].

[47] Pharr, M., Jakob, W., and Humphreys, G. *Physically Based Rendering: From Theory to Implementation (3rd ed.)*, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, Oct. 2016.

[48] Phong, B. T. Illumination for computer generated pictures. *Commun. ACM 18*, 6 (jun 1975), 311317.

[49] Shirley, P., and Marschner, S. *Fundamentals of Computer Graphics*, 3rd ed. A. K. Peters, Ltd., USA, 2009.

[50] Tongbuasirilai, T., Unger, J., Kronander, J., and Kurt, M. Compact and intuitive data-driven brdf models. *The Visual Computer 36*, 4 (2019), 855872.

[51] Torrance, K. E., and Sparrow, E. M. Theory for off-specular reflection from roughened surfaces∗. *J. Opt. Soc. Am. 57*, 9 (Sep 1967), 1105–1114.

[52] Trowbridge, T. S., and Reitz, K. P. Average irregularity representation of a rough surface for ray reflection. *J. Opt. Soc. Am. 65*, 5 (May 1975), 531–536.

[53] Turk, G., and Levoy, M. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1994), SIGGRAPH '94, Association for Computing Machinery, p. 311318.

[54] Veach, E. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[55] Vicini, D., Speierer, S., and Jakob, W. Path replay backpropagation: Differentiating light paths using constant memory and linear time. *Transactions on Graphics (Proceedings of SIGGRAPH) 40*, 4 (Aug. 2021), 108:1–108:14.

[56] Walter, B., Marschner, S. R., Li, H., and Torrance, K. E. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (Goslar, DEU, 2007), EGSR'07, Eurographics Association, p. 195206.

[57] Wikipedia. By henrik - own work, cc by-sa 4.0. `https://commons.wikimedia.org/w/index.php?curid=3869326`, 2023. [Online; accessed 18-February-2023].

[58] Wikipedia. User:Henrik — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=User%3AHenrik&oldid=1080297653`, 2023. [Online; accessed 21-February-2023].

[59] Yang, J., and Xiao, S. An inverse rendering approach for heterogeneous translucent materials. In *Proceedings of the 15th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry - Volume 1* (New York, NY, USA, 2016), VRCAI '16, Association for Computing Machinery, p. 7988.

[60] Zeltner, T., Speierer, S., Georgiev, I., and Jakob, W. Monte carlo estimators for differential light transport. *ACM Trans. Graph. 40*, 4 (jul 2021).

[61] Zhang, C., Miller, B., Yan, K., Gkioulekas, I., and Zhao, S. Path-space differentiable rendering. *ACM Trans. Graph. 39*, 4 (2020), 143:1–143:19.

[62] Zhao, S., Jakob, W., and Li, T.-M. Physics-based differentiable rendering: From theory to implementation. In *ACM SIGGRAPH 2020 Courses* (New York, NY, USA, 2020), SIGGRAPH '20, Association for Computing Machinery.

A Naive workflow for geometry reconstruction

In this section, we provide a *naive* approach for geometry reconstruction for simple objects. We utilized a similar approach for the results shown (second and fourth row) in Figure 12. The procedure may be as follows: (1) Construct a simple Mitsuba scene with a light source and a sphere (or any other simple object) with multiple vertex positions. Note that the number of vertex positions might play a critical role in this procedure. (2) Import the scene from the previous step. (3) Load a (set of) reference images. (4) Select the vertex positions of the object for optimization. (5) Pick appropriate optimization parameters (e.g., min., and max. clamp values) and hyperparameters. For example, using Metashape, one could utilize the reconstructed objects' min/max bounding box values as min/max clamp values. (6) Start the optimization procedure. (7) Export results (i.e., vertex positions and consequently reconstructed mesh). (8) Use the reconstructed mesh from the previous step to further optimize the material properties of the object. As in the second and fourth row of Fig. 12, one can optimize vertex positions and materials of an object simultaneously.

Note that mesh reconstruction is not the primary target of our tool. Since geometry reconstruction might require additional constraints that are not implemented in our tool, the above-described procedure may provide unsuccessful results.

B A bottom-up analysis for synthetic data

In this section, we provide an extensive analysis of our results from synthetic data. We made a deliberate choice to keep our thorough analysis in the Appendix in order to illustrate the thought process that led to our conclusions on the reconstruction of translucent materials. To maintain the enthusiasm that we experienced during the evaluation phase, please note that we use a different *tone* to effectively communicate our results and engage the reader.

We will first start with an example where the Principled BSDF [14] is in use. In Figure 13, we show a reference image of a bunny object assigned to a Principled BSDF [14] and the corresponding initial image generated by manipulating reference parameter values. The reference and initial parameters after modification can be seen in Table 6.

parameters	reference	initial
α	0.01	0.5
c	$0.41, 0.82, 0.99$	$0.1, 0.1, 0.1$
σ_s	0.9	0.02
η	1.49	1.54

Table 6: Reference and initial parameters values from Figure 13.

We begin the optimization with default optimization parameters and hyper-

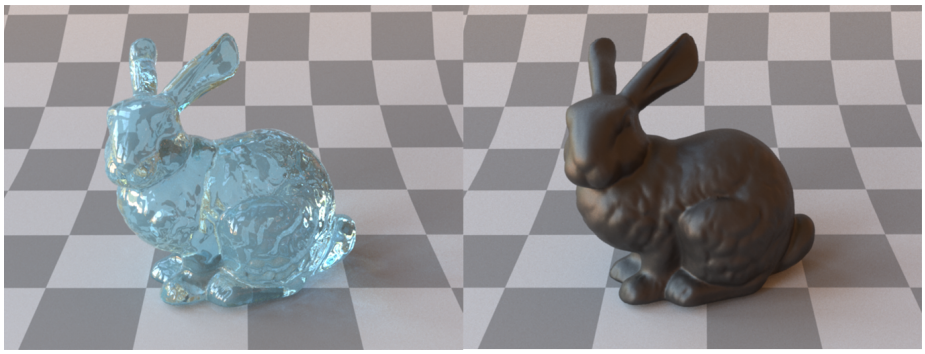

Figure 13: Left: Reference image. Right: Initial image.

parameters. The result of the optimization is shown in Figure 14.

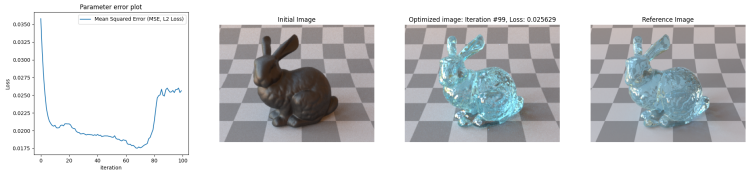

Figure 14: Results from the first optimization. See also Table 7.

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.0010	Default	Default
c	$0.41, 0.82, 0.99$	$0.2501, 0.8068, 0.9990$		
σ_s	0.9	0.9990		
η	1.49	2.0140		

Table 7: Results from the first optimization. See also Figure 14.

Our first attempt is clearly not good enough (Figure 14). The index of refraction (η) immediately pops up since it is not even near the reference result (Table 7). Other optimized parameters look better; however, those can also improve. η seems to be the main suspect why other parameters are not near enough to the corresponding reference value. Mainly because η seems to move in the wrong direction—we may put additional constraints for the η parameter, such that it moves in the correct direction (i.e., it should decrease from the initial value of 1.54 and near the reference value of 1.49). One possible approach for this would be using a maximum clamp value for the η parameter, which ensures the η parameter stays lower than the defined maximum clamp value during optimization. The provided default maximum clamp value for η is apparently not restrictive enough. In our second attempt, we indeed try out this approach and restart the optimization. The results are shown in Figure 15 and Table 8.

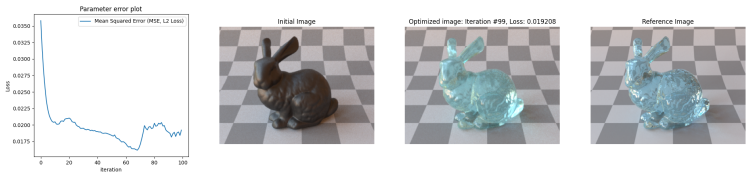

Figure 15: Results from the second optimization. See also Table 8.

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.2178	Default	Default
c	$0.41, 0.82, 0.99$	$0.4482, 0.9465, 0.9990$		
σ_s	0.9	0.9477		
η	1.49	1.4532		max=1.55

Table 8: Results from the second optimization. See also Figure 15.

This time the results seem better (Figure 15 and Table 8). If we look at η, we see that this time it moved in the *right direction*. However, this time α parameter seems to be quite off. Other parameters, such as base color and specular transmission, are not near enough to the reference as well. Visually, we can also clearly see the difference between the reference and optimized image. This time the α parameter seem the be the suspect. However, compared to the previous attempt, the α parameter seems to move in the *right direction* (i.e., actually nearing the reference value). Thus using the same approach as in the last case does not seem to make a lot of sense. One approach [6] to handle this case would be increasing the iteration count since α seems to move in the right direction, but most likely did not have enough *time* (i.e., iteration count) to near the target. Consequently, this time we only change the iteration count—we double it to 200; the default was 100—and test our approach once again. The results are shown in Figure 16 and Table 9.

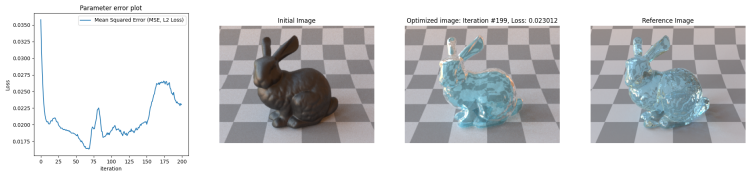

Figure 16: Results from the third optimization. See also Table 9.

Yikes! This time, α looks better; however, η seems to be way off from the target (Figure 16 and Table 9). In our first approach, η moved away from the target (the value increased), but this time it passed by the reference value (1.49) and moved even lower to the value of 0.85. The good news is the other

[6] Another option would be increasing the learning rate of α from 0.03 to possibly 0.06 (or more).

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.0010		
c	$0.41, 0.82, 0.99$	$0.4121, 0.8714, 0.9990$	ic=200	Default
σ_s	0.9	0.9138		
η	1.49	0.8558		max=1.55

Table 9: Results from the third optimization. See also Figure 16.

parameters look *better*, and fortunately, we have other tools at our disposal to handle the η issue. One might argue since the η value moved past the reference value, the learning rate might have played a role in it. The learning rate for η seems to be too fast for optimization. Next, we modify the learning rate of η from 0.03 to 0.003 and hope the results will look better. The results are shown in Figure 17 and Table 10.

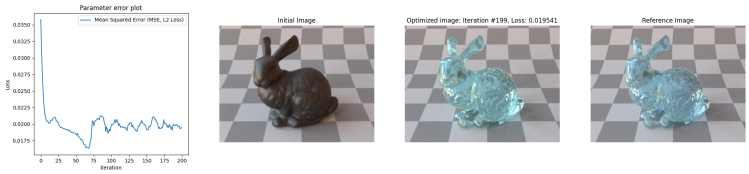

Figure 17: Results from the fourth optimization. See also Table 10.

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.0010		
c	$0.41, 0.82, 0.99$	$0.4131, 0.9194, 0.9990$	ic=200	Default
σ_s	0.9	0.9541		
η	1.49	1.5393		max=1.55, lr=0.003

Table 10: Results from the fourth optimization. See also Figure 17.

Overall, the result looks *better* (Figure 17 and Table 10). This time η is not way off the result, but it seems it did not moved too much from its initial value (1.54). This time we may have lowered the learning rate too much. We increase the learning rate of η to 0.008. Another tool disposal at our hand is using different samples per pixel (*spp*) value during optimization. With the argument that the noise introduced by the Monte Carlo (MC) sampling may cause unsatisfactory results, we increase the *spp* value from 4 to 8 [7]. The corresponding results are shown in Figure 18 and Table 11.

This time, at the end of the 199 iterations, the result does not particularly look *good* (Figure 18 and Table 11). Specifically, the η parameter—again—seems to be quite low. However, one thing that we can immediately note is the *smoothness* of the parameter error plot. In our previous attempts, the parameter error

[7] Ironically enough, the *optimization* tool that we have developed also requires an *optimization* on the author's end.

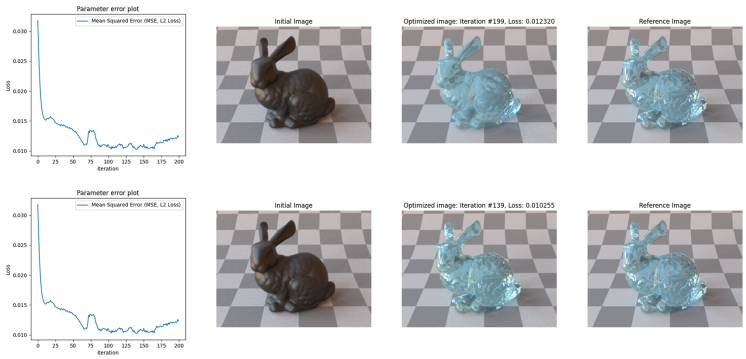

Figure 18: Results from the fifth optimization. Results from the last iteration (top) and the minimum loss (bottom). See also Table 11.

parameters	reference	optimized (last)	optimized (min. error)	hyperparam.	opt. param.
α	0.01	0.0052	0.0302		
c	$0.41, 0.82, 0.99$	$0.4471, 0.8523, 0.9990$	$0.4131, 0.8624, 0.9990$	ic=200, spp=8	Default
σ_s	0.9	0.8411	0.9245		
η	1.49	1.2197	1.4848		max=1.55, lr=0.008

Table 11: Results from the fifth optimization. See also Figure 18.

plot looked almost always quite *wobbly*. We can roughly see in the parameter error plot that before 175 iterations, the loss seems to be lower than after 175 iterations (i.e., latest iteration 199). This is actually good news since our tool provides a functionality that allows the user to switch between the minimum loss (best result) and the last iteration of the optimized scene state. Consequently, this functionality comes to our rescue, and we switch to the scene state with the minimum error rate, and *voila*—visually, the optimized image is indeed looking much better. We output the optimized parameters at iteration number 139 and observe very promising estimated parameter values (Figure 18 and Table 11).

To understand whether this result was achieved by modifying the *spp* value (thus the noise introduced by MC sampling with the default *spp* value 4) or learning rate, we take the same configuration as in the previous example and, this time, change only the learning rate from 0.003 to 0.008 (i.e., we don't touch the *spp* value). The results are shown in Figure 19 and Table 12.

parameters	reference	optimized (last)	optimized (min. error)	hyperparam.	opt. param.
α	0.01	0.0010	0.0145		
c	$0.41, 0.82, 0.99$	$0.4360, 0.9318, 0.9990$	$0.3243, 0.5904, 0.6991$	ic=200	Default
σ_s	0.9	0.9464	0.8784		
η	1.49	1.4248	1.5387		max=1.55, lr=0.008

Table 12: Results from the sixth optimization. See also Figure 19.

The visual results are not particularly *bad*; however, we immediately notice the parameter error plot is quite unstable (Figure 19 and Table 12). Even switching to the minimum loss state does not help us. The values are not way

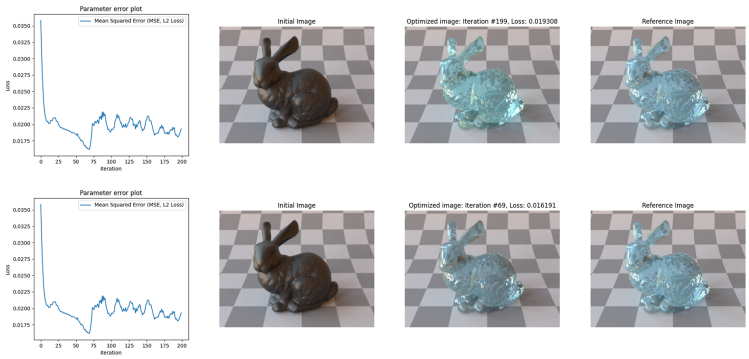

Figure 19: Results from the sixth optimization. Results from the last iteration (top) and the minimum loss (bottom). See also Table 12.

off, but definitely not quite there yet, especially as in our previous successful attempt (Figure 18). Consequently, we conclude that our assumption regarding the noise introduced by MC sampling is indeed *justifiable*, and it does indeed affect the optimization procedure. Subsequently, we get ahead of ourselves and try the same configuration as in our successful attempt, however this time with *spp* value 16 (i.e., $spp = 16$, η: learning rate = 0.008, η max. clamp value = 1.55, iteration count = 200). The results are shown in Figure 20 and Table 13.

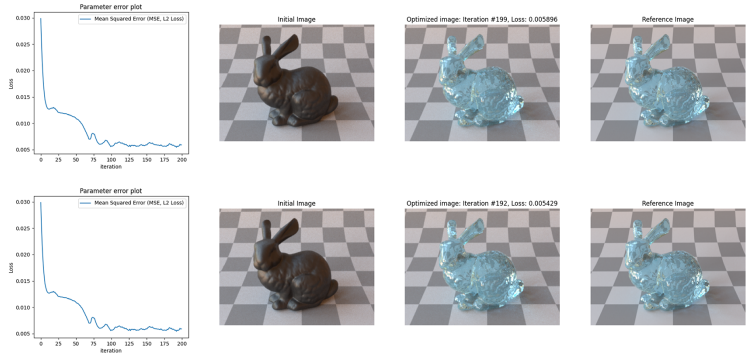

Figure 20: Results from the seventh optimization. Results from the last iteration (top) and the minimum loss (bottom). See also Figure 20.

parameters	reference	optimized (last)	optimized (min. error)	hyperparam.	opt. param.
α	0.01	0.0234	0.02132		
c	$0.41, 0.82, 0.99$	$0.4101, 0.8309, 0.9990$	$0.4054, 0.8320, 0.9990$	ic=200, spp=16	Default
σ_s	0.9	0.90987	0.9084		
η	1.49	1.4766	1.4899		max=1.55, lr=0.008

Table 13: Results from the seventh optimization. See also Figure 20.

This time, the results seem even much *better* than our previous successful attempt! It seems like this time, we don't even have to switch to the best result. The parameter error plot is quite smooth, and the last iteration visually seems acceptable enough. However, for the sake of completeness, we switch to the scene state with the minimum error. Visually there are not many noticeable changes. The minimum error seems to be achieved in iteration 192 and resulting parameter values also seem quite near to the scene state in the last iteration. Once again, we further conclude that the noise introduced by MC sampling plays a role during the optimization. The higher *spp* values provided us with better results, but one must note that with higher *spp* values, the optimization procedure takes longer.

Now, we test the same translucent material estimation problem as in Figure 14 using the dual buffer method by Deng et al. [9]. As in Figure 14—other than changing the optimization function to the dual buffer method—we begin with overall default optimization parameters and hyperparameters. The results are shown in Figure 21 and Table 14.

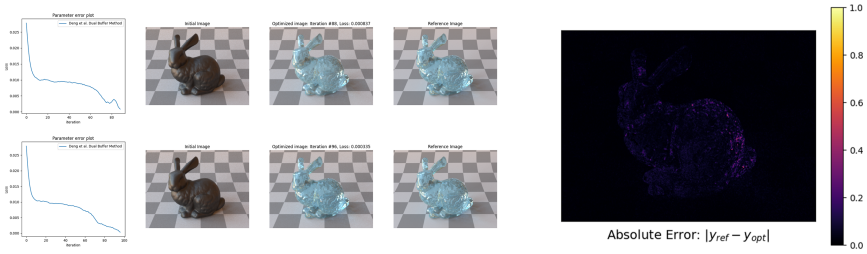

Figure 21: Results from the eighth optimization. Results with *spp*=4 (top), and *spp*=8 (bottom). Absolute error (right column) between the reference and optimization result (*spp*=8). See also Table 14.

parameters	reference	optimized (*spp*=4)	optimized (*spp*=8)	hyperparam.	opt. param.
α	0.01	0.0010	0.02132	DBM	Default
c	$0.41, 0.82, 0.99$	$0.4294, 0.7980, 0.9483$	$0.3877, 0.7996, 0.9795$		
σ_s	0.9	0.8364	0.9205		
η	1.49	1.5110	1.4780		

Table 14: Results from the eighth optimization. See also Figure 21.

In a fascinating turn of events, such as Ray Allen's amazing game-tying 3-pointer in Game 6 (NBA Finals 2013), or when Carl Sagan instructed the NASA team of Voyager 1 for a photograph of planet Earth from a record distance of about 6 billion kilometers, as part of that day's family portrait series of images of the solar system where the Earth's apparent size is less than a pixel and the planet appears as a tiny dot against the vastness of space among bands of sunlight reflected by the camera, the dual buffer method steals the show by reconstructing the bunny-object's translucent material properties in only 88 iterations (top row in Figure 21). Although we set the iteration count to 100, the iteration stops

at iteration 88 because the default minimum error value for the loss defined by our tool is 0.001, and since the loss rate at iteration 88 is 0.000837, the iteration stops. The resulting image looks quite similar to the reference image, and translucent parameter values are also quite near to the target. But since we are quite ambitious—and naive—we give the same configuration another shot, where we set the minimum error value for the loss from 0.001 to 0.0005 and increase the spp to 8. The results are shown in the bottom row in Figure 21 and Table 14.

Similar to our previous attempt, the dual buffer method introduced by Deng et al. [59] produces either visually or numerically acceptable results and accomplishes this in only 96 iterations (Figure 21 and Table 14). Once again, the optimization procedure stops earlier than the defined iteration count of 100 since the loss at step 96 (0.000335) is lower than the modified minimum error value (0.0005).

Consequently, after all the results we gathered from Figure [14-21], for our tool, we *highly* recommend the use of the optimization function *dual buffer method* by Deng et al. [59] for the task of translucent material reconstruction. We also *deduce* that the default optimization parameters and hyperparameters that our tool uses are highly compatible with the dual buffer method configuration. Therefore, the translucent material reconstruction results that we are going to present after this point almost always begin with the default parameters and the dual buffer method.

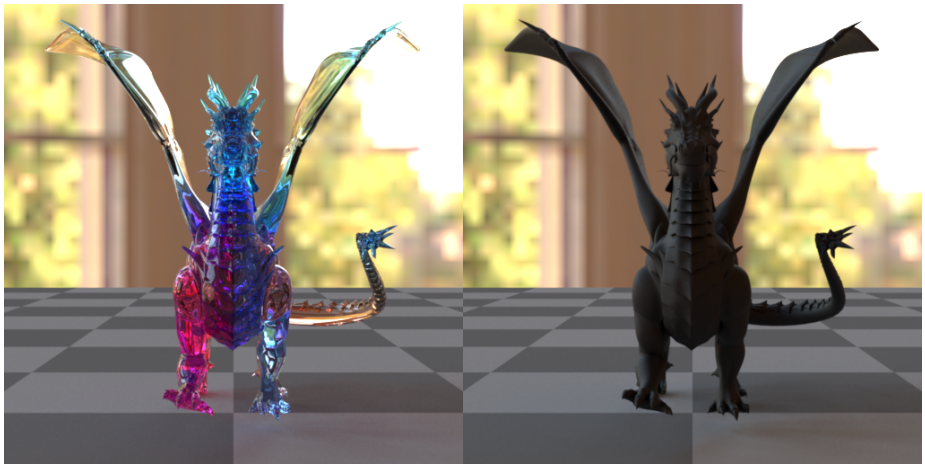

Figure 22: Left: Reference image. Right: Initial image. Dragon model by Delatronic [8], CC-BY license.

Next, we show another example where the Principled BSDF [14] is in use. However, this time we increase the complexity of the geometry and change the representation of the base color to a bitmap texture with varying colors. In Figure 22, we show a reference image of a dragon object assigned to a Principled BSDF [14] and the corresponding initial image generated by manipulating

parameters	reference	initial
α	0.001	0.7
c	bitmap	bitmap
σ_s	0.9	0.1
η	1.49	1.64

Table 15: Reference and initial parameters values from Figure 22.

reference parameter values. The reference and initial parameters after modification can be seen in Table 15. Since we obtained fine results in our previous example from Figure [14-21], we begin the optimization with the dual buffer method by Deng et al. [59] and samples per pixel (*spp*) value of 8. The result of the optimization is shown in Figure 23 and Table 16.

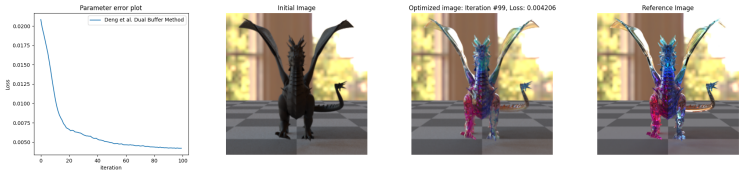

Figure 23: Results from the first optimization. See also Table 16.

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.1264	DBM, *spp*=8	Default
c	bitmap	bitmap		
σ_s	0.9	0.0010		
η	1.49	1.8079		

Table 16: Results from the first optimization. see also Figure 23.

Visually, our initial attempt looks quite poor (Figure 23). Moreover, when we take a glance at the optimized parameters, we observe that the values are not near enough to the reference values (Table 16). This is specifically the case for the σ_s parameter, which seems like it moved completely in the wrong direction. Another interesting parameter is the base color. As mentioned previously, this time, we are using not an RGB color but rather a bitmap texture. Later, we will discuss the bitmap texture that represents the base color parameter in more detail, but first, we will focus on the task at our hand. We also observe that the η parameter moved in the wrong direction. In the previous example, where we estimated the Principled BSDF parameters of the bunny, we had a similar issue, and we took an approach where we clamped the maximum value of η. This time, we follow a similar approach for the specular transmission (σ_s) parameter, where

we change the minimum clamp value from 0.001 to 0.1. For the moment, we don't change the maximum clamp value of η and hope that the right movement of the σ_s parameter will influence the movement of η during optimization. The results are shown in Figure 24 and Table 17.

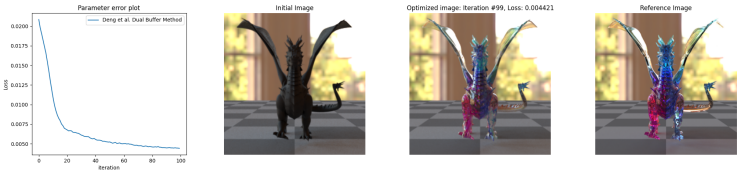

Figure 24: Results from the second optimization. See also Table 17.

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.1299	DBM, *spp*=8	Default
c	bitmap	bitmap		
σ_s	0.9	0.1000		min=0.1
η	1.49	2.0180		Default

Table 17: Results from the second optimization. See also Figure 24.

Unfortunately, we observe no improvements in our results either visually or numerically (Figure 24 and Table 17). The σ_s parameter seems to stuck in its minimum clamp value throughout the optimization. Moreover, the η parameter seems to be worsened and move even farther from our previous attempt. One key thing that we learned from our previous examples was the noise introduced by MC sampling affects the optimization procedure. In this example, the geometry of the dragon is more complex than the bunny. Therefore, we presume that the noise introduced by MC sampling is higher during rendering since the scattering of light from the object is more complex as well. With this assumption, we test our hypothesis and increase the *spp* value from 8 to 16. Additionally, we increase the iteration count from 100 to 200 and bump the minimum clamp value of the σ_s parameter from 0.1 to 0.11, with the argument that the optimization may require more steps and a gentle push to move in the right direction. The results are shown in Figure 25 and Table 18.

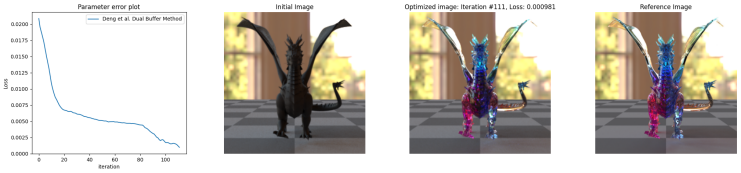

Figure 25: Results from the third optimization. See also Table 18.

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.0084	DBM, *spp*=16, ic=200	Default
c	bitmap	bitmap		
σ_s	0.9	0.9542		min=0.11
η	1.49	1.4758		Default

Table 18: Results from the third optimization. See also Figure 25.

Fortunately, this time both visual and numerical results look *good* (Figure 25 and Table 18). The optimization ended at iteration 111 because the loss at iteration 111 (0.000981) was lower than the default minimum loss (0.001). Both η and σ_s parameters seemed to be quite near to the reference values. To see whether we can get even closer to the reference values, we lower the minimum default error from 0.001 to 0.0001 and start the optimization again with the same configuration. The results are shown in Figure 26 and Table 19.

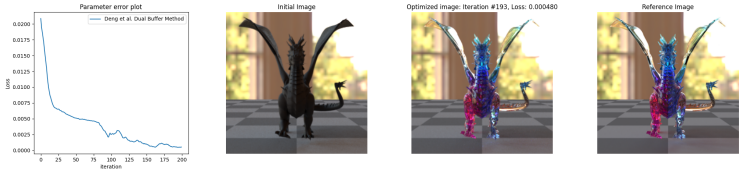

Figure 26: Results from the fourth optimization. See also Table 19.

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.0119	DBM, *spp*=16, ic=200, min. err=0.0001	Default
c	bitmap	bitmap		
σ_s	0.9	0.9770		min=0.11
η	1.49	1.5458		Default

Table 19: Results from the fourth optimization. See also Figure 26.

This time the optimization finishes in iteration 200 (Figure 26 and Table 19). At the end of iteration 200, the overall error (loss) rate is 0.000523. When we switch to the optimization state with the minimum loss throughout the whole procedure, we observe minimum loss at iteration 193 is 0.000480. For conciseness, we discuss only the optimization state at iteration 193 since numerical results are quite identical to the optimization state at iteration 200. Interestingly, although the overall loss rate is lower than our previous attempts (i.e., the previous overall loss rate was 0.000981), and although visually both optimized images at iterations 111 and 193 look quite identical, we observe that the η parameter is actually farther from the target. In conclusion, we remind ourselves that a lower error rate not necessarily refers to better material estimation.

Previously, we mentioned that we will discuss the bitmap texture that represents the base color parameter in more detail. When we look at the outputted

optimized bitmap texture, we observe that the resulting bitmap texture looks far away from the reference—both shown in Figure 27. However, the look of the optimized bitmap texture is expected since the optimization begins from a completely empty (black) bitmap texture, and only those pixels in the bitmap texture are optimized where the mapping of texture takes place (i.e., the object's UV-mapping). Inevitably, the optimized bitmap texture takes the shape of the object, more specifically, the UV map specified for that object. Nevertheless, the good thing is the look of the final rendered image, which we have shown in Figure 25.

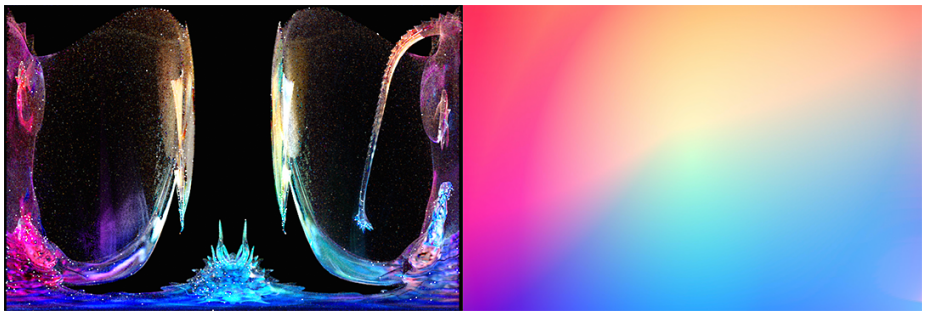

Figure 27: Left: Optimized bitmap texture from Figure 25. Right: Reference.

However, one limitation of this approach is that the optimized bitmap texture only works for those camera poses (i.e., only loaded reference images) that our scene description contains. The idea is much easier to grasp visually. In Figure 28, we show another render of the object with the optimized values; however, this time, we rotate our camera to observe the side of the dragon. Compared to its reference counterpart, we observe that the optimized dragon object's color is not near enough to the reference. This is especially the case on the wings of the dragon, which can also be seen in Figure 27—left and right mid-part of the optimized texture; the wings are still mostly black.

Consequently, one could make the argument, if we use multiple camera poses, we would also be able to optimize the other part of the object's bitmap texture. Fortunately, since our tool provides this functionality, we can test this assumption. In Figure 30, we show the results where we use multiple reference images for optimization. In total, we use 4 sensors; where we place the camera front, back, left, and right sides of the dragon. The results are shown in Figure 30, 29 and Table 20. Fortunately, our assumption was indeed correct; with multiple camera poses, we are able to optimize the bitmap texture of the object more exactly, and this can also be seen in the outputted bitmap texture in Figure 31.

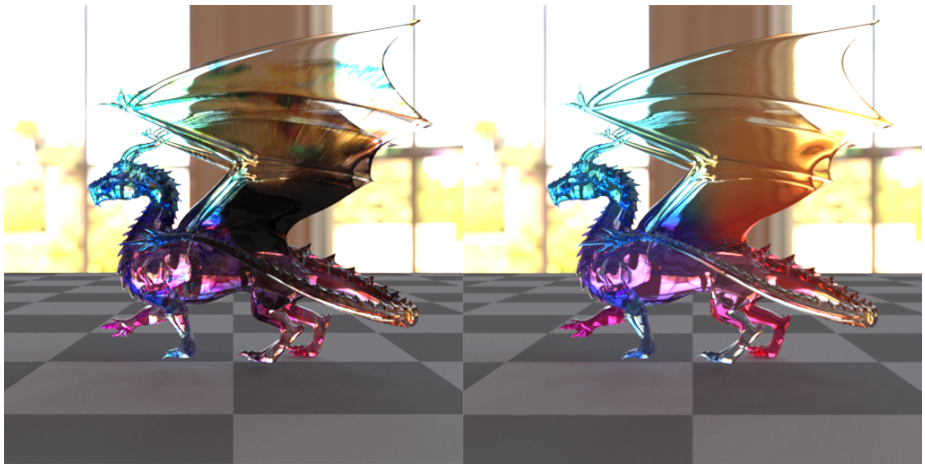

Figure 28: Left: Optimized Dragon model from Figure 25, rendered from its side. Right: Corresponding reference image.

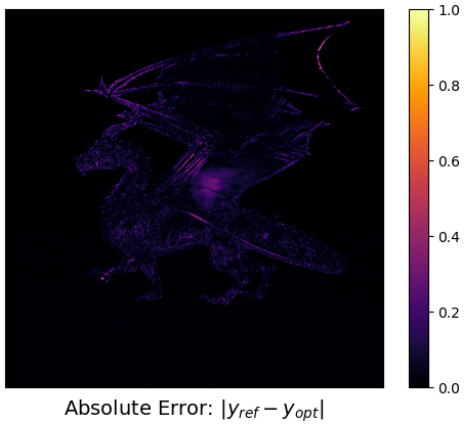

Figure 29: Absolute error between the reference and optimized image from its side. See also Figure 30.

parameters	reference	optimized	hyperparam.	opt. param.
α	0.01	0.0019	DBM, *spp*=16, ic=200, min. err=0.0001	Default
c	bitmap	bitmap		
σ_s	0.9	0.9728		min=0.11
η	1.49	1.5022		Default

Table 20: Results from the fifth optimization. See also Figure 30.

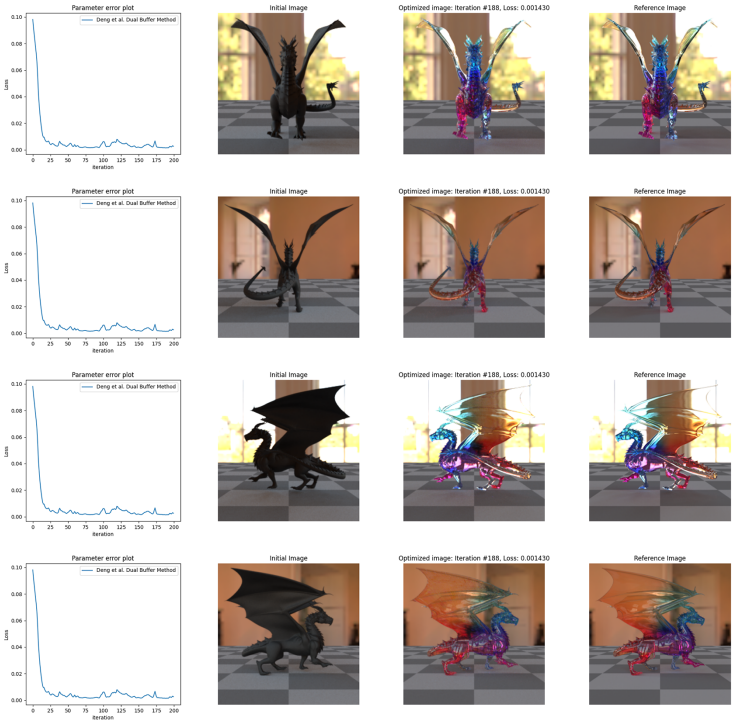

Figure 30: Results from the fifth optimization (multiple cameras). See also Table 20.

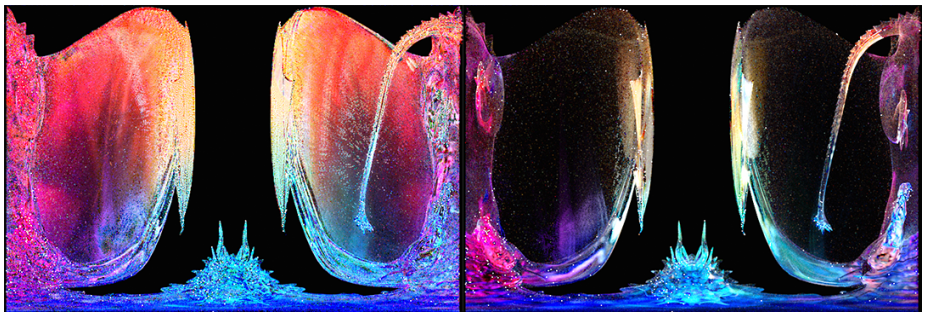

Figure 31: Left: Optimized bitmap texture from Figure 30. Right: Optimized bitmap texture from Figure 25

We now switch to our results, where we use our second approach: namely *Roughdielectric BSDF* [56] with a *homogeneous participating medium* [33]. Consequently, these examples use a differentiable volumetric path tracer (*prbvolpath*) [33] and also reconstruct volumetric material properties such as the extinction coefficient (σ_t) and single-scattering albedo (c) of the medium. We kindly remind the reader that our tool selects the appropriate integrator depending on the scene parameters that the loaded scene file contains. Therefore, if the loaded scene file includes a homogeneous participating medium, our tool selects mitsuba's (*prbvolpath*) integrator [33].

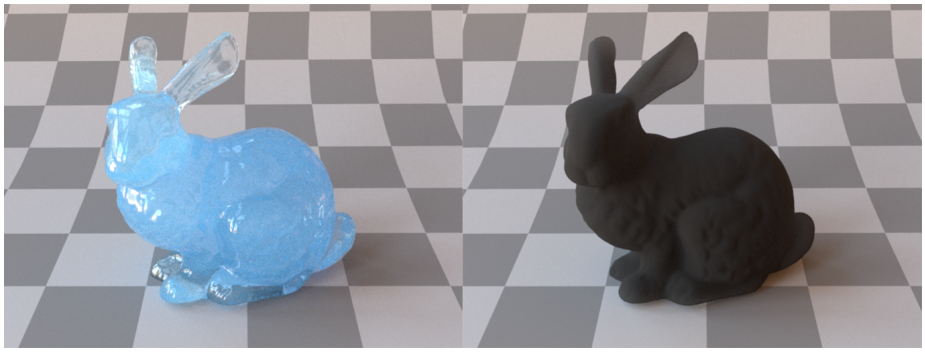

Figure 32: Left: Reference image. Right: Initial image.

parameters	reference	initial
α	0.01	0.5
η	1.49	1.54
c	$0.41, 0.82, 0.99$	$0.01, 0.01, 0.01$
σ_t	0.4	0.98

Table 21: Reference and initial parameters values from Figure 32.

In Figure 32, we show a reference image of a bunny object assigned to a

Roughdielectric BSDF [56] with a *homogeneous participating medium* [33] and the corresponding initial image generated by manipulating reference parameter values. The reference and initial parameters after modification can be seen in Table 21.

Similar to our first example in Figure 14, a bunny object is shown in Figure 33. Since we obtained quite successful results in our previous attemps, we use the dual buffer method by Deng et al. [59] and samples per pixel (*spp*) value of 8. The results from our first attempt are shown in Figure 33 and Table 22.

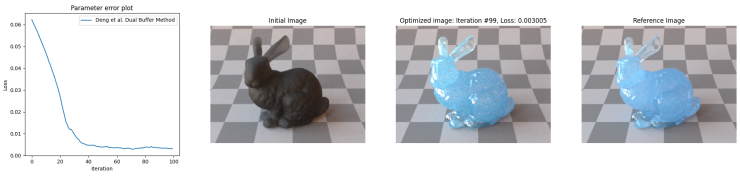

Figure 33: Results from the first optimization. See also Table 22.

parameters	reference	optimized	hyperparam.	opt. param
α	0.01	0.0010	DBM, *spp*=8	Default
η	1.49	1.9670		
c	$0.41, 0.82, 0.99$	$0.59021, 0.9281, 0.9990$		
σ_t	0.4	0.6037		

Table 22: Results from the first optimization. See also Figure 33.

Although visually, our results look fine enough, surprisingly, our initial attempt provides us with numerically incorrect results (Figure 33 and Table 22). This is especially the case for the η parameter. Once again, we observe that the η parameter is moved in the wrong direction. Since we had a similar issue previously, we change our configuration where we increase the *spp* value from 8 to 16 and iteration count from 100 to 200 and hope that the η parameter will converge in the right direction and consequently affect other parameters as well. The results from our second attempt are shown in Figure 34 and Table 23.

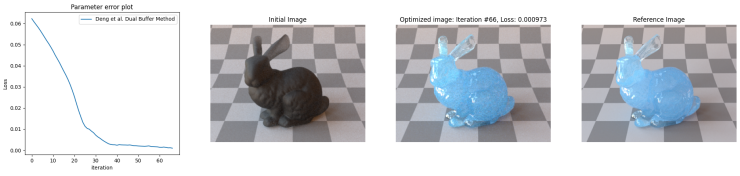

Figure 34: Results from the second optimization. See also Table 23.

In our second attempt, the optimization ends at iteration 66 since the overall loss (0.000973) was lower than the default minimum error rate (0.001). Visually the results look good. Numerically almost all optimized values look fine except

parameters	reference	optimized	hyperparam.	opt. param
α	0.01	0.0035	DBM, *spp*=16, ic=200	Default
η	1.49	1.6247		
c	$0.41, 0.82, 0.99$	$0.4091, 0.8614, 0.9990$		
σ_t	0.4	0.4762		

Table 23: Results from the second optimization. See also Figure 34.

the η parameter. Interestingly, although the overall loss in our last iteration was lower than the defined minimum error, the η parameter again seemed to move in the wrong direction. Since this configuration provided us with almost successful results, we extend our configuration where we set the maximum clamp value of η to 1.55, decrease the minimum error rate from 0.001 to 0.0001, and lower the iteration count from 200 back to 100, and hope more exact numerical results. The results are shown in Figure 35 and Table 24.

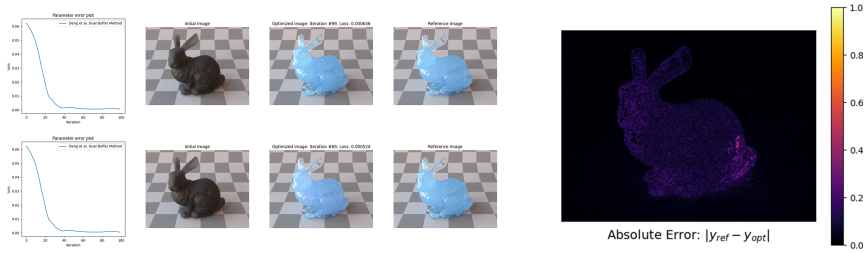

Figure 35: Results from the third optimization. Results from the last iteration (top) and with the minimum loss (bottom). Absolute error (right column) between the reference and optimization result (top). See also Table 24.

parameters	reference	optimized (last)	optimized (min. error)	hyperparam.	opt. param
α	0.01	0.0152	0.0010	DBM, *spp*=16, min. error=0.0001	Default
η	1.49	1.4806	1.4739		max=1.55
c	$0.41, 0.82, 0.99$	$0.4732, 0.8473, 0.9990$	$0.5004, 0.8224, 0.9990$		Default
σ_t	0.4	0.50240	0.4655		

Table 24: Results from the third optimization. See also Figure 35.

In our third attempt, the optimization ran fully and ended with an overall loss rate of 0.000636. This time not only the visual results look fine but also the numerical results (Figure 35 and Table 24). We observe that the η parameter got quite close to its corresponding reference value. The extinction coefficient (σ_t) parameter seems to be farther than our previous attempt, but since the visual results are fine enough, we take it as an acceptable result. However, with the tools at our disposal, one could further tune the optimization parameters and hyperparameters to acquire *possibly* even more exact results. When we switch to the scene state where the minimum loss was achieved throughout the optimization, we observe similarly acceptable numerical results. The minimum loss rate seems (0.000524) to be achieved at iteration 69. We observe that the

σ_t parameter is closer to its corresponding reference value. However, the η and c parameters are just a bit farther compared to the last optimization.

Next, we show another example where the *Roughdielectric BSDF* [56] with *homogeneous participating medium* [33] is in use. Similar to the Principled BSDF case, this time, we increase the complexity of the geometry using the Dragon [8] model. However, compared to the Principled BSDF case that we showed in Figure 22, rather than using a bitmap texture to represent the color of the object, we use a grid-based volume data structure [33] to represent the object's varying color value. The reference and initial images can be found in Figure 36, and corresponding parameter values are shown in Table 25.

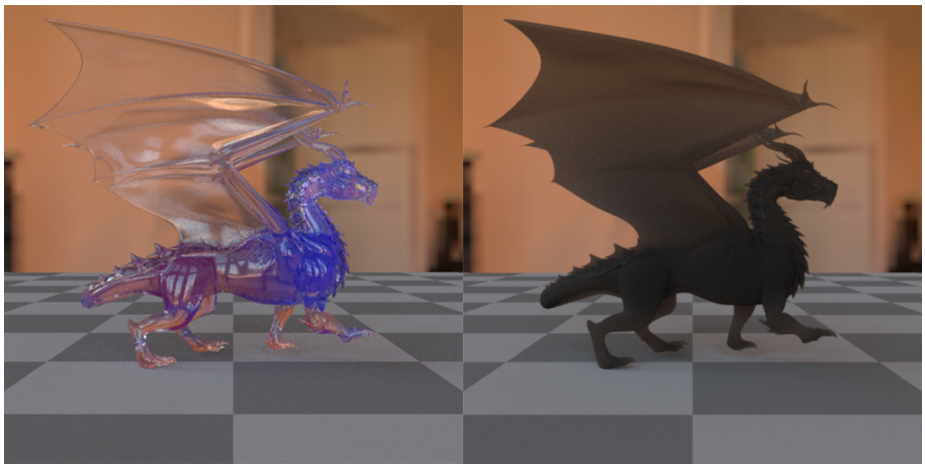

Figure 36: Left: Reference image. Right: Initial image.

parameters	reference	initial
α	0.01	0.98
η	1.49	1.54
c	Grid-Based Volume	Empty Grid-Based Volume
σ_t	0.4	0.98

Table 25: Reference and initial parameters values from Figure 36.

The experience we gained where we used a bitmap texture to represent the object's color should provide us with valuable insights. As in the bitmap texture case, optimizing the grid-based volume data structure with only one camera (i.e., one reference image) would not help us to reconstruct the object's *overall* varying color. Consequently, we begin the optimization with multiple sensors—where we place each camera at the front, back, left, and right side of the dragon object.

Moreover, we also make use of the experience that we gained from the previous example from Figure 33 to Figure 35 and start the optimization with a *similar* configuration. More concretely, we use the dual buffer method by Deng et al. [59] as the optimization function, set the *spp* value to 16, and leave other

parameters as default. The results are shown in Figure 37 and Table 26. Please note that in our initial attempts for the sake of conciseness, we only show the optimization results from only the side of the Dragon [8] model.

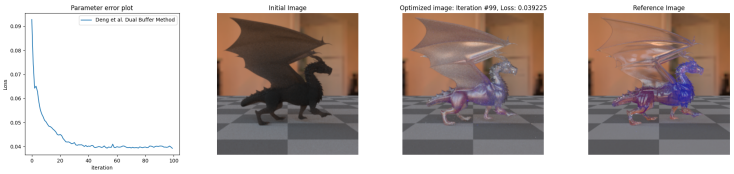

Figure 37: Results from the first optimization. See also Table 26.

parameters	reference	optimized	hyperparam.	opt. param
α	0.01	0.1229	DBM, spp=16	Default
η	1.49	2.2165		
c	grid-based volume	grid-based volume		
σ_t	0.4	0.2645		

Table 26: Results from the first optimization. See also Figure 37.

In our first attempt, both visual and numerical results are quite poor. The first parameter that pops to the eye is again the η parameter. Since previously the η parameter also caused similar issues, we know roughly how our second approach should look. We expect that if the η parameter moves in the right direction the other parameters will also improve. Consequently, in our second attempt, we test our hypothesis by setting the maximum clamp value of η to 1.55 and leave other parameters as in our first attempt. The results are shown in Figure 38 and Table 27.

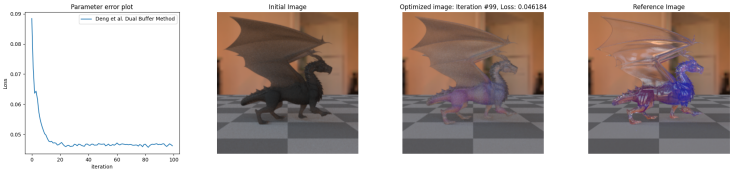

Figure 38: Results from the second optimization. See also Table 27.

parameters	reference	optimized	hyperparam.	opt. param
α	0.01	0.3063	DBM, spp=16	Default
η	1.49	1.5499		max=1.55
c	Grid-Based Volume	Optimized Grid-Based Volume		Default
σ_t	0.4	0.2695		

Table 27: Results from the second optimization. See also Figure 38.

Interestingly our second approach also results in quite poor results. We observe that the η parameter seems to be stuck at the value of 1.55, which is the maximum clamp value we defined for the η parameter. More interestingly, after 100 iterations, the α parameter seems also to be stuck at 0.3, which is also troublesome since it most likely affects the σ_t and η parameters' correct convergence. Visually, the incorrectness of the α parameter is quite clear; the surface of the dragon is still extremely rough.

In the meantime, we tried different configurations for better results. For example, we used higher *spp* values, larger iteration counts, and lower or higher learning rates. Interestingly, on many of our attempts, we observed that the α parameter remained around the range of values 0.2 to 0.3, which visually indicates a quite rough surface material. Unfortunately, our attempts were in vain, and we were quite unsuccessful with our approach. For the sake of brevity, we omit the details regarding our unsuccessful attempts; however, in the future, it would be interesting to analyze which pitfalls—theoretical and practical—affect our translucent material reconstruction procedure in more detail.

Next, we mainly focus on which approach helps for the task at our hands. During the development period and testing the optimization procedure, we observed on some occasions optimizing some scene parameters separately that produce discontinuities provided better results. Consequently, as a last resort, we separate the optimization into **two** parts: *first* we optimize the α, σ_t and c parameters defining the iteration count to 50. Next, with the optimized values from the *first part*, we restart the optimization by including the η parameter with another 50 iterations. The results are shown in Figure 39 and Table 28.

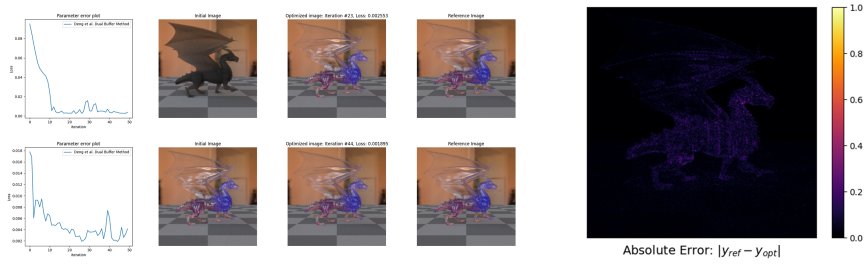

Figure 39: Results from the third optimization. Results from the first (top) and second (bottom) parts. Absolute error (right column) between the reference and optimization result (part-2). See also Table 28.

parameters	reference	optimized (part-1)	optimized (part-2)	hyperparam.	opt. param
α	0.01	0.0010	0.0081	DBM, spp=4	Default
η	1.49	1.54	1.4861		
c	grid-based volume	grid-based volume	grid-based volume		
σ_t	0.4	0.3593	0.3556		

Table 28: Results from the third optimization. See also Figure 39.

With this approach, we observe relatively acceptable visual and numeri-

cal results. The main drawback of this approach is dividing the optimization into two parts. Consequently, with this approach, there are two steps that the user needs to accomplish manually. *First*, the user should find the group of parameters that the optimization works well together and start the procedure and eventually export them. *Second*, the user should start the optimization by including the parameter that causes difficulty in the first place and use the exported optimized parameters from the first half of the procedure. More specifically, in case of difficulties during translucent material reconstruction—which are not solvable with the functionalities that our tool provides—our recommendation is including the η parameter **only** in the *second* part of the reconstruction procedure, as we showed in Figure 39.

In this section, we proposed two BSDF models for translucent material reconstruction using a gradient-based optimization procedure, and we showed the corresponding visual and numerical results for synthetic data. As we noticed in this section, translucent material estimation is complex, and the task often requires additional constraints such that the parameters converge in the right direction. Consequently, in this section, we not only wanted to show our results using the proposed BSDF models but also how to apply additional constraints for different scenarios using our tool.

One important thing to note is, in the case of synthetic data, applying additional constraints such as maximum clamp value for a specific parameter is rather straightforward since we already know the corresponding reference value. However, this task gets more complicated in the case of real-world data since we *often* do not know the corresponding reference values. Consequently, we relied—we also expect other users—more heavily on the resulting visuals whenever we included additional constraints for the optimization.