Efficient and Accurate Optimization in Inverse Rendering and Computer Graphics

Michael Fischer

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

February 6, 2025

I, Michael Fischer, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Michael Fischer

Abstract

Efficient and accurate representation of graphic assets, a long-standing task in the graphics community, has achieved new heights with the advent of learning-based methods by representing visual appearance as neural networks. Surprisingly, such visual appearance networks are often trained from scratch – an expensive operation that ignores potentially helpful information from previous training runs. This thesis therefore introduces Metappearance, an algorithm which optimizes over optimization itself and enables orders of magnitude faster training times at indistinguishable visual quality while retaining the network's adaptability to new, unseen data.

Moreover, even a fully converged network, albeit a smooth function, does not guarantee optimization success when employed in an inverse rendering scenario. In fact, it is common for inverse rendering to exhibit plateaus – regions of zero gradient – in the cost function, which hinder gradient-based optimization from converging. Chapter 4 therefore introduces an algorithm that smooths out such plateaus by convolving the rendering equation with a Gaussian blur kernel and thus successfully optimizes scenarios where other, rigid methods fail to converge.

Finally, while recent research has shown that specialized treatment of the renderer's internals can yield correct, usable gradients, there is no unified, systematic way of differentiating through *arbitrary*, black-box graphics pipelines. We therefore introduce the concept of neural surrogates, which allow differentiating through arbitrary forward models without requiring access to, or making any assumptions on, the rendering pipeline's internals. We show that our neural surrogate losses can successfully optimize various graphics tasks and scale well to high dimensions, a domain where traditional derivative-free optimizers often do not converge.

Acknowledgements

This thesis is the culmination of my PhD work at University College London, carried out from November 2020 to December 2024. I am deeply indebted to several individuals and their continuous and unwavering support throughout this time. Without you, the following thesis would not have been possible.

First and foremost, the people that require no introduction – but shall receive one regardless – are Tobias Ritschel and Niloy Mitra, my supervisors. It is my firm belief that a young PhD student could wish for no better supervisors, and their guidance and invested time is unparalleled. Especially from you, Tobias, I've learned a lot about research methodology and the corresponding ideation process, and hope to continue to do so in the future. This thesis is in no small part the result of our countless discussions, getting-to-the-bottom-of-things talks, coding- and debugging sessions and figure iterations — thanks a lot!

A big thank you goes out to my London colleagues, who helped create a positive and creative atmosphere in the lab: Nels and Ziwen, who were always in the office and who I've shared countless lunches with; Luca and Sanjeev, my cohort-peers, who were progressing along the same trajectory and who I've spent several deadlines with; Philipp, David and Preddy, who left shortly after I joined but whose friendship and advice has been invaluable along the way; and finally Niladri, Romy and Chen, the younger generation, whose eagerness inspires me to keep pushing myself towards new topics and being a better researcher.

Additionally, I am deeply grateful for the continuous support of Meta Reality Labs, who have sponsored the cost of my tuition and stipend (grant number 5034015) and taken me on as an intern during the summer of 2023 in Redmond, Washington.

Specifically, Carl Marshall for being a great mentor and project manager, Zhao Dong for great discussions and Zhengqin Li for his hands-on help.

Further, I would like to express my gratitude towards the Ezra family and the Rabin Ezra scholarship trust, who have decided to support my research and career with a bursary award in early 2024.

A special mention should also go out to my co-authors at Adobe Research; Thibault Groueix, Vova Kim, Iliyan Georgiev and Valentin Deschaintre, who have supported me during my internship in the summer of 2024 and provided valuable project guidance and insights while simultaneously being great friends altogether.

When speaking of great friends, the following people must not remain unmentioned: Barbora, Georg, Timo, David, Johannes, Martin, Jan and Sanjeev - thanks for all the fun activities, dinner parties and good times all over the world.

Finally, I would like to thank my parents and brothers for their support throughout this time, and for putting up with my deadlines and stress during Christmas breaks, constant travel, not being home much, and always talking about lofty things that no one (including me) really understands.

My final and biggest gratitude goes to Jenni, who has been extremely supportive and understanding through all the ups and downs of this journey. Be it from the early days of my first PhD year, when we shared a room and both worked from home due to Covid-19 restrictions, or later in London, when countless evenings and weekends were spent in the lab before deadlines.

I couldn't have done it without any of you, so please take a moment to pat yourself on the back and be sure to have my heartfelt gratitude. Now, enough with the sentiment, let's get into the research parts!

Impact Statement

This thesis presents three novel approaches for efficient network training and differentiable, inverse graphics. The contributions presented in this work were published[1] at SIGGRAPH ASIA 2022, CVPR 2023 and SIGGRAPH 2024, respectively, which reside among the premier venues for scholarly work in the fields of computer graphics and computer vision. All publications in this thesis have been peer-reviewed by three (CVPR) to five (ACM ToG) independent reviewers in a double-blind review process. To facilitate further academic work, we have publicly released code, data and trained models (where applicable) for the presented approaches.

Michael Fischer and Tobias Ritschel: "Metappearance: Meta-learning for visual appearance reproduction." ACM Transactions on Graphics (TOG) 41.6 (2022): 1-13, [64].

Michael Fischer and Tobias Ritschel: "Plateau-reduced differentiable path tracing." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2023), [66].

Michael Fischer and Tobias Ritschel: "ZeroGrads: Learning Local Surrogates for Non-Differentiable Graphics." ACM Transactions on Graphics (TOG) 43.4 (2024): 1-15, [67].

[1] Some of the figures and tables have been adapted for format to better fit the layout of this thesis. The content was not altered.

Although this thesis primarily focuses on the algorithms behind optimization in inverse rendering, my broader research at UCL has also included work on learned 3D appearance transfer and 3D material selection, where optimization-based approaches play a crucial role, resulting in the following publications:

Michael Fischer, Zhengqin Li, Thu Nguyen-Phuoc, Aljaž Božič, Zhao Dong, Tobias Ritschel, Carl Marshall: "NeRF Analogies: Example-Based Visual Attribute Transfer for NeRFs." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2024), [70].

Michael Fischer, Iliyan Georgiev, Thibault Groueix, Vladimir G. Kim, Tobias Ritschel, Valentin Deschaintre: "SAMa: Material-aware 3D Selection and Segmentation." arXiv preprint 2024, currently under submission, [69].

Contents

Chapter 1

Introduction

Since the beginning of mankind, light has been humanity's guide, illuminating the world's intricate beauty with astonishing detail and near-infinite speed. However, what seems effortless for our eyes in reality is a complex interplay of physics, biology, and learning. In computer graphics, we strive to replicate this interplay – capturing the way light bounces, scatters and propagates though a scene before finally being received by an observer – through algorithms that reconstruct reality at the highest fidelity. Yet, even though these algorithms continually grow more powerful, they remain blind to their own inefficiencies: ignoring the lessons of past optimizations, stalling on plateaus in optimization landscapes, and failing to handle the opaque inner workings of modern rendering pipelines.

Much like light itself, this thesis takes a journey – exploring pathways to illuminate these challenges and bend the rules of conventional optimization with three novel algorithms. We introduce Metappearance, a method that learns how to learn, allowing networks to adapt rapidly to new tasks by distilling knowledge from previous optimizations. We smooth optimization landscapes through Gaussian convolutions, traversing plateaus that halt traditional methods. And we replace blind trial-and-error with intelligent neural surrogates, empowering optimization of complex modern black-box graphics pipelines. Just as light transforms our perception of the world, these methods transform the process of optimizing visual representations, pushing optimization in computer graphics closer to the elegance and efficiency of human perception.

To ground these ideas, we turn to the technical underpinnings of photorealism in computer graphics. Achieving such realism traditionally relies on algorithms that approximate the rendering equation [140], a high-dimensional integral that describes how incoming light, or *radiance*, scatters through a scene and therefore ultimately shapes how that scene is perceived by an observer.

Yet, while the accurate modeling of light transport is a necessary condition for achieving photorealism in rendering, it is not a sufficient one – we further need high-fidelity graphics assets, such as materials and textures, scattering functions that describe reflection and refraction within the scene, and accurate illumination models to achieve real-world appearance reproduction. For an efficient usage, these assets must strike the balance between the highest possible fidelity and practical applicability in their respective use-cases. A good counter-example are scanned bi-directional reflectance distribution functions (BRDFs): while they permit highest-quality reflectance reproduction, their storage requires several gigabytes, making their use in real-world applications impractical [317]. With the recent advances in deep learning and neural rendering, assets are commonly encoded into deep neural networks, leading to *deep graphics assets* [193; 249; 293; 108]. Their benefits include compression and reduced storage [293; 250] at equal or higher quality [293; 249], smoothness for easy interpolation, and amenability to parallelization and real-time inference [203; 201]. Moreover, visual appearance networks can entirely be *fit from data* – an important distinction from previous work, which hand-designed sophisticated models and algorithms to closely replicate real-world appearance. Using neural networks allows designers and practitioners to loosely define a *space of algorithms* (via the network architecture), in which the most suitable algorithm is then selected via optimization, by fitting the network to the given real-world visual appearance data. It should be mentioned that the above benefits come at the cost of editability, as networks are much harder to interpret than hand-designed models.

Surprisingly, the research community trains most deep graphics assets "in isolation", i.e., without re-using experience from previous or parallel optimization iterations. Moreover, models are usually initialized from scratch [103] and hence

forego a potential inductive bias that the optimization could benefit from [63; 214; 296], as each newly initialized network will have to traverse regions in parameter space (e.g., close to the initialization) which likely have little in common with the optimal parameters, resulting in prolonged training times.

An alternative, more traditional way of parametrizing assets in graphics are *scene parameters*, such as the position or intensity of a light source, the diffuse albedo of a wall, or the roughness of a surface. The raison d'être of these parameters is to adjust the scene configuration or the employed (shading) models to achieve the desired level of photorealism, often based on reference observations from the real world. Their optimal configuration, however, is seldomly known, and it can take skilled artists hours of work to reproduce the appearance of even a single object or observation with high fidelity.

The field of inverse rendering therefore aims to *automatically* regress these scene parameters from data in the form of images or other measurements, leading to the fundamental question of how to do so efficiently and reliably. A straightforward way could employ a trial-and-error approach, where we simply try all parameter configurations and thus will eventually find the correct one. While this might work for simple problems, it quickly becomes intractable for even modest problem dimensionality. As an alternative, we could use gradient-based optimization, where, starting from an initial guess, we minimize an error measure between the real-world observations and the current parameters' renderings. However, this is difficult, as both common forms of rendering (rasterization and path tracing) are not easily differentiable due to step functions and discontinuities in the forward model [183; 179; 237; 38]. Even if we manage to make the rendering process differentiable, there still is no guarantee that our gradient-based optimization will actually converge to a useful parameter value – in other words, the mere existence of gradients does not guarantee a successful optimization outcome [192].

One of the underlying reasons is the existence of plateaus – constant regions of zero gradient – in the optimization landscape, which commonly arise under standard image metrics and impede gradient flow during the optimization. Moreover, making a

renderer differentiable requires sophisticated and specialized treatment of edge cases [169; 183] that often result in entirely re-writing the rendering engine [219; 131; 133; 12] – a time-consuming manual process that cannot be scaled to all graphics engines we might wish to differentiate through, and which is additionally bound by project requirements such as dedicated programming languages or hardware constraints.

This thesis therefore asks the following question: How can we achieve *fast* network training, while at the same time being able to use the resulting network in an inverse rendering scenario with a *smooth* cost landscape and under an *arbitrary* graphics pipeline?

1.1 Contributions

To make progress towards answering this question, this thesis presents three works that address each of the aforementioned subproblems individually. After giving an overview over relevant concepts and terms in Chapter 2, the first contribution of this thesis is Metappearance, presented in Chapter 3, which focuses on the rapid and efficient training of deep graphic assets that encode visual appearance. Subsequently, Chapter 4 addresses the problem of plateaus in the cost-landscape of optimization tasks and presents a novel, convolved version of the rendering equation that enables convergence on plateau-laden optimization scenarios. Lastly, Chapter 5 presents ZeroGrads, a novel algorithm for making arbitrary graphics pipelines differentiable by approximating the local cost landscape via an easily differentiable neural network.

1.1.1 Metappearance

In Chapter 3, we introduce Metappearance, a meta-learning approach to visual appearance [55] reproduction. The motivation for Metappearance originates in the fact that deep graphic assets are traditionally trained using on of three paradigms: For *overfit* networks, a NN is overfitted onto a single problem instance (e.g., a single BRDF), which leads to very high reproduction quality, but suffers from long training times and does not generalize to other problem instances [193; 293; 200; 335; 79]. Contrary, *general* networks are trained to replicate all problem instances of a dataset in a feed-forward manner and exhibit fast inference times even for unseen inputs, but

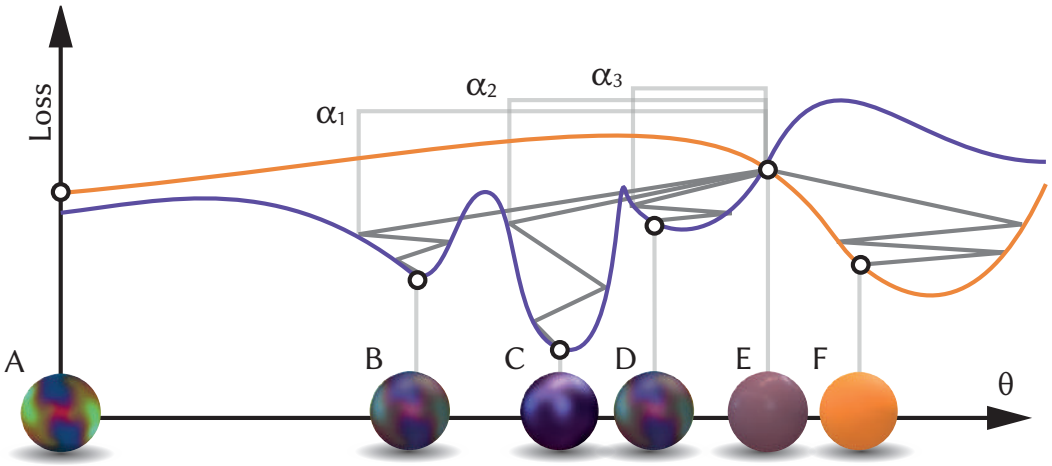


Figure 1.1: A schematic overview of how meta-learning facilitates a traversal of the cost landscape: by moving the NN's initialization from A to E, the general time-to-goal distance to the valid goals C and F is shortened. Moreover, by adjusting the stepsize to α_2, the algorithm ensures to not converge to degenerate solutions B or D but can reach both correct solutions equally well.

lack in quality, as the entire dataset must be compressed into the networks current set of weights [50; 119; 107]. Finally, the *fine-tune* approach creates a preliminary result by executing a general network and then launches a second, non-differentiable overfitting procedure to improve the individual prediction quality. This is not end-to-end differentiable and thus leads to unnecessary features being encoded in the general stage, while still not being possible at interactive runtimes [108; 52].

Metappearance combines the advantages of these different approaches by using meta-learning [63; 172] to make the fine-tuning stage differentiable. In essence, Metappearance optimizes over gradient descent itself, so as to adjust the model's initialization and the subsequent learning's stepsize. Adjusting the initialization leads to an inductive bias that allows the model to generalize to new, unseen instances, while adjusting the stepsize enables inference at interactive runtimes (less than one second), at indistinguishable visual quality. Due to the nature of the algorithm, a further benefit of our approach is that the resulting meta-trained networks are much more data-efficient than their traditionally-trained counterparts.

1.1.2 Plateau-Reduced Differentiable Path Tracing

The second work presented in this thesis (Chapter 4) tackles the problem of plateaus in inverse rendering. A plateau is a flat region in the cost landscape that prevents tradi-

Figure 1.2: An exemplary visualization of an inverse rendering problem with a plateau, and our solution. In a rigid optimization, rotating the cup around its z-axis introduces a plateau in the cost landscape (the blue curve) as soon as the handle is occluded by the cup. In our formulation (middle image), the smoothed cup's handle always is (slightly) visible and hence continuously influences the objective, leading to the much smoother orange curve.

tional gradient-based optimization from converging, as the gradient on the plateau is zero, leading the optimization to stall. In this work, we take inspiration from research on differentiable rasterization [179; 237; 38; 259], which has shown that replacing discontinuities by smooth approximations (such as the Sigmoid function) allows gradient flow and hence enables end-to-end differentiation of the rasterization process. Unfortunately, rasterization is limited in the amount of fidelity it can represent [237], as it only solves a simplified, single-bounce version of the rendering equation and (without significant extension) hence is unable to produce sophisticated light transport phenomena like subsurface-scattering, global illumination or caustics. In this work, we therefore extend the idea of "differentiation via blurring" to the domain of path tracing and propose a novel algorithm that convolves the high-dimensional rendering equation with a Gaussian smoothing kernel, leading to a smoother cost landscape. We develop a scheme to efficiently importance-sample this novel equation and show our algorithm's superior convergence on various inverse rendering applications. Our work can be applied as a straight-forward extension to both black-box and differentiable renderers and works on problems with sophisticated light transport, such as refractions, caustics and global illumination.

1.1.3 ZeroGrads: Learned Local Neural Surrogates

The third work presented in this thesis builds upon the previous chapter and extends the applicability of plateau-free inverse rendering to arbitrary graphics pipelines. To

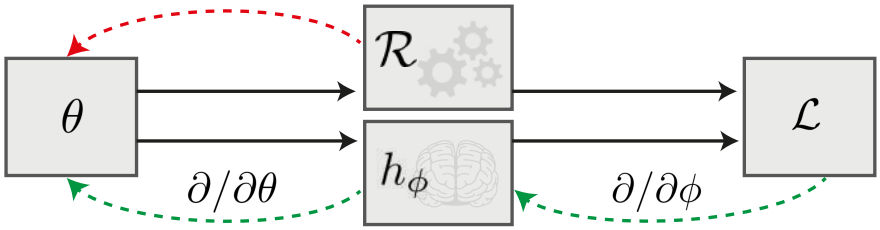


Figure 1.3: While regular forward models $\mathcal{R}$ might not be able to provide gradients w.r.t. the scene parameters θ (red arrow, top), our approach, ZeroGrads, enables this via a learned, differentiable surrogate function h (green, bottom) that maps parameters θ to their associated loss values and can be differentiated analytically.

this end, we introduce ZeroGrads, a novel algorithm that allows differentiating trough arbitrary graphics pipelines (Chapter 5). While we have previously established that a traditional renderer can be modified to produce usable gradients (a more thorough review will be given in Chapter 2), this is a highly non-trivial, manual task which requires significant implementation time, expertise and the use of a domain-specific, dedicated programming language [12; 324; 131]. In ZeroGrads, we instead observe that while we might not be able to calculate the renderer's *gradients*, sampling its *loss values* is trivial when provided with a reference, even under arbitrary graphics pipelines. The key idea of ZeroGrads therefore is that we can fit a differentiable *surrogate* function to these loss samples and then differentiate the surrogate to obtain a gradient estimate. As querying the loss function might be expensive and in large parts irrelevant for the current gradient step, we propose a *local* surrogate that is trained self-supervised alongside the parameter we optimize via an efficient sampling scheme. We show that our neural surrogate can be used to successfully differentiate through various non-differentiable graphics problems, such as visibility in rendering or combinatorical problems and discrete parameter spaces in procedural modeling, and show that ZeroGrads scales well to higher dimensions, where other derivative-free optimizers struggle to converge.

With the contributions now outlined, the following chapter will provide an overview over the relevant concepts and terms used throughout this thesis, before Chapters 3, 4 and 5 will then present the aforementioned individual contributions, respectively.

Chapter 2

Background

2.1 A Primer on Computer Graphics

Computer graphics concerns itself with the art of transforming raw data (e.g., image textures and triangle meshes) into images that humans can observe, understand and admire. Over the past decades, the field of graphics has evolved drastically, from first principles such as Bézier curves [88] and ray casting [265] to inverse renderers [169; 133; 237], new geometry representations [226; 147] and data-driven, neural image generators [105; 263; 332]. It is impossible to cover all these aspects in detail in the limited framework of this thesis, so the following section will give a broader overview over the general concepts and then focus on explaining the relevant terms and algorithms for the subsequent chapters in greater detail.

2.1.1 Image Formation

To understand the presented contributions to differentiable rendering in Chapter 4 and Chapter 5, we will first review the forward model of the rendering process. Rendering usually starts from an image formation model (e.g., a pinhole or fisheye camera) and a scene description containing several geometry primitives (spheres, boxes, cylinders, triangles forming meshes, ...) and light sources. From these, an image can be rendered with numerous techniques, the most prominent two being ray (path) tracing and rasterization.

Ray Tracing uses the camera's extrinsic (position and orientation) and intrinsic (focal length, optical center and skew) parameters to formulate the equation of a ray

through each pixel of the camera's image plane. For a pinhole camera model, this is usually expressed as $r(t) = o + td$, where o and d are the ray's origin and direction, respectively.

The purpose of this ray is to determine the color of the pixel that is currently being processed. Since we know the ray's equation and the scene primitive's position, we can test the ray against intersections with all primitives, yielding the ray's *hitpoint* in the scene. Once the ray intersect a primitive, the resulting hitpoint provides access to the primitive's material properties, which will be used for shading computations. Additionally, a second second ray is cast towards the light sources in the scene, determining whether the current hitpoint has a direct line of sight to a light source (i.e., is illuminated), or whether it is occluded and thus in shadow. Depending on the ray tracer's configuration, several other surface interactions might occur, such as (inter-) reflection, refraction, or scattering events.

Rasterization, the other prevalent rendering technique, projects the 3D positions of scene vertices onto the camera's 2D image plane via the camera's model, view, and projection matrices, which results in 2D triangles in screen space. All triangles are clipped against the view frustum, i.e., the visible portion of the screen, and triangles covering multiple pixels are rasterized by interpolating vertex attributes (e.g., normals or texture coordinates) across the triangle's surface using barycentric interpolation. Per-pixel lighting calculations, such as Phong shading [26], then determine the final color. Since there is no equivalent to the direct visibility query from ray tracing, occlusion between overlapping triangles must be resolved in a post-projection step via comparing the depth along the camera's viewing axis (z-test).

Both rasterization and ray tracing have distinct advantages and downsides. Since rasterization relies on projection, which in turn is realized via matrix multiplication, it lends itself well to parallelization on modern graphics processing units (GPUs) and thus can rasterize millions of triangles in a matter of milliseconds. However, due to the projective nature of the rasterization process, it is limited to *local* lighting and its extension to non-local phenomena requires advanced techniques like shadow-mapping [288; 29; 180; 261] or screen-space reflections [191; 110]. Ray tracing, on

the other hand, must test each ray against all primitives in a scene[1], which leads to longer per-pixel processing times, but is easily extendable to model non-local interactions (see Sec. 2.1.3). Modern rendering pipelines thus combine the best of both approaches by leveraging rasterization for the base image generation, while using ray tracing for selective visual effects like reflections.

2.1.2 Visual Appearance

Regardless of the employed rendering algorithm, the fidelity of the final image is primarily dictated by the scene's material properties, which describe the interaction of light with the intersected surfaces. Notably, realistic *visual appearance* [55] is achieved through creating plausible and pleasing visual patterns or materials across all positions and incident light- and view-angles in the scene. Since the reproduction of visual appearance is the motivation of Chapter 3 of this thesis, the following section will detail the relevant appearance modalities.

Textures. In the simplest case, we can ignore angular dependence and describe visual appearance as *texture* [138; 60; 82]. Textures are usually defined on a 2D image grid and then mapped onto a surface by interpolating the vertices' texture coordinates. They can be realized in many ways, with the simplest and most common form being an image texture, an array of RGB values. As such, the texture itself – in contrast to other modalities like BRDFs – does not exhibit view- or light-dependence (even though the applied shading model might). We distinguish between stationary and non-stationary textures; the former have uniform visual characteristics across space [240], i.e., any random patch will share the same statistics as the original texture, while the latter class does not enforce this and thus can be used to model spatially-varying appearance.

In contrast to artist-designed images, textures can also be generated procedurally [58] in various ways, a topic relevant for both Chapter 3 and Chapter 5 in this thesis. Early work combined noise patterns of varying frequency and amplitude to achieve realistic-looking variants of water-, glass- and marble-textures [234; 235]. Those

[1]This can be accelerated via techniques like bounding volume hierarchys (BVHs) and backface-culling, but the general statement holds regardless.

Figure 2.1: Textures, created (left to right) by photography, artists, procedural modeling, style transfer [82], Perlin noise [234] and a generative model [118]. The insets show the algorithms' inputs, where applicable.

early principles laid the cornerstone for a whole research field of texture synthesis and are at the core of many of the texture generators in modern modeling tools (for an excellent overview we refer to the survey of Raad et al. [247]).

Additionally, textures can be *synthesized*, either from real-world images [59; 112; 197], or from noise. In the context of this thesis, it is important to note that the community has developed ways of transforming noise into *arbitrary* textures which closely match the visual statistics of a target image. One particularly notable approach in this line of research is the "neural algorithm of artistic style" by Gatys et al. [82], where noise is transformed such that its Gram matrices match those of a reference image in the feature space of a neural network (the VGG-19 architecture [277]). The insight that the inner product in VGG feature space aligns "unreasonably well" [333] with the human perception of visual statistics sparked great interested in the community and was foundational for the research area of neural style transfer. Notable approaches that will be re-used in Chapter 3 of this thesis are feed-forward texture synthesis [302], where the lengthy optimization procedure of the original work by Gatys is replaced by a single network execution (or forward pass), and methods like Ada-In [125], which transfer the style of one image to another by aligning their statistics, specifically mean and variance, in VGG feature-space.

It should further be mentioned that there exist learned approaches that train networks to output plausible textures without relying solely on the style transfer literature, such as generative adversarial network (GAN)- or diffusion-based approaches, which regress to the distribution of plausible images, conditioned on an input image or text prompt [262; 76; 118]. They are out of scope of this thesis and only included here for completeness. Fig. 2.1 shows examples of the different textures created by the mentioned approaches.

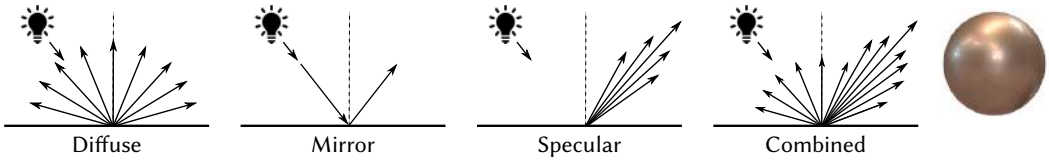


Figure 2.2: Schematic types of surface reflectance for an incident light direction. The right column shows a measured BRDF from the MERL database [189] rendered onto a sphere using the Cook-Torrance shading model and illuminated by an environment map.

BSDFs. Contrary to textures, which vary across space but not across angle, bi-directional scattering distribution functions (BSDFs) describe the way a surface receives and emits light, and vary across angle, but not across space. Generally speaking, the BSDF is a function which receives the incoming and outgoing light directions (ω_i and ω_o in Eq. 2.1) and determines the ratio between the incident and emergent light energy. BSDFs can be sub-classed into bi-directional transmittance distribution functions (BTDFs), which describe how surfaces transmit light and shall be ignored here for brevity, and BRDFs, which describe the reflectance properties of a surface [210; 91]. The BRDF determines whether the incoming light is reflected diffusely, i.e., scattered in all directions with similar magnitude, or specularly, i.e., reflected in a mirror-direction, with the intensity of the specular component determining the strength of the specular highlight (the width of the lobe). Real materials usually exhibit a combination of diffuse and specular reflectance, as shown in Fig. 2.2, while being *energy-preserving*, i.e., they can not emit more energy than they receive, and *Helmholtz-reciprocal*, i.e., ω_i and ω_o are interchangeable.

The accurate representation of BRDFs has been a topic of ongoing research in the graphics community. Traditionally, BRDFs are measured using a Goniophotometer, a spherical device that holds the material to be measured in the center and captures numerous measurements from varying illumination- and viewing-angles. The measurements for each view-light angle pair are then stored in a large look-up table, which, during later BRDF evaluation, will be queried and interpolated. Since this capturing approach is expensive, time-consuming, requires delicate calibration and yields large datasets that require extensive post-processing, alternative methods expressing the BRDF as parametrized, mathematical models have emerged. These

have the advantage that they are faster to evaluate and lend themselves well to optimization, e.g., by regressing the model parameters from real-world observations.

A simple yet historically significant shading model is the Blinn-Phong model [26], which approximates surface reflectance using a linear combination of diffuse and specular components as well as ambient lighting. It describes the Lambertian, diffuse term as the dot-product between incident light direction and surface normal, while the specular component is expressed as the dot product between the viewing direction and the half-vector (the vector halfway between the light- and viewing direction), raised to the power of a glossiness parameter controlling the sharpness of the specular highlight. However, the simplicity and computational efficiency of the Blinn-Phong model make limiting assumptions such as uniform surface characteristics, which is why it has largely been replaced in modern graphics pipelines by more intricate and physically-accurate shading models such as the Cook-Torrance [41], GGX [310], Ward [314] or Beckmann [18] BRDFs. These models express surface variations using microfacet-theory, representing the surface as a distribution of sub-pixel polygons with differing orientations, each reflecting light differently. This allows modeling effects like roughness (via increasing the microfacet's normal variations), geometric attenuation (by the facets masking each other) and accurate Fresnel or energy conservation, resulting in superior visual fidelity compared to the simpler Phong model.

The challenges in reconstructing the parameters of these models from sparse, real-world measurements have motivated numerous approaches, including those explored in Chapter 3 of this thesis.

svBRDFs, the next-higher level of visual appearance, combine textures and BRDFs to capture variations across angle *and* space. They are widely used in media and industry and constitute the majority of modern graphics assets. Similar to textures, a spatially-varying BRDF (svBRDF) is defined on a 2D grid and can be queried using texture coordinates. For each pixel in the grid, the shading model's parameters are defined via *maps*, which are essentially textures encoding the per-pixel reflectance properties of the asset. These maps typically include diffuse and

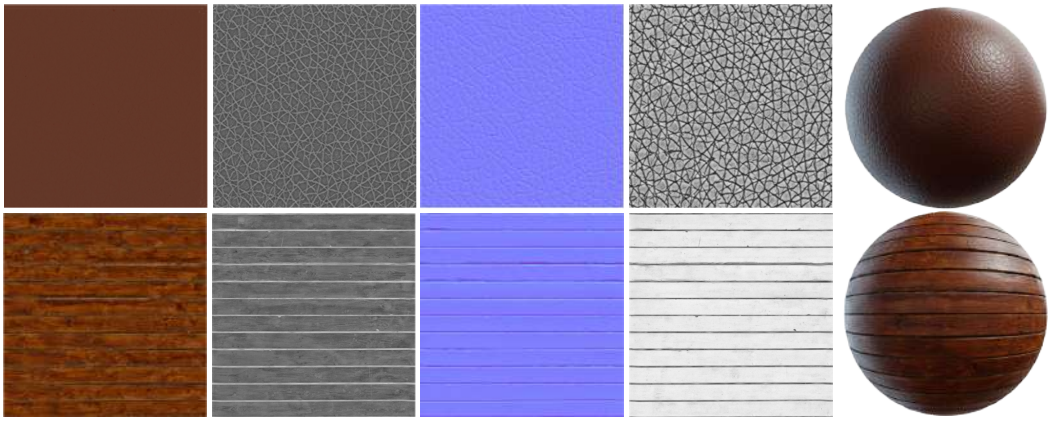


Figure 2.3: svBRDF maps of a stationary (leather, top) and a non-stationary (wood, bottom) material, from polyhaven.com, with the final path-traced rendering on the right.

specular albedo, roughness, and often additional data such as normal and height for bump- or displacement-mapping. During rendering, these values are queried and interpolated based on the hitpoint's texture coordinates, and then passed to the employed shading model to compute the final appearance of the surface.

Similar to textures, svBRDFs can be classified as either stationary or non-stationary, depending on whether their maps exhibit the same visual characteristics across all spatial sub-patches. Similar to BRDFs, they can be captured using either a Goniophotometer or light stage, or regressed from measurements and observations via inverse rendering (for a more detailed description, see Sec. 2.1.4).

2.1.3 The Rendering Equation

In addition to accurate visual appearance, photorealistic rendering needs to model accurate light transport in the scene. Building on Sec. 2.1.1, the rendering equation, introduced in the seminal work of Kajiya [140], extends the ray tracing algorithm to full light paths by describing how light transports in a scene. The equation describes the combined radiance[2] leaving a point $\mathbf{x}$ in the outgoing direction ω_o as

$$L_r(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_\Omega f_r(\mathbf{x},\omega_i,\omega_o)\, L_i(\mathbf{x},\omega_i)\, (\mathbf{n}\cdot\omega_i)\, \mathrm{d}\omega_i, \tag{2.1}$$

[2]The total amount of energy per unit area per unit solid angle emitted or reflected by a point towards the direction of an observer. In general, L_r, L_e, f_r and L_i can additionally be parametrized by wavelength λ and time t, which we omit here for brevity.

Figure 2.4: Light transport phenomena that can be modeled by the rendering equation. From left to right: caustics created from a glass, global illumination bleeding from the walls onto the boxes, reflections and refractions on the glass and multi-bounce light transport, and soft shadows.

where L_e is the radiance *emitted* by $\mathbf{x}$, and L_i is the *incident* radiance from direction ω_i, modulated by the surface's BRDF f_r and attenuated by the angle between the surface normal $\mathbf{n}$ and the incident direction ω_i from the upper hemisphere Ω.

To accurately determine the incident radiance L_i at $\mathbf{x}$, one must integrate over all possible directions covering the unit hemisphere around $\mathbf{n}$. However, the radiance $L_r(\mathbf{x}')$ a neighbouring point $\mathbf{x}'$ emits towards $\mathbf{x}$ is itself again dependent on the L_i it receives from *all other* points in the scene (see Fig. 2.5). This recursive behaviour of light transport, sometimes colloquially referred to as the "curse of dimensionality", thus makes the rendering equation a self-referential integral that cannot be solved analytically for even the most modest of cases.

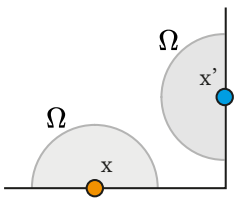


Figure 2.5

It is, however, precisely this recursive nature that allows the rendering equation to express many common forms[3] of light transport with great accuracy and physical realism – the caustics that a glass of wine casts onto a table's surface, the soft falloff of an occluder's shadow or the light bleeding from one diffuse surface onto another – and that makes it so appealing to light transport researchers (Fig. 2.4 visualizes these examples). While the graphics community has accepted that (to date) there is no way of analytically solving the rendering equation, research has employed several ways of *numerically approximating* the underlying integral, the most salient one being Monte Carlo (MC) integration, a concept upon

[3] Some popular light transport phenomena that remain elusive for the original rendering equation are participating media like smoke or fog, polarization, fluo- or phosphorescence and wave-optics phenomena like diffraction or interference.

which Chapter 4 and Chapter 5 rely heavily.

In Monte Carlo integration, the intractable integral is approximated by a discrete sum, averaging the contributions from random samples drawn from the integration domain. Importantly, MC has the convenient property that it allows us to estimate a solution by simply evaluating the integrand at these random samples. Another advantageous property of MC methods is that their complexity and convergence rates are not affected by the problem's dimensionality.

In the general case, a Monte Carlo estimator of an integral can be written as

$$\int_{\Omega} f(x)\mathrm{d}x \approx \frac{1}{N}\sum_{i=0}^{N}\frac{f(x)}{p(x)}$$

where x are random samples on the integration domain Ω, drawn according to a probability density function (PDF) p. Indeed, it is this stochasticity that is etymological to the Monte Carlo estimator: the result of a single execution of $f(x)$ will vary, depending on the chosen sample[4]. The *average* over many samples, however, will be very close to the true value of the integrand – in fact, arbitrarily close, given sufficient samples.

In the context of the rendering equation, we calculate a separate MC estimate for each pixel in the image plane by repeatedly sampling light paths into the scene starting from this pixel. At each intersection of the path with the scene, the surface's properties modulate the energy throughput along the path, such that the final pixel color is determined by all the ray's interactions with the scene. After a number of scene interactions, or "bounces", we terminate the ray and add the final pixel color to the current pixel's estimate.

The ray's bounces usually are constructed by sampling a random direction vector at each ray intersection. Unsurprisingly, the uniform PDF is not proportional to the integrand in most cases. While MC will still return a correct result in the limit, this discrepancy between $f(x)$ and $p(x)$ will lead to "wasted" samples, i.e., rays that do not hit a high-energy region of the scene (such as a light source)

[4]The Las Vegas estimator, in contrast, will always yield a correct solution (in the information-theoretical sense, as in, per specification) but vary in runtime [25; 117].

before being terminated, and thus do not markedly contribute to the pixel's color estimate. The graphics community has therefore put significant effort into devising efficient sampling patterns and distributions which bias uniform sampling towards regions of higher energy. This process, known as importance sampling, better aligns the samples' PDF with the integrand and in turn reduces the estimator's variance, leading to faster convergence. The number of importance-sampling techniques is vast, ranging from cosine-hemisphere, light- and BSDF-sampling over more sophisticated techniques like next-event estimation (NEE) [243], Metropolis light transport (MLT) [305], neural radiance caches (NRCs) [202] or Gaussian mixture models (GMMs) [309], potentially combined via multiple importance sampling (MIS) [304].

However, computing the corresponding sampling density p, which is crucial for an unbiased estimator, might be non-trivial. One of the ways in which recent, learned importance sampling techniques circumvent this problem (and which is the basis of one of the approaches in Chapter 3) is through the use of normalizing flows (NFs), a family of neural networks that convert one sampling distribution into another while simultaneously providing the converted samples' PDF [54; 150]. Popular examples include neural importance sampling [200] and primary sample space (PSS) sampling [335; 145][5]. In PSS sampling, a NF is trained to map random-uniform samples to high-throughput samples that are concentrated in the important regions of the scene's PSS. The NF simultaneously provides the samples' PDF, which enables an unbiased MC estimator and significantly reduces variance in challenging rendering scenarios, such as scenes with complex lighting, caustics, or glossy surfaces.

On the contrary, a known, analytical PDF can be importance-sampled via inverse transform sampling [53], also called the Smirnov transform. To do so, we integrate the PDF to get its cumulative distribution function (CDF), and then invert it either via algebraic transforms or via precomputed lookup-tables [71; 205; 222] to get the *inverse* CDF (ICDF). Feeding random-uniform numbers into the ICDF generates random samples that are distributed according to the original PDF.

Moreover, one of criteria that the PDF of an unbiased MC estimator must fulfill

[5]The PSS is a $2(k+1)$-dimensional hypercube representing all possible light paths in a scene, i.e., all sampling directions for each light ray's k bounces.

is $p(x) > 0$ for all x where $f(x) \neq 0$, ensuring that all contributing points x are sampled with non-zero probability. It is therefore not obvious how to importance-sample for functions with negative values, as we cannot construct PDFs that align well with the sub-zero regions of the function. As a remedy, Owen and Zhou [224] introduce a technique termed positivization, where the function is decomposed into subparts that are either entirely positive or entirely negative, and a separate PDF is used for sampling either part before combining the results via MIS. Ascertaining that the used PDFs are proportional to the function's absolute value $|f(x)|$ allows computing an unbiased low-variance estimator on functions with negative values, a finding that will be re-used in Chapter 4 of this thesis.

2.1.4 Inverse Rendering

Finally, while the previous sections have described the *forward* rendering process, it is its *inverse* that is at the core of this thesis. Contrary to computer vision, whose goal is to establish scene understanding and -reasoning from images or measurements (e.g., for semantic segmentation, object detection, etc.), inverse rendering describes the process of recovering the unknown variables of parametric models used to model the scene or world – e.g., the parameters of a Phong BRDF. Indeed, early work on inverse rendering focused on lighting- or reflectance estimation [188; 187], with Ramamoorthi and Hanrahan [252] framing inverse rendering as a deconvolution between lighting and BRDF and recovering both illumination and BRDF from photographs, and Yu et al. [326] presenting a method for recovering the reflectance properties of scene objects from a sparse set of images.

Inverse rendering, however, should be distinguished from *differentiable* rendering (which uses gradients to drive the optimization process), since inverse rendering can be performed entirely gradient-free. Papas et al. [225], for instance, use simulated annealing (SA), a gradient-free optimizer, to regress the parameters of a heightfield that creates a reference caustic, while Li et al. [167] use reinforcement learning (RL) to find the parameters of a material graph given a reference image.

Differentiable rendering, in contrast, is the process of deriving and propagating gradients throughout the rendering process, which in turn can then be used to drive

inverse rendering scenarios. Popular examples include volumetric rendering, where neural networks (NNs) [193; 194; 14; 15] or primitives [147; 124] are fitted to encode real-world scene captures, or reflectance estimation [3; 50; 51; 52], where svBRDF parameters are regressed from image observations.

While differentiable rendering will be treated separately in Sec. 2.3, it should be noted that modern inverse rendering is often done in the neural network domain, since the NN's smoothness lends itself well to optimization. How exactly these networks are optimized will be discussed in the next section.

2.2 Learning and Optimization

For reasons of brevity, this thesis will not give in-depth explanations about deep learning theory or detail the specific neural networks or architectures used. While it has been shown [168] that architectural decisions certainly do influence the learning behaviour of these models (e.g., the use of residual- or skip-connections [264; 104]), we can – in a more general sense – abstract the network as a function f_ϕ, which consumes some input θ and whose parameters ϕ control the function's behaviour and can be adapted through different techniques. We assume the reader to be familiar with basic machine learning (ML) concepts like multi-layered perceptrons (MLPs) and convolutional neural networks (CNNs). For an excellent overview of the plethora of additional deep learning architectures and techniques, we refer to the textbook by Goodfellow [86].

2.2.1 Gradient-based Learning

The powerful advantage of using learning-based approaches is that they can be fit to observed measurements and data, thus allowing us to iteratively improve upon our imperfect, possibly hand-crafted models. In modern machine learning and neural network optimization, the cornerstone method is undoubtedly gradient descent, particularly its first-order variant. The concept is straightforward: after feeding some input θ through our model or function, we can compare its prediction to the corresponding ground-truth data (also called target or reference) and calculate the error e:

$$f_\phi(\theta) = \hat{y}, \quad e = \hat{y} - y\,.$$

Subsequently, in order to minimize the model's error (i.e., to maximize its usefulness), we find an analytical expression for the cost landscape[6] and then descend a small step against to the *gradient* of this landscape, i.e., into the direction of steepest descent, to decrease the model error and hence improve its prediction for the next iteration. Iterative application of this process will converge in a valley of the loss landscape,

[6]The terms loss and cost are used interchangeably in the ML literature. Both are to be differentiated from the model's *error*, which simply is the difference between the model's prediction and the corresponding ground-truth data. The loss (or cost) is an additional function that is *acting* on this error, popular examples are the mean squared error (MSE) or Kullback-Leibler divergence (KLD).

where the slope is zero, and hence yield a (local) minimum. Interestingly, gradient descent works agnostic to the parameters that are being updated: we can use it to decrease the error by updating either the model's parametrization ϕ (the common scenario in NN training, where the network weights ϕ are changed to better fit some data) or by assuming a fixed function f_ϕ and updating the model's input θ (a common scenario in inverse rendering). Additionally, there exist hybrid combinations where both the model input and parameters are adapted simultaneously; examples are meta-learning (Sec. 2.2.2) and joint radiance field- and camera optimization [228].

The mainstream popularity of gradient descent in the modern machine learning literature is owed to several factors. First, it scales well to billions of parameters and is trivially parallelizable, across both parameters and compute devices. Second, it is integrated with most modern machine learning frameworks through dedicated, efficient and easy-to-use automatic differentiation (AD) libraries like PyTorch's AUTOGRAD [231], JAX's AUTODIFF [31; 30] or Tensorflow's GRADIENTTAPE [1]. Explicit gradient information therefore is readily available, a necessary precondition for gradient descent in contrast to derivative-free or rule-based systems (discussed later in Sec. 2.3.4). Finally, gradient descent benefits from a long history of research, enjoys strong convergence guarantees (under convex objectives) and has proven to be easily robustifiable to optimization noise.

Importantly, gradient descent is most often used in its first-order variant, due to factors like computational efficiency. As such, it computes the *local* gradient at the current position in parameter space. For the case of noisy or rugged loss landscapes, this can be extremely detrimental, as it makes the algorithm susceptible to getting stuck in local minima. A concept that alleviates this is the notion of *stochastic* gradient descent (SGD), which, instead of processing the whole input dataset before gradient evaluation, computes the gradient on a stochastic subset, or (mini-) batch, of the dataset. This concept, introduced originally to alleviate computational constraints practitioners were facing with large datasets, has surprisingly been shown to improve convergence of gradient descent in the presence of noise [61; 102]. This is attributed to the fact that the stochasticity introduced by mini-batch sampling not only reduces

computational costs but also acts as a regularizer, helping models escape sharp minima and find flatter, more generalizable solutions, where low loss values are achieved even in (or precisely due to) the presence of stochastic noise.

Overall, most AD frameworks compute the gradient of the loss w.r.t. the optimization parameters via recursive application of the chain rule, which allows expressing the gradient of a composition of functions as the product of the functions' individual gradients. For the common example of a scalar loss function $\mathcal{L}$ acting on the model's prediction error, this can be expressed as

$$\mathcal{L}(\hat{y}, y) = \mathcal{L}(f_\phi(\theta), y) \tag{2.2}$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial f_\phi(\theta)} \cdot \frac{\partial f_\phi(\theta)}{\partial \theta}\,. \tag{2.3}$$

Notably, the chain rule is employed purely for convenience: one could just as well derive manual expressions for each parameter's gradients using pen and paper, and directly hard-code them in the gradient update step. This increased implementation effort would lead to significant savings in computation time and memory and is one of the key drivers of efficient rendering frameworks [157; 147]. Most gradient-based optimizers and applications, however, employ the standard autodiff-backpropagation via the chain rule for ease of development. As we will see later in Sec. 2.3, this convenience sometimes comes at a price. Indeed, in certain scenarios, the naïve application of the chain rule will lead to incorrect gradients.

Without going into detail, it is worth briefly mentioning the role of *optimizers* in learning. Once the gradients are computed, various strategies exist for updating the model parameters. The standard stochastic gradient descent (SGD) update rule subtracts the gradient, scaled by a learning rate, from the current parameters. More advanced methods extend this with auxiliary hyperparameters or leverage additional information to improve convergence. One early example is the momentum- or heavy-ball strategy proposed by Polyak [239], which adds inertia to the descent direction to smooth out oscillations (i.e., the etymological heavy ball's descent does not get stalled by local minima as easily). The more modern Nesterov momentum

[209] extends this by computing the gradient at a point slightly ahead of the current parameters, effectively ”looking ahead” to correct the update direction and reduce oscillations. The popular Adam optimizer [149] combines momentum with adaptive learning rates by maintaining moving averages of both the gradients (first moment) and their squared values (second moment), enabling smoother gradient estimates and faster convergence. In spite of its popularity, it should be noted that Adam is not free of flaws; recent research, for instance, has shown that its per-parameter normalization does not preserve rotation equivariance for vector-valued parameters, leading to suboptimal optimization results [173].

In summary, gradient descent can well be described as the workhorse of modern machine learning optimization. Yet, despite its many advantages, it struggles with poorly conditioned problems, saddle points, local minima and vanishing or exploding gradients, requiring techniques like gradient clipping, learning rate scheduling, and preconditioning.

For completeness, it should be mentioned that there exist higher-order descent methods that leverage more information about the loss landscape, such as curvature, to improve convergence. Popular examples are the LBFGS [175; 241] or Levenberg-Marquardt [165; 186] algorithms and Newton-Raphson-type methods [244]. These methods, while offering improved convergence, come with their own set of limitations, such as computational complexity due to the use of Hessian information and often suboptimal scaling with problem dimensionality, hindering their adoption in mainstream machine learning literature. Since they are not actively used in this thesis, we will defer their discussion to rendering-specific sub-problems in Sec. 2.3. Similarly, the antonym of gradient-based optimization, derivative-free optimization (DFO), will be discussed in Sec. 2.3.4.

The following sections will outline the fundamentals of two applications of gradient-based learning: *meta-learning*, where models learn to optimize efficiently, and *differentiable rendering*, where (proxy-) gradients are used to solve inverse rendering problems.

2.2.2 Meta-Learning and Learning to Learn

Meta-learning lifts learning to the meta-domain by optimizing over gradient descent itself. The core idea behind *gradient-based* meta-learning[7] is to repeatedly optimize a model's performance on a base learning task, and then, using the information gathered from these previous optimization runs, improve the model's performance on future, unseen individual tasks. In the seminal Model-Agnostic Meta-Learning (MAML) algorithm introduced by Finn et al. [63], the learning task therefore is divided into two steps: first, the model performs an *inner loop*, where a limited number (usually between five to ten) of gradient descent update steps are taken. This is followed by an outer loop, which utilizes the final performance of the inner loop as optimization target and takes a step along the *meta-gradient* to improve the model's performance during the next inner loop iteration. This meta-gradient, flowing back from the outer through the inner loop, can be thought of as carrying the information necessary for "learning from previous optimizations", i.e., taking into account how the model adapts under gradient descent steps at the current iteration (Fig. 2.6 gives a schematic overview of the inner-outer-loop separation and gradient flow). The

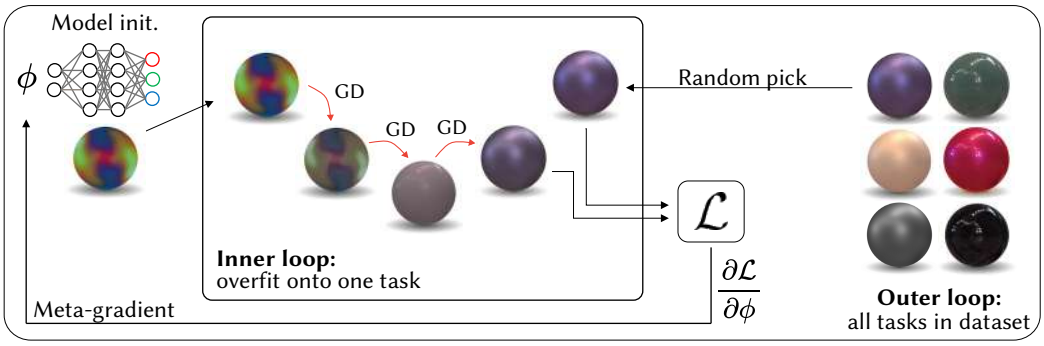


Figure 2.6: Schematic overview of the MAML algorithm: In the outer loop, a single task is sampled from the set of learning problems and processed in the inner loop, where the learner takes a small number of gradient descent (GD) steps towards the task reference, before a final loss $\mathcal{L}$ is computed, whose gradient is then backpropagated to the model initialization.

model therefore learns a prior (the literature often uses the term "inductive bias") over the base task in the outer loop, while the inner loop is used to quickly adapt this

[7] For completeness, it should be mentioned that there exist gradient-free meta-learning strategies like learning the parameters of evolutionary algorithms [158], which are out of scope here.

general knowledge to specific tasks. Examples of such priors and individual tasks are numerous; the physics underlying a quadruped's locomotion and the quadruped moving to a specific target location in a navigation task, or the stochastic appearance pattern that forms a texture and the actual realization of a specific wood or stone texture in a texture synthesis tasks.

Meta-learning therefore fills the gap between the more traditional learning paradigms overfitting, general inference and fine-tuning (for a more thorough explanation of this taxonomy, see Chapter 3) by extending this landscape with a "learned (general) fine-tuning" approach: by leveraging gradient information *through* the inner loop, i.e., through gradient descent itself, we can take into account how the model updates over time and optimize for this behaviour in an end-to-end fashion. MAML thus allows adaption to new, *unseen* tasks from the same domain in only a few gradient descent steps (thus achieving *fast* inference within seconds) and with high accuracy. The algorithm has been applied to the model's initialization [63; 296], SGD's per-parameter learning rates [172], sampling patterns for BRDF reconstruction [174] or smoothing functions [319]. MAML has sparked significant follow-up work: CAVIA [339] meta-learns a context-agnostic set of base parameters and only modulates a small set of task-specific parameters, WarpGrad [73] introduces warp-layers that meta-learn a smoother optimization landscape and Deleu et al. [46] meta-optimize across the number of inner loop optimization steps.

An important distinction is to be made between meta- and hyper-learning, although the terms are sometimes used interchangeably. The consensus in most of the literature, and in this thesis, is that meta-learning directly optimizes over the parameters of a model, whereas hyper-learning uses one model to directly, in a feed-forward manner, predict the parameters (for instance, NN weights) of a second model, which will then perform some downstream task. Hyper-learning therefore effectively bypasses the need for task-specific optimization during inference, gaining in memory efficiency at the cost of generality. Adapting the hyper-network is trivial since it produces weights of another network, i.e., the parametrization of a (smooth) function, through which we can easily calculate gradients.

Another flavor of meta-optimization is learning-to-learn (L2L), where networks learn to directly predict parameter updates or gradients given past observations. This approach has first been popularized by Andrychowicz et al. [6] and since then sparked significant follow-up work. The underlying idea to use a network which, conditioned on the previous optimization states, directly outputs new parameters or, more commonly, the per-parameter *gradients* that are suitable for the optimization problem. Typically, L2L uses recurrent neural networks (RNNs), mostly long short-term memory (LSTM) networks [113], since they can reason over time by compressing previous optimization information into their hidden state, thus allowing memory savings since the entire optimization trajectory needs not to be stored in memory or the weights of a neural network. For the case of short optimization trajectories, *unrolling* is sometimes used, where an RNN's sequential layers are unrolled along the time axes and stacked along the network's depth. This enables gradient predictions in a single network forward pass but is limited to small networks and few time steps due to otherwise prohibitive memory consumption. Adler and Öktem [2] show that this technique can be used to reconstruct tomography images, while Flynn et al. [74] use unrolling for novel view synthesis.

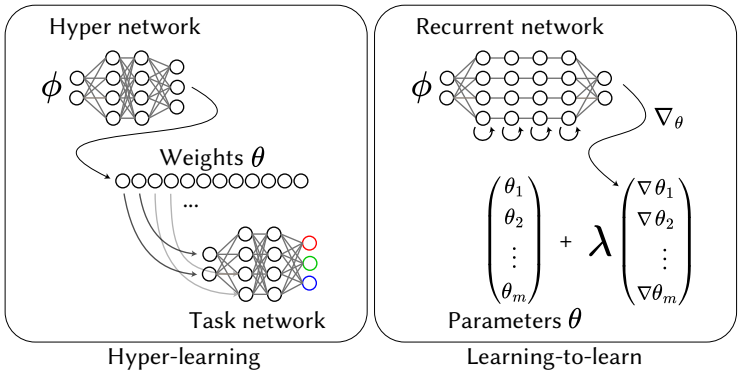


Figure 2.7: Schematics of hyper-learning and learning-to-learn.

In general, however, L2L approaches often perform inferior to meta-learning methods like MAML or CAVIA, and are harder to train due to the brittleness of LSTM training. They are not used throughout this thesis, so they will not be further discussed here. The area might, however, experience renewed interest with the recent release of the extended LSTM (xLSTM) architecture [17].

2.3 Differentiable Rendering

With a solid foundation about light transport and optimization, we can now approach the core of this thesis: the problems that arise when differentiating the rendering process. Note that rendering becomes easily differentiable when executed by a neural network [206; 193; 298; 155; 332], which is usually a problem-specific solution and thus unfit for *general purpose* differentiable rendering. The following section will therefore review the general-purpose differentiable rendering approaches with relevance to this thesis by first formulating a concise list of problems in differentiable rendering and then discussing current approaches in conjunction with their respective solutions to these problems.

2.3.1 Problems with Rendering Gradients

Problems in differentiable rendering can arise in the following scenarios:

(a) **Gradients are undefined.** At point singularities or -discontinuities, e.g., the kink in the ReLU, the function is continuous but its gradients are undefined. In theory, this could cause gradient-based optimizers to diverge. In practice, however, this surprisingly poses little problem, since many frameworks have hard-coded numerical values for these edge cases or resort to sub-gradients. Moreover, these discontinuities are very unlikely to be hit in an optimization.

(b) **Gradients are zero.** This more prominent scenario occurs for two reasons: Firstly, the renderer might use functions that are non-differentiable, i.e., whose derivative is zero everywhere and then gets multiplied during autodiff's chain rule application, leading to zero gradients altogether. Secondly, long-range dependencies in the optimization might not be accurately reflected by the employed objective function, which often operates pixel-wise, leading to regions of zero change (plateaus) in the objective, which effectively stall the optimization due to the gradient being zero.

(c) **Gradients are unknown.** This occurs when we are dealing with non-differentiable pipelines or languages, such as Blender or OpenGL. Since the framework is not written in a language that supports native AD, we cannot

easily backpropagate the image-space error to the current parameters. Rewriting the required operations to be differentiable is a manual and tedious process that does not scale, and even hard-coding the gradient expressions might be impossible due to performance or hardware constraints.

(d) **Gradients are memory-expensive.** This issue stems from the fact that autodiff-frameworks record and store all forward computations in order to run backpropagation, and arises when differentiating through a MC path tracer: since naïve application of AD and the chain rule must store each light bounce for each ray for each pixel as a node in the computation graph or gradient tape, rendering (and differentiating) a full scene requires prohibitive amounts of GPU memory for even modest resolutions[8].

(e) **Gradients are incorrect.** Additionally, MC path tracing might suffer from *incorrect* gradients for the case of discontinuous integrands, which frequently occur, e.g., for positional- or BRDF-derivatives. Naïvely applying automatic differentiation to the MC estimates of the pixel radiance will yield incorrect gradients here due to the fact that, under discontinuous integrands (e.g., the silhouette of an object moving through the pixel), Leibniz' rule of differentiation under the integral sign is violated and the interchange of integration and differentiation – which is precisely what AD is doing – is no longer valid.

(f) **Gradients are noisy.** The two most prominent causes for the issue of high-variance gradients in differentiable rendering are noise from MC sampling, which can be tackled via established techniques like importance sampling or control variates, and noise from re-building gradient estimates from scratch at each optimization iteration, a situation that is common for SPSA-style estimators.

The following sections will now detail the differentiable renderers that are used throughout this thesis and those of scientific relevance in the context of differentiable rendering as it is used here, referring back to the above list of problems as Prob. (i).

[8] For an impressive visualization of such a graph see the suppl. video to [220], 4:45min onwards.

2.3.2 Differentiable Rasterization

The main issue in differentiating the rasterization process is the first case of Prob. (b): non-differentiable functions in the rendering pipeline. Specifically, the Heaviside step function in the edge- and z-tests (see Sec. 2.1.1), which returns either one or zero and whose gradient thus is undefined at the step and zero everywhere else.

Evidently, the use of this function in conjunction with the chain rule will lead to zero gradients for the entire pipeline. Differentiable rasterizers therefore use several techniques to prevent this: a traditional, hard rasterizer can be made differentiable by either using soft approximations of the non-differentiable step function or by approximating the gradients themselves during the backward pass.

Soft Approximations. The problem with using a hard rasterizer during the forward pass is illustrated in Fig. 2.8: no (infinitesimally) small change to the parameters θ_1 or θ_2 will change the color at pixel P_1, since the triangle does not overlap the pixel. Assuming a non-negligible difference in depth, a similar argument can be made for the occluded vertex of the smaller triangle behind pixel P_2: no small change will bring the vertex to the front, therefore, the image-space loss will not change, which leads to the gradient w.r.t. this parameter being zero everywhere.

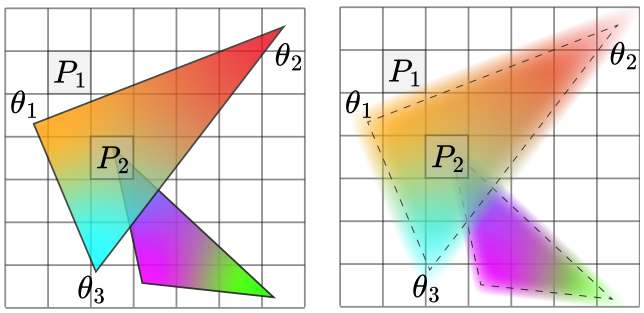


Figure 2.8: Hard (left) and soft (right) rasterization.

However, as illustrated on the right side of Fig. 2.8, we can use a soft approximation of the rigid step function, e.g., a Sigmoid, whose smoothness can be controlled by the slope's inclination. This leads to the hard edge-tests becoming *smooth*, i.e., the parameters θ_1 and θ_2 now *continuously* influencing the pixel color at P_1. In a similar spirit, the hard z-test can be replaced by a *smooth aggregation* function, usually chosen to be a variant of the softmax-operator, such that *all* triangles under the pixel of interest contribute to its final color and thus receive gradients during backpropagation. In combination, these operations allow the image-space gradients to flow from pixels to (potentially distant) triangle

vertices and create a differentiable rasterizer that can optimize 3D attributes from pure 2D supervision. Note that the other operations employed in a rasterizer, such as projections and vertex interpolation, are continuous and differentiable by design and do not need special treatment.

The first to leverage this principle for differentiable rasterization were Liu et al. [179]. Interestingly, they formulate the Sigmoid decision boundary that models the influence of a parameter on a pixel as *probability*, which establishes interesting connections to variational optimization, discussed in Sec. 2.3.4. The optimal choice of smoothing function (or probability distribution), however, is non-trivial and problem-dependent, as shown by Petersen et al. [237], who conduct a study comparing different approximations and aggregation functions. Wu et al. [319] therefore propose to meta-learn the optimal smoothing function.

The idea of replacing non-differentiable functions with smooth approximations is the basis of the approach presented in Chapter 4 of this thesis.

Interestingly, these soft approximations come with the drawback of making the final image blurry, since hard surfaces now become transparent around their silhouettes, necessitating schemes of scheduling the blur over the course of the optimization or hyperparameter tuning.

Analytical Derivatives. The second way of making a rasterizer differentiable, as mentioned in the beginning of this section, is to keep the original, hard rasterizer during the forward pass, but to approximate the gradients during the backward pass. In their seminal work OpenDR, Loper and Black [182] derive local pixel-space gradients via differential filtering, i.e., repeated execution of the standard forward rasterizer with small perturbations in positive and negative x- and y-direction, akin to a Sobel filter. While this works for simple scenarios, this locality leads to poor optimization performance, since semi-distant triangle vertices cannot influence the current pixel color and therefore are not optimized correctly. Similarly, Kato et al. [143] propose the neural mesh renderer (NMR), which approximates gradients during the backward pass by linearly interpolating between the values found left and right of the discontinuities and then uses the slope of this interpolated line as

gradient signal. Both methods are unable to model non-local pixel-vertex interactions and to optimize a triangle's depth [179]. NVDiffRast [157] introduces a different approach to differentiable rasterization by employing a triangle coverage test based on barycentric coordinates for analytic antialiasing. After finding all triangles on silhouette edges via discrepancies of their triangle IDs, they use distance-to-edge antialiasing (DEAA) to propagate visibility and occlusion gradients between adjacent triangles. This involves calculating where and to what extend the line connecting the pixel centers is intersected by the silhouette, which produces a blend weight that determines how much each adjacent pixel's property (e.g., color) affects the final pixel value. This ensures smooth gradient flow across edges and occlusions and enables depth and visibility optimization.

Note that all the above approaches do not necessarily strive to produce *correct* gradients, but are rather focused on *useful* gradients, as mentioned in the excellent survey by Kato et al. [144]. Metz et al. [192] additionally show that gradient correctness does not necessarily entail usefulness in an optimization.

Blurring. Combining the idea of analytical derivatives and soft approximations, the concept of blurring in image space has also been used for differentiating through the rasterization process. Reddy et al. [254] propose a smooth aggregation function for SVG images and additionally regularize their approach via multi-resolution processing and spatial smoothing through Gaussian pyramids [255]. Rhodin et al. [259], on the other hand, propose a novel scene model which replaces traditional, hard primitives (and their corresponding discontinuous visibility changes) with Gaussian densities. These can be analytically ray-traced and lead to smooth visibility falloffs, allowing for proper gradient flow from image to 3D properties.

As with soft approximations, blurring in image space usually requires careful tuning and scheduling of hyperparameters (e.g., the standard deviation of the Gaussian), since we usually are comparing the blurred image to a hard reference. A similar blurring-trick will be reused in Chapter 4 and Chapter 5 of this thesis, albeit in parameter- instead of image space.

2.3.3 Differentiable Path Tracing

While (differentiable) rasterization is limited to modeling local shading, general-purpose differentiable MC path tracing can backpropagate through the entire light transport in a scene and thus is a versatile tool for optimizing secondary effects like global illumination or BSDF parameters. However, this introduces certain problems that need to be solved, serving as one of the motivations for Chapter 5 in this thesis.

Memory Consumption. As mentioned in Prob. (d) of the initial list of gradient problems, in order to be able to run the gradient computation *backwards* through the computation graph, the entire *forward* model must be stored in memory. This requires storing all computations at each ray-scene intersection (i.e., each bounce) of each ray of each pixel, which quickly becomes prohibitive. As an elegant remedy, the *adjoint* formulation has been proposed by both Nimier-David et al. [220] and Stam [287]. The idea is that, instead of storing all the light rays and then "walking back" their paths through the scene during backpropagation, the rendering process is split into two phases: a primal and an adjoint rendering pass. The primal pass computes a standard MC path traced image, which is then compared to the reference under an objective $\mathcal{L}$. The adjoint pass propagates a novel quantity, the so-called "differential radiance", through the scene (see [220] for a mathematical definition). This quantity is emitted by objects whose gradients we would like to calculate, e.g., the brightness of a light source in the scene, and scatters through the scene like regular radiance, making it amenable to the benefits of decades of MC rendering research. Crucially, the differential radiance received by the camera sensor is a MC estimate of the gradient of $\mathcal{L}$ w.r.t. the scene parameters, i.e., the gradient of the forward rendering. In addition, reformulations via self-adjoint operators allow exchanging the propagation directions: Nimier-David et al. [220] show that it is possible to scatter differential radiance *from the sensor* into the scene, the same way regular radiance scatters, and thus vastly reduce the estimate's variance, saving on computation and time.

One issue of this formulation, however, is that the derived equations for the differential radiance reference the primal incident radiance, thus coupling the two

light transport simulations. This requires recursive integral evaluations for both quantities, leading to a quadratic memory footprint in the number of scattering events. While Nimier-David et al. [220] propose a version of the adjoint algorithm that circumvents this problem at the cost of bias, Vicini et al. [306] show that this bias can have severe consequences for the optimization and instead propose to "replay" the primal paths with the same random variates, leading to a scheme for unbiased gradients with linear memory footprint.

Discontinuous Integrands. Another problem that differentiable MC path tracing has to take into account are potential discontinuities in the radiance integrals, as mentioned in Prob. (e). Let's take positional derivatives as a motivating example: given the current pixel $P(\theta)$ depending on the scene parameter θ, the standard way of estimating the derivative of P w.r.t. θ is tracing a few rays through P before calling automatic differentiation to get the gradients[9]. In short, we are interested in the derivative of the pixel integral (top) and its MC estimator (bottom)

$$P(\theta) = \int_\Omega f(x,\theta)\mathrm{d}x \tag{2.4}$$

$$\hat{P}(\theta) = \frac{1}{N}\sum_i f(x_i,\theta) \tag{2.5}$$

which, according to Leibniz rule of differentiation under the integral sign with constant boundaries [72], can be expressed as

$$\frac{\partial}{\partial\theta}P(\theta) = \frac{\partial}{\partial\theta}\int_\Omega f(x,\theta)\mathrm{d}x = \int_\Omega \frac{\partial}{\partial\theta}f(x,\theta)\mathrm{d}x \tag{2.6}$$

$$\frac{\partial}{\partial\theta}\hat{P}(\theta) = \frac{\partial}{\partial\theta}\sum_i f(x_i,\theta) = \sum_i \frac{\partial}{\partial\theta}f(x_i,\theta)\,. \tag{2.7}$$

In particular, for the discrete case which we are computing (and differentiating) in practice, an estimator of the gradient simply is the gradient of the estimator, expressed as the sum of the individual samples' derivatives. However, moving the derivative operator *inside* the integral is only valid if the integrand (and its derivative)

[9]Note that the previous section on accelerating AD in differentiable MC path tracers is completely orthogonal to this section: using the adjoint formulation without special consideration of positional derivatives will yield incorrect gradients, while the memory footprint of both re-sampling and reparametrization will still be prohibitively large without the adjoint formulation.

exist and are continuous in the parameter to be differentiated [183].

For the initial example of positional derivatives, this is not the case around silhouette edges: once the edge starts moving into the pixel, the integrand will at one point inevitably jump, namely when the edge crosses a sampling location, leading to a discontinuity in the integrand and thus the violation of the right-hand equality in Eq. 2.7. However, automatic differentiation – being unaware of this fact – will still compute the right side of Eq. 2.7, resulting in incorrect derivatives (see the right part side of Fig. 2.9 for an example).

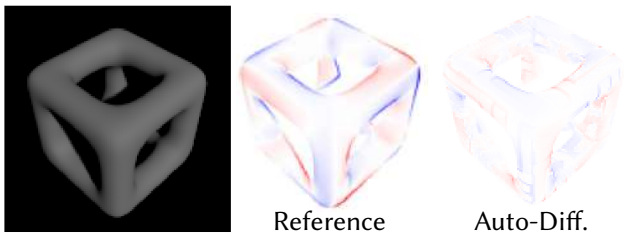


Figure 2.9: Gradients w.r.t. a rotation of the cube, in blue and red for pos. and neg. magnitude, respectively. Figure adapted from [183].

The literature therefore employs two main approaches for resolving this problem: edge sampling and integral reparametrization. Both approaches remark that the pixel integral can be decomposed into a continuous part which can be automatically differentiated without issues (e.g., Phong shading or texture interpolation, similar to the continuous operations in differentiable rasterization in Sec. 2.3.2), and a *boundary* term, which causes discontinuities and thus requires special treatment.

The main idea behind edge-sampling originates from the fact that traditional area-sampling does not accurately capture the small variations caused by, e.g., moving triangles. This becomes visible in Fig. 2.10, which displays a pixel and two enclosed triangles: in the left part, the blue triangle is moving upwards, but the change in covered pixel area (and thus in pixel color) is not reflected by the area samples (in black). Li et al. [169] therefore propose to sample the pixel *on* the triangle edges (middle part in Fig. 2.10). The key idea here is that each edge divides the space into two

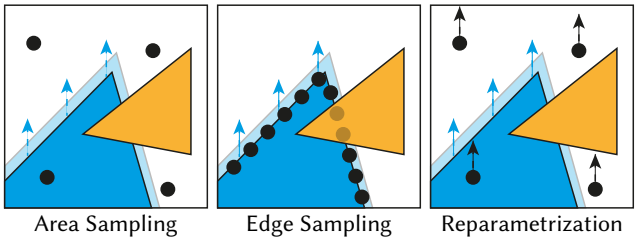


Figure 2.10: Sampling methods for discontinuous integrands. Figure partially inspired by [169].

half-spaces (with different color values, since they are on different triangles), and that the final pixel color is a weighted combination thereof. Since edge-sampling yields samples exactly on the boundary between the two half-spaces, moving them will change the ratio between the colors and thus continuously influence the final pixel integral, leading to correct and unbiased gradients for the boundary term. Additionally, occlusion is resolved correctly by design, since moving occluded samples does not contribute to the boundary integral.

This technique is, however, non-trivial to extend to secondary visibility and to importance-sample, since there are millions of triangles in a scene and infinite possibilities of placing samples on the triangles' boundaries, without a clear way of determining the regions of high importance. Loubet et al. [183] therefore propose an alternative, the so-called reparametrization solution. They observe that the integrand's discontinuities are caused by changing θ and propose to remedy this by integrating in a space where the samples *follow*, i.e., move in tandem with, the parameter θ. This is achieved via a change of variables in the integral, effectively offsetting the samples' position by the same amount with which the discontinuity (or the triangle causing it) moves (right side in Fig. 2.10). The resulting integral can be solved by standard MC techniques and differentiated via AD, since it is now smooth in θ. Using an efficient implementation of this reparametrization via spherical convolutions, the reparametrization method outperforms edge-sampling in terms of speed by an order of magnitude.

Both techniques, while being able to successfully differentiate through the MC path tracing process in a general-purpose fashion, require expert knowledge about rendering and light propagation and significant implementation effort, constraining their use to the graphics community and their respective implementations, Redner [169] and Mitsuba [129]. Chapter 5 of this thesis therefore asks the question whether we can "get away" with something simpler; making less assumptions, requiring less implementation effort and supporting more (arbitrary) forward models. The necessary fundamentals will be discussed in the subsequent section.

2.3.4 Variational and Gradient-Free Optimization

Contrary to the previously discussed approaches, both variational- and derivative-free optimization (DFO) aim to make as few assumptions as possible about the underlying model that is being optimized. While Jamieson et al. [134] show that the convergence of DFO methods is generally inferior to gradient-based methods, this generality and their applicability to arbitrary forward models make them attractive for optimizing non-differentiable programs or languages. The field of DFO can be divided into methods that try to *estimate* a gradient, usually via function evaluations, and methods that try to find local minima via *population sampling*, e.g., through evolutionary strategies. DFO is complemented by variational optimization, which takes a probabilistic approach closely tied to Bayesian modeling and aims to descend along a smoother, stochastically perturbed function of the objective with similar local minima as the original objective function.

Gradient Estimators. Gradient estimation techniques can further be grouped into zeroth- and first-order methods. Zeroth-order methods do not use any derivative information and use function evaluations to determine the estimated descent direction. A classic example are finite differences (FD), which evaluate the function under small perturbations to determine the slope of the function response. The combination of forward and backward difference, the *central* difference, is defined as

$$\frac{\partial f(\theta)}{\partial \theta_i} \approx \frac{f(\theta + \varepsilon\, \theta_i) - f(\theta - \varepsilon\, \theta_i)}{2\varepsilon}, \tag{2.8}$$

where ε is a hyperparameter usually set to a small number and θ_i is a one-hot unit vector modulating only the i-th component of θ. As ε approaches zero, this approximation becomes more and more accurate [221], but Eq. 2.8 already reveals the main drawback of this method: since we need to modulate θ component-wise, estimating the full gradient ∇_θ requires $2n$ function evaluations for an n-dimensional parameter. One remedy to this approach was introduced by Spall [283] via the simultaneous perturbation stochastic approximation (SPSA) algorithm: instead of perturbing each dimension individually, SPSA perturbs all dimensions at once, and

estimates the gradient as

$$\frac{\partial f(\theta)}{\partial \theta} \approx \frac{f(\theta+\varepsilon p)-f(\theta-\varepsilon p)}{2\varepsilon p}. \tag{2.9}$$

Notably, p here is a *vector* of mean-zero random variates, leading to the etymological *simultaneous* perturbation. SPSA thus allows to estimate a gradient with only two function evaluations per iteration independently of problem dimension; a significant improvement to the $2n$ evaluations required by finite differences, especially as dimensionality increases. This estimator, although derived from a different perspective, can be interpreted as a special case of the plateau-reduction approach presented in Chapter 4 of this thesis.

The idea of gradient estimation via measurements of the loss function can be extended via (local) fitting of a parametric model to these loss samples. This class of algorithms, often summarized as *surrogate-* or *model-based* gradient estimators, form the basis of the approach discussed in Chapter 5 of this thesis.

Gradient-Free Optimizers. In lieu of analytic or estimated gradients, the predominant other class of algorithms is the one of search-based, gradient-free optimizers, which will validate the approach discussed in Chapter 5 and shall thus shortly be reviewed here.

An ever-popular, illustrative example of gradient-free optimizers are genetic algorithms (GAs), where an initial population of candidate solutions evolves over time. The idea behind GAs (and, in fact, many other evolutionary strategies like particle swarm optimization (PSO), the bat-algorithm or ant colony optimization), is that this population's fitness is measured by an objective function that closely resembles (often: is identical to) the loss function that we seek to optimize. Each individual population member, after traversing a part of the search space, is then assessed through this objective function and only the "fittest" members are allowed to procreate – thus spawning new offspring candidates – and advance to the next generation, while the remaining solutions are discarded. Evolutionary algorithms thus incrementally hone in on promising regions of the search space while keeping the exploitation-exploration balance through random mutations of promising individuals.

The covariance matrix adaptation evolution strategy (CMA-ES) extends this process by modeling the underlying procreation- and mutation-probabilities as distributions and adapting their covariance matrices over time. While respecting this correlation between solution candidates and time steps significantly aides optimization performance, it also requires the storage of the quadratic covariance matrix, which quickly becomes expensive for high-dimensional problems.

Contrary to evolutionary strategies, simulated annealing (SA), another popular gradient-free optimizer, models the exploration of the search space via the analogy of annealing metals in metallurgy and thermodynamics. Based on an initial candidate solution, new solutions are sampled around the current solution based on a temperature parameter determining the sampling radius. The temperature parameter "anneals", i.e., diminishes, over time, thus limiting the sampling radius (and thus the exploration of the search space), as well as the probability of accepting an inferior solution in hope of future reward. SA uses purely random exploration, not taking into account how well individual solutions perform and as such is preferable over evolutionary or gradient-based methods when an approximate, global optimum is more desirable than an accurate, local minimum.

The Achilles' heel of gradient-free optimizers, however, is the dimensionality of the optimization problem to be solved. While they might work well on low-dimensional problems, the increased inter-variable dependencies, search space extent and exacerbated noise that come with more dimensions often lead to non-convergence when problem dimensionality increases. Chapter 5 shows how the spectral bias of neural networks can be leveraged to circumvent this limitation.

Variational Optimization. Having examined gradient estimators and search-based approaches, we now turn to variational optimization, where parameter updates often rely on probability distributions over candidate solutions. The intuition here is often that, although the original function might not be differentiable, we can differentiate over the *expectation of a perturbed version* of this function. Building on results from Gumbel [94], an illustrative example is provided by Berthet et al. [22] through their *perturbed optimizers*, which allow the differentiation

of arbitrary functions through variational perturbation. The idea is that, instead of computing the gradient of a single function, we compute the gradient of a perturbed ensemble of function evalations, where the perturbations follow certain characteristics (e.g., for the case of [22], are from a zero-mean distribution). Contrary to the single-evaluation case, changes to the optimization parameter θ will then result in changes of the *expected value* of this ensemble and thus yield smooth changes even on plateaus or discrete spaces, given the right choice of distribution parameters. Fig. 2.11 illustrates this: the black, rigid parameter would achieve a zero gradient, since it lies on a constant region of the solution space. The variational interpretation (the blue circle) instead achieves a non-zero gradient, since moving the parameter will now *continuously* influence the expected loss.

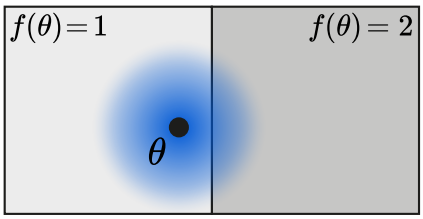


Figure 2.11

Le Lidec et al. [161] apply this principle to rendering and successfully differentiate through a rasterizer. Staines and Barber [285, 286] further remark that the difference between the variational objective, i.e., the loss function under perturbation, and the rigid objective can be made arbitrarily small under the assumption that the employed distribution allows the contraction of its probability mass to a sufficiently high density, such as the Gaussian distribution, for instance.

In the context of robotics, variational optimization is also termed stochastic smoothing [57] and used for differentiating through non-differentiable events such as collision or contact [291; 196]. Finally, Chaudhuri and Solar-Lezama [36, 37] establish the term “smooth interpretation” and show that entire programs can be made differentiable through Gaussian perturbations, a finding which both Chapter 4 and Chapter 5 will heavily rely on.

With the fundamentals of image formation, light transport, learning and differentiating through these processes now covered, the following chapters will introduce the individual algorithms that form the main contribution of this thesis.

Chapter 3

Metappearance: Meta-Learning for Visual Appearance Reproduction

Abstract

There currently exist two main approaches to reproducing visual appearance using Machine Learning (ML): The first is training models that generalize over different instances of a problem, e.g., different images of a dataset. As one-shot approaches, these offer fast inference, but often fall short in quality. The second approach does not train models that generalize across tasks, but rather over-fit a single instance of a problem, e.g., a flash image of a material. These methods offer high quality, but take long to train. We suggest to combine both techniques end-to-end using meta-learning: We over-fit onto a single problem instance in an inner loop, while

Figure 3.1: We propose meta-learning for a wide range of appearance reproduction tasks. Given as few as 10 optimization steps, our method (top in each subfigure) achieves quality comparable to overfit-approaches (bottom in each subfigure) that take orders of magnitude more training iterations.

also learning how to do so efficiently in an outer-loop across many exemplars. To this end, we derive the required formalism that allows applying meta-learning to a wide range of visual appearance reproduction problems: textures, BRDFs, svBRDFs, illumination or the entire light transport of a scene. The effects of meta-learning parameters on several different aspects of visual appearance are analyzed in our framework, and specific guidance for different tasks is provided. Metappearance enables visual quality that is similar to over-fit approaches in only a fraction of their runtime while keeping the adaptivity of general models.

3.1 Introduction

Reproduction of visual appearance [55] is a key part of Computer Graphics that has achieved new levels of simplicity, speed and accuracy thanks to recent developments in ML. The classic use of ML for appearance reproduction was to capture light or materials from very little input, sometimes only single images [50; 84], without access to ground truth maps. Approaches that are capable thereof usually train for a long time on large datasets and achieve impressive levels of generalization, often due to CNNs that recognize patterns in the data. Unfortunately, this generality comes at the price of not matching the target precisely: we might get a great looking BRDF or svBRDF from a single image, but it might not exactly match the input.

More recently, a second line of research has evolved, where no attempt is made to generalize over a large dataset, and, instead, non-linear optimization and differentiable rendering are used to explain visual appearance in input images [82; 193; 249]. These methods minutely match the reference, but need many input observations, take long to train and can be slow to execute. Typically, such approaches use point-operations, e.g., MLPs, rather than CNNs.

A first step to combine these two training paradigms was introduced by adapting the output of a model from the general class in a second, non-end-to-end step, the so-called fine-tuning or post-refinement stage [52; 108; 78; 95]. Approaches that use fine-tuning usually run an additional number of gradient steps (in the order of magnitude 10^3) towards a specific target, which greatly improves reconstruction

quality, but inflates runtime to the order of minutes, whereas feed-forward CNNs operate in milliseconds.

A dilemma materializes: Should one rather make a user wait in order to provide them with high quality output, or would it be better to provide fast, interactive results that might be of inferior quality? Both solutions are unsatisfactory, which is why in this work, we aim to diminish this quality-speed-gap and provide quality that is a) close to model-overfitting or fine-tuning, and b) available at interactive runtimes, close to those of general feed-forward networks.

We achieve this by harnessing the power of *meta-learning*: building on the MAML algorithm [63] from the machine learning community, our framework Metappearance uses two nested optimization loops, where the outer loop is sequentially presented with all exemplars in a (training) dataset. For each exemplar, the inner loop is then tasked with over-fitting a model onto this specific exemplar. Characteristically, the inner loop operates under the constraint of a very limited number of available gradient descent steps, typically around 10 only. Metappearance hence learns to efficiently drive the inner optimization towards a specific target, but still is able to exploit coherency and priors in the data due to knowledge gathered in the outer loop.

In this chapter, we present a framework that formalizes the application of meta-learning to the task of visual appearance reproduction. Importantly, we do not propose new visual appearance methods or new loss functions, nor do we compare methods or analyze their properties. In fact, quite the contrary: we keep the methods the same, but instead propose a different way of training them. By comparing our approach against "traditional" training paradigms, we show which types of applications can benefit from meta-learning and explore the implications on performance and quality. We validate that Metappearance outperforms general inference followed by fine-tuning through ablation- and convergence-studies. Additionally, we, for the first time in the graphics literature, make the connection between meta-learning, model compression and data efficiency. We show that Metappearance speeds up faithful appearance reproduction by several orders of magnitude, while keeping all desirable properties of the respective base approaches and similar visual quality.

In summary, our contributions are

- Metappearance[1], a model that adapts to new, unseen visual appearance tasks in only a few steps of gradient descent.
- Optimizing for a fast and accurate optimizer of this model.
- Instances of this model that accurately match texture, BRDFs, svBRDFs, illumination, or light transport orders of magnitude faster than strong baselines, at comparable quality, and
- An analysis of our method's properties, its convergence and its behaviour under ablation.

3.2 Previous Work

3.2.1 Visual Appearance

We consider visual appearance reproduction, the task of generating plausible and accurate visual patterns across all positions and orientations from evidence captured for some angles and locations.

Ignoring angle and considering an exemplar's statistics, we would talk about appearance as *texture* [138; 60; 82]. When angle matters, we would call this bi-directional reflectance distribution function (BRDF) [210; 91] and when both space and orientation are considered, spatially-varying BRDF (svBRDF) [43]. Guarnera et al. [91] summarize these approaches. Textures [302; 82], BRDFs [84] and svBRDFs [50] have all been acquired and represented by means of ML, for which Tewari et al. [297] provide a survey. We defer discussion of the specific existing solutions for all those sub-problems to Sec. 3.4.1.

3.2.2 Learning

More important to our problem is how the different methods are trained, i.e., optimized, given either the information of a single instance or an entire set of exemplars (Tab. 3.1).

[1]The code is available at `https://github.com/mfischer-ucl/metappearance`.

Table 3.1: Different ways to optimize for visual appearance reproduction. We acknowledge that encoding visual appearance in neural networks comes at the cost of editability (rightmost column).

	General	Fast	Accurate	Compact	Editable
`General`	✓	✓	×	×	×
`Overfit`	×	×	✓	✓	×
`Finetune`	×	×	✓	×	×
`Meta` (ours)	✓	✓	✓	✓	×

General Learning A typical paradigm is to collect a training dataset, say, 2D images, to curate them with appearance supervision, e.g., BRDF parameters, and to learn a mapping from the image to those parameters, for example through a CNN [84]. Often, such methods create a latent space. While it is a strength that this space will mostly contain valid exemplars, it comes at the expense of a bottleneck, reducing specific details. In simple words, a 100-dimensional latent space can ensure every latent code is a grass texture, but it cannot represent the exact location of 200 grass blades. Examples of such approaches include work by Henzler et al. [107] (texture), Georgoulis et al. [84] (BRDF) or Deschaintre et al. [50]; Kuznetsov et al. [154]; Gao et al. [78]; Guo et al. [95] (svBRDF) or Zhu et al. [337, 338]; Bako et al. [9]; Huo et al. [127] (light paths). These methods generalize well to new data, but do not *exactly* match the test-time input, and hence are *general* and *fast*, but not *accurate*, as per the taxonomy established in Tab. 3.1. Moreover, they often require an encoder- and decoder branch, which makes them not *compact*. We call these **`General`** and formalize them in Sec. 3.3.2.

Over-fit Optimization A second, more classical approach is to not seek generalization, but to fit a model to samples of a specific problem instance. This technique has seen a recent increase in popularity due to the emergence of coordinate-based neural representations, and often is used in conjunction with MLPs. Examples are numerous and include most works related to neural radiance fields (NeRF) [193] as well as others for texture [155], BRDFs [293], svBRDFs [334] or the entire light transport [335; 200]. We call these methods **`Overfit`** and define them in Sec. 3.3.3. Most **`Overfit`** approaches are *accurate* and *compact* (they usually do not require

an encoder, as they do not need to generalize), but neither *fast* to train nor *general*.

Fine-tuning A combination of above approaches is sometimes used, where first a general network is trained and then, when the target instance is known, is optimized a second time [52; 108]. Some have employed optimization in latent space [294] while keeping the rest of the network fixed [141; 142; 78; 95], or in pixel space after a user-adjusted number of iterations, aiming to fit the target perfectly. We here name these **`Finetune`** and define them in detail in Sec. 3.3.4. Approaches that use fine-tuning or post-optimization usually are *accurate*, but neither *fast* (post-refinement usually happens at non-interactive runtimes) nor *general* (once the model is fine-tuned, it cannot be used for general inference anymore). Most fine-tuning models are *compact*, as it is usually enough to store the fine-tuned decoder and the corresponding latent code ([108]), although this is not always the case ([52].

Hyper- and meta-learning Hyper-networks produce weights of another network [97]. This has been applied to appearance [190; 24], BRDFs [293] and NeRF-like representations [279]. Meta-learning, instead, does not directly produce the parameters of another network, but guides the optimization that drives the inner learning. This optimization is often based on gradient descent, so the outer optimization produces setting such as start values and step sizes. Sometimes, the gradient rule itself is learned [2; 253]. Applications of meta-learning were proposed for geometry [280], super-resolution [120] and animation [312], layered depth images [74], as well as for NeRF by Bergman et al. [21] and Tancik et al. [296].

Approaches that use meta-learning are *fast* and *general* by construction, as they can run inference on new, unseen samples in only a few gradient steps. As we will show in this work, meta-learning for visual appearance reproduction is also *accurate*, as its output is close to overfit- or fine-tuning quality. Moreover, meta-learning enables *compact*ness, as the model initialization and optimization themselves are learned, and hence do not need to rely on latent codes produced by, e.g., bulky encoder networks. We would not be aware of work attempting to model visual appearance using meta-learning, as we set out to do in Sec. 3.3.5.

3.3 Our Approach

After introducing the problem we solve (Sec. 3.3.1), we provide a common formalization of three previous solutions (Sec. 3.3.2, Sec. 3.3.3 and Sec. 3.3.4), and finally introduce Metappearance (Sec. 3.3.5).

3.3.1 Problem statement

We now discuss representing visual appearance, its parametrization, and finally its optimization.

Representation We represent visual appearance as $L_\theta(\mathbf{x}|I)$, a radiance function of a positional-directional coordinate $\mathbf{x}$, conditioned on input I and parametrized by the tunable vector θ. The coordinate $\mathbf{x}$ can be two-, three- or higher-dimensional and might be positional, directional or both. The condition I varies per application and could be a single image, sparse measurements or light paths.

Parametrization Parameterizing L_θ by θ is possible in a large number of ways, for instance through a plain, pixel-based RGB image, spatial data-structures, or a more implicit representation, like a CNN or an MLP, and the parametrization might make use of hard-coded, rendering-like operations. For now, we deliberately do not specify this further and only require L to be differentiable w.r.t. the parameters θ. Table 3.2 will give examples for instances of this model which we will evaluate in our experiments.

Optimization Let us assume a scalar function $\textsc{Loss}(\theta, T)$ that is low if L_θ explains the data T well (i.e., $L_\theta(\mathbf{x}_i|I_i) \approx L_i \; \forall \; i \in T$) and high otherwise. We will specify different ways to define this loss, leading to different approaches of reproducing visual appearance. Let us further assume that we have access to an optimizer function $\textsc{Learn}(\theta_0, \alpha, T, \textsc{Loss})$ which performs gradient descent (GD) that starts at θ_0 to change parameter θ with step size α so as to minimize the loss $\textsc{Loss}$. This procedure is given in Alg. 1, where n_l is the number of GD iterations.

Combining $\textsc{Loss}$ and $\textsc{Learn}$ leads to different methods. In classic learning, both the initialization and optimization step size are hyper-parameters chosen by the user. We will see that **`Meta`**-learning chooses these optimally through optimization.

Algorithm 1 Classic learning: The function `grad`$(\cdot)$ differentiates its first argument (an expression) with respect to the second.

procedure LEARN(θ_0, α, T, LOSS)
    $\theta = \theta_0$
    for $i \in \{1, \ldots, n_1\}$ **do**
        θ -= $\alpha \cdot$ `grad`(LOSS(θ,T), θ)
    end for
    return θ
end procedure

3.3.2 General

`General` methods, that attempt to map a condition I directly to appearance, use the loss described in Alg. 2:

$$\text{LOSS}_{\texttt{General}}(\theta, T) = \mathbb{E}_{i \in T}[\Delta(L_\theta(\mathbf{x}_i|I_i), L_i)] \tag{3.1}$$

where $\Delta(\cdot,\cdot)$ here, and in the following, can refer to any norm. Visual appearance problems usually are ambiguous: One I_i can typically be explained by more than one parameter vector θ. Over the course of training, a **`General`** optimization sees many different conditions I_i and hence can build priors about what solutions are more likely than others. These priors are then used to generalize to new conditions under new angles and positions, e.g., a new 2D photo of a sphere that can then provide reflectance for new 3D angles and positions. However, the encoding of these priors that then handle variations over I must be performed under the constraint of a finite budget of parameters. In typical applications, this results in more or less subtle forms of smoothing: a generated BRDF does not quite resemble the BRDF the input specifies, some spatial details are lost in svBRDFs, etc. We will show examples of this in Sec. 3.4.1.

3.3.3 Over-fitting

Differently, in over-fitting, the loss is

$$\text{LOSS}_{\texttt{Overfit}}(\theta, T) = \mathbb{E}_{i \in T}[\Delta(L_\theta(\mathbf{x}_i|I), L_i)] \tag{3.2}$$

Algorithm 2 Classic loss. The function `sample`$(\cdot)$ takes a set as an argument and returns a random index into that set.

1: **procedure** LOSS(θ, T)
2: $cost = 0$
3: **for** $i \in \{1, \ldots, b\}$ **do**
4: j = `sample`(T)
5: $cost$ += $\Delta(L_\theta(\mathbf{x}_j | I_j), L_j)$
6: **end for**
7: **return** $cost/b$
8: **end procedure**

where, importantly, I is constant and does not depend on i. This task is comparatively easy, as the network only has to deal with one specific input. Consequently, results are often of higher quality than in the **`General`** setting. However, the optimization now lacks the synoptic approach that sees all instances and can use this "bigger picture" to build priors and make do with fewer information in lower time. Typically, over-fitting approaches need many iterations to train, take from minutes to hours to converge, and often require further regularization, e.g., by physical constraints, to avoid overfitting to specific training positions and directions.

3.3.4 Fine-tuning

Both overfitting and generalization can be combined in a trivial way: First run a general method on the input, second, optimize the output so that it resembles the input even more. Fine-tuning usually starts from initial parameters θ_0, that have been trained across many inputs (e.g., the converged state of a general model, as per Sec. 3.3.2), and then optimizes these for a fixed target, as in over-fitting (Sec. 3.3.3), with a fixed step size α_F. This means to compute

$$\text{LEARN}(\text{LEARN}(\theta_0, \alpha_G, T, \text{LOSS}_{\texttt{General}}), \alpha_F, T, \text{LOSS}_{\texttt{Overfit}}). \tag{3.3}$$

While the general step can be re-used across several inputs, the subsequent fine-tuning (essentially, over-fitting) optimization must be repeated for each new input. **`Finetune`** is faster than **`Overfit`**, as only the inner optimization needs to be executed for inference, while the outer step is a feed-forward network execution, and

sometimes is accelerated further by increasing the learning rate α_F. Still, optimization usually takes in the order of minutes, i.e., it is slow compared to a single feed-forward execution of the general network that typically would take milliseconds. Moreover, the solution might diverge from the priors that informed the first step. By jointly training over both the general projection and the fine-tuning stage, we overcome these issues in quality and speed, as explained next.

3.3.5 Meta-learning

A general model's training is agnostic to the fact that later fine-tuning iterations will be used to further improve the results. This drives the **`General`** projection step towards learning unnecessarily detailed representations while missing other important features and over-smoothing the space (see Sec. 3.3.2). The **`General`** step hence will try to incorporate features that fine-tuning might include anyway, and subsequently disregard other, more general elements that the fine-tuning operator might miss.

If we do not know how to trade those properties, could we instead learn how to do that? Could we learn how to perform an optimization optimally? To do that, we need i) a domain to optimize over, ii) to understand what is "optimal", and iii) an actual algorithm. We will now look into these aspects.

As our optimization domain, we consider meta learning of both the initial solution θ_0 as well as a per-parameter step size α. Both the initialization and the step sizes are fixed between tasks and stay constant at test time. We stack θ_0 and α into a *meta parameter* vector, denoted as $\phi = \{\theta_0, \alpha\}$. Meta learning can then be formalized[2] as a new loss (Alg. 3):

$$\text{LOSS}_{\texttt{Meta}}(\phi, \mathcal{T}) = \mathbb{E}_{i \in \mathcal{T}}[\text{LOSS}_{\texttt{Overfit}}(\text{LEARN}(\phi, T_i, \text{LOSS}_{\texttt{Overfit}}), T_i)]. \quad (3.4)$$

The first thing to note is that the loss is defined on meta-parameters ϕ and that it calls LEARN with these, to quantify how suitable they are for an inner learner. Second, it samples from a *space of tasks* $\mathcal{T}$ which consists of multiple optimization

[2]In a slight abuse of notation, as LEARN takes four parameters, while it is called with three here, where the first is a tuple holding the first two arguments, init and stepsize.

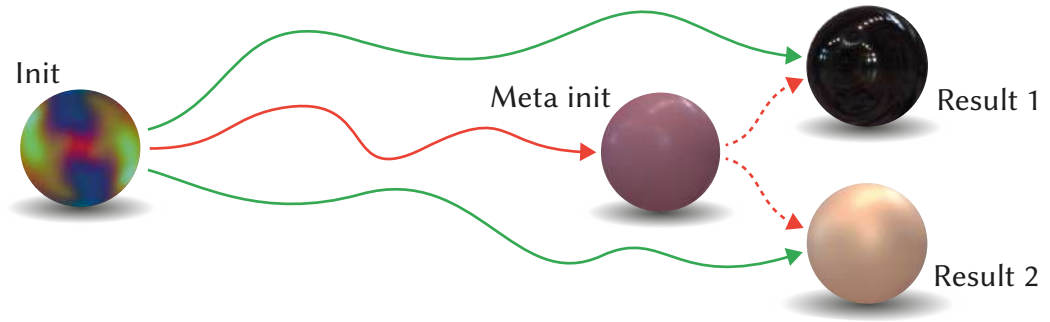


Figure 3.2: Learning the init: Trajectories for **`Meta`** and **`Overfit`** for the example task of BRDF representation. The dotted line denotes inner optimization. Note how the dotted trajectories for **`Meta`** are shorter, i.e., faster learning.

tasks T_i, e.g., multiple BRDFs, instead of a single task. The sampled task T is the same for meta-train and meta-test; the same for the call to LEARN and to LOSS. Because `sample` inside the loss function is randomized, different positions and directions are used for meta-test and meta-train. Doing so, parameters that generalize across positions and directions inside one task are advantaged.

To actually perform meta-learning, we

$$\text{LEARN}(\phi_0, \alpha_\text{M}, \mathcal{T}, \text{LOSS}_{\texttt{Meta}}),$$

i.e., perform common learning with an advanced loss and a meta-initialization, ϕ_0, as well as a meta step-size α_M.

Algorithm 3 Meta-learning involves a loss that depends on the hyper-parameters of calling the function LEARN on the actual task.

```
1: procedure METALOSS(φ, 𝒯)
2:     cost = 0
3:     for i ∈ {1,...,b} do
4:         j = sample (𝒯)
5:         cost += LOSS(LEARN(φ, 𝒯_j, LOSS), 𝒯_j)
6:     end for
7:     return cost/b
8: end procedure
```

By encouraging network parametrizations that enable few-step convergence on unseen samples, meta-learning optimizes over optimization itself. More specifically, in our scenario, the inner optimizer learns to over-fit to the appearance of one

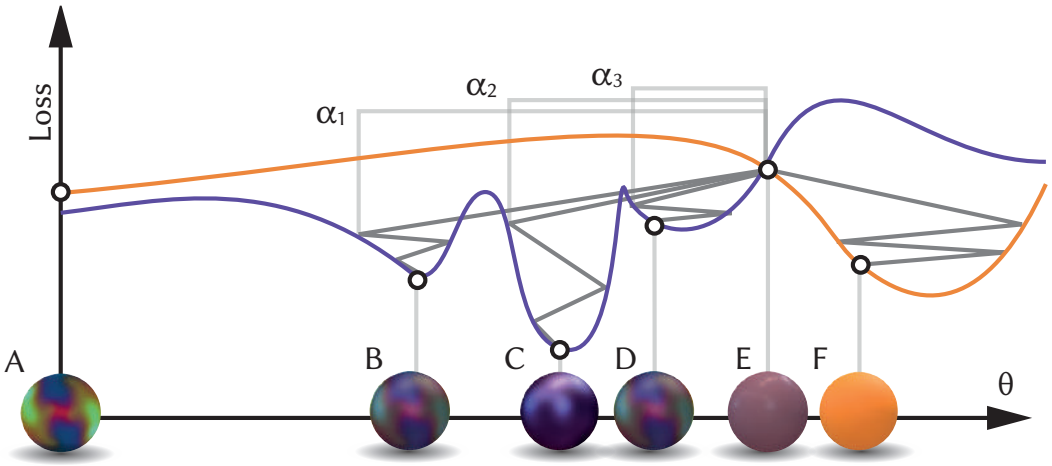


Figure 3.3: Learning the step size: The orange and violet curve show the loss (vertical) for different parameters θ (horizontal) for two BRDF tasks. The gray α-intervals denote three alternative step sizes. The zig-zags are the convergence paths for specific choices of step size. Please see the text for discussion.

exemplar. The outer optimizer then changes the inner optimizer's start parameter values, so that the next inner-loop execution will achieve improved results and do so much quicker. Fig. 3.2 illustrates this idea with two very basic tasks. As with other losses, the metaloss is computed across a batch, i.e., ϕ_0 and α_M are updated with information averaged across multiple optimizations (the `for` loop in Line 3).

Fig. 3.3 illustrates the purpose of learning the step size. As explained in Fig. 3.2, meta-learning will change the init from A to a suitable position E. When choosing the step size right, (α_2 for this init) the optimizer will converge to the correct BRDFs, here C and F. With a step size too small, α_3, or a step size too large, α_1, we converge to less suitable results (D or B, respectively) for the violet task. As the above considerations might be different in higher dimensions, we parameterize the step size as a vector instead of a scalar, which allows anisotropic gradient steps [172]. During meta-inference, i.e., when using the meta-trained model to quickly infer a result for a new, unseen sample provided by a user, the step-sizes are fixed, and only the model weights are changed.

Jointly learning the model initialization and the corresponding step sizes combines the quality of over-fitting with the ability to build priors of general approaches. In practice, the inner training loop takes several orders of magnitude fewer iterations than common over-fitting and is up to two orders of magnitudes faster than

fine-tuning, which enables execution at interactive rates: in most applications, our inference time is less than 1 second.

Implementation Our implementation follows the MAML framework proposed by Finn et al. [63]. As the name suggests, the meta-learner is agnostic to the inner network used, which makes the approach flexible and well-suited for our different application scenarios. We learn our per-parameter stepsize (see Fig. 3.3) according to the approach presented in Meta-SGD [172]. For details of the different meta-learning algorithms and tools used, please see Appendix Sec. A.1.

3.4 Evaluation

We have introduced a framework for using meta-learning for visual appearance reproduction, but how well does it compare to more traditional training approaches? To answer this, we will now demonstrate the effectiveness of Metappearance on a variety of different applications. We will now introduce those, including notes on previous work and the architecture (Sec. 3.4.1), then outline the evaluation protocols (Sec. 3.4.2), and, finally, report qualitative and quantitative results (Sec. 3.4.3).

3.4.1 Applications

We consider six increasingly complex applications (Table 3.2): i) RGB textures, ii) BRDFs, iii) stationary and iv) non-stationary svBRDF maps from flash images, v) illumination maps from RGB images with normals and finally vi) the entire light transport in a scene.

Neither the tasks addressed nor architectures used are novel; the contribution lies in the way they are trained. We re-iterate that it is hence not our goal to compare different *approaches* (e.g., CNN vs. MLP for BRDF encoding), but rather compare different methods of *training* a specific approach. We will detail each application next.

Textures. In a TEXTURE, RGB appearance varies over space, but has uniform visual feature statistics [240]. Gatys et al. [82] optimized for a finite image in pixel space such that its VGG activation statistics match the exemplar, a solution that would be `Overfit` in our taxonomy. Later, Ulyanov et al. [302] trained a single

CNN to perform this task feed-forward. Huang and Belongie [125] have shown how control over (instance) normalization can produce new textures corresponding to a `General` solution in the logic of this work. Henzler et al. [107] show how to do this conditioned on an input image, optionally involving a step of `Finetune`. For a comprehensive survey, we refer the reader to Raad et al. [247].

These methods are exemplary for the spectrum we challenge: either they take long to learn and fit the input exactly, or they are fast and only approximate the input. We study a design based on Ulyanov et al. [302] and Henzler et al. [107] as per TEXTURE in Table 3.2. For the exact network and training setup, please confer Appendix Sec. A.2.1.

BRDFs. While the RGB textures varied in space, but not in angle, we now look into visual appearance varying with angle, but not over space, the classic BRDF representation task. We use a network the learn the BRDF responses for given light- and view-directions. Our experiments follow Sztrajman et al. [293] and Hu et al. [119], who both use networks combined with custom parametrizations to encode the MERL [189] BRDF database. Details on related work and the architectures used are found in Appendix Sec. A.2.2.

Stationary svBRDFs. The next-higher level of complexity are stationary spatially varying BRDFs (SVBRDFSTAT) that combine spatial and angular variation of reflectance, as also surveyed by Guarnera et al. [91]. The theme recurs: optimization is slow but matches the target well, while feed-forward networks are fast, but often do not reproduce the target.

Specifically, we study estimating stationary svBRDFs from flash images, pioneered by Aittala et al. [3], denoted SVBRDFSTAT in Table 3.2. We look at a design using a noise-conditioned encoder-decoder, as demonstrated in Henzler et al. [108]. We show re-lit results, parameter maps and all network details and training routines in Appendix Sec. A.2.3.

Non-stationary svBRDFs. Besides stationary svBRDFs, we look into estimating non-stationary ones (SVBRDFNONSTAT), also from flash images. This task was explored by Deschaintre et al. [50] as well as Guo et al. [95] and Gao et al. [78] before.

They all combine learning with fine-tuning in different ways. While Deschaintre et al. [52] use additional information and upsampling, Gao et al. [78] and Guo et al. [95] optimize first in a latent space, and later in the pixel space given only the target flash image.

We adapt the architecture from Deschaintre et al. [50], an encoder-decoder with a re-rendering loss, trained supervised under L_1 on synthetic flash images, SVBRDFNONSTAT from Table 3.2. For testing, both the reference as well as the inferred results are rendered from a set of novel view and light directions and compared.

Illumination. While the previous applications have looked into different forms of reflectance, another important application for visual appearance is estimating ILLUMINATION. To study the relation to meta-learning, we consider the task of representing natural spherical illumination itself as a NN. In particular, we consider an encoder-decoder that takes as input a diffuse shaded low-dynamic range (LDR) image of a sphere and outputs the high-dynamic range (HDR) environment map. Training data is rendered using the Laval HDR environment map dataset [79] to illuminate spheres of random materials. For evaluation, we render a second scene under the reference- as well as the inferred illumination and compare both results. Details are found in Appendix Sec. A.2.5.

Light transport. The ultimate explanation for visual appearance is the light transport in a scene itself [304]. To this day, robust handling of all forms of light transport remains a challenge [146]. One important building block that recently received a lot of attention is the *guidance* of paths [156; 309; 199; 200; 109; 335; 338], where previous paths build a model (parametric or NN-based) that is then used to steer path generation towards relevant paths that reduce variance more effectively. As this strategy involves optimization, it can also be meta-learned, so as to transfering understanding of light transport across scenes.

To do so, we study the architecture of Zheng and Zwicker [335], which relies on a normalizing flow [258] to learn a map from the unit hypercube to PSS, such that path density matches the one of the target scene. In other words, the normalizing

Table 3.2: Overview of all appearance reproduction applications we consider. Below, "ED" denotes encoder-decoder, "NF" is normalizing flow, "PN" is a Point-Net.

Application	Input I	Output L_θ	Domain $\mathbf{x}_i$	Architecture θ	Metric	Source
TEXTURE	RGB image	Infinite RGB map	2D Pos	ED CNN+Noise	VGG Gram L_1	Ulyanov et al. [302]
BRDF	Angle pair	RGB reflectance	4D Dir	CNN+MLP	L_1	Sztrajman et al. [293]
SVBRDFSTAT	Flash image	Infinite BRDF map	2D Pos, 4D Dir	ED CNN+Noise	VGG Gram L_1	Henzler et al. [108]
SVBRDFNONSTAT	Flash image	Finite BRDF map	2D Pos, 4D Dir	ED CNN	Re-render L_1	Deschaintre et al. [50]
ILLUMINATION	RGBN image	RGB HDR Envmap	2D Dir	ED CNN	Re-render L_1	Georgoulis et al. [83]
TRANSPORT	PSS sample	Light path + prob.	n-D Pos/Dir	PN MLP+NF	NLL	Zheng and Zwicker [335]

flow learns a scene-dependent PSS warp that is then applied to random PSS samples. Once trained, the model can be used to generate new paths as well as their probability. For details on architecture and training, please see Appendix Sec. A.2.6.

While previously these methods were applied to a single scene, we consider an entire set of scenes. To study this, we use a Cornell-box like configuration that is populated by a random number of between 1 and 5 spheres of random diffuse, glass or metal materials in random positions, a mirror that is randomly placed at a sidewall, and an area light positioned randomly on the ceiling. To quantify success, we measure the negative log-likelihood (NLL) across the test-set, as is done in [335], as using image-based metrics to quantify training error would require rendering the entire test-set per training epoch and method, which is computationally intractable. If the model matches the scene-dependent radiance distribution well, i.e., it correctly adapts to the light transport within the scene, the resulting NLL will be low. We report image-space metrics with the trained models for an equal number of rays in Tab. 3.4.

3.4.2 Methodology

Protocol We compare our method against the approaches listed in Sec. 3.3. The protocol is as follows: Let $\mathcal{I}$ denote the entire training data set, e.g., all BRDF samples in the MERL database. For `Overfit`, we sample a single input I from $\mathcal{I}$ and train a network on I, and only I, until convergence. `General` denotes a network that has been trained on all elements in $\mathcal{I}$ and then is conditioned on the particular I we want inference for. `Finetune` applies a pre-defined number of n fine-tuning steps to the output of `General` to further improve the result for a particular I. Finally, our proposed `Meta`-learning is trained on all tasks in $\mathcal{I}$, but

under the constraint of being given only a fixed budget of n_l gradient descent steps, with $n_l \ll n$. At inference time, a model conditioned on a particular I can then quickly be instantiated by updating the initial meta-model parameters n_l times.

Timing All experiments report inference time, i.e., the time elapsed between first presenting an input to the network and receiving its final output. We refrain from reporting render- or path-tracing times, as these do not change across methods. Please note that for **Meta**, inference time is different from (meta-) training time: meta-inference is quick, as we only need to perform a small number of gradient steps and do not need to calculate costly higher-order derivatives. We report this figure, as this is what a user would experience when presenting new, unseen inputs to one of our meta-trained applications. During meta-training, however, the opposite is true: we often need to backpropagate though the gradient operator itself, and do so for many examples. This leads to long meta-training times: **Meta** trains roughly twice as long as **General**. For a more detailed listing of training times, we refer to Appendix Tab. A.1. Note that for inference, the speed and memory consumption of the deployed NN is unaffected by meta-learning.

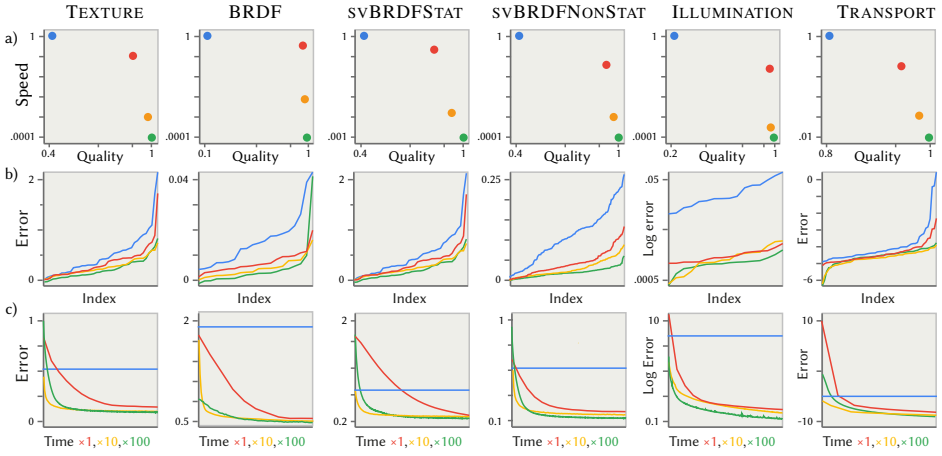
Metrics We evaluate each method on unseen input from the test set, with the particular evaluation metric depending on the application (column "Metric" in Table 3.2). In particular, meta-learning does not "cheat" by disclosing any test data during training; the split is the same as in conventional training. This means that **Meta** is presented with entirely new tasks (e.g., a completely unseen BRDF) instead of just withheld samples from a previously processed task (Appendix Fig. A.1visualizes this).

3.4.3 Results

We summarize quantitative results in Tab. 3.3. We consistently show lower compute time than **Finetune** and **Overfit** at only slightly reduced quality. The speed-quality plots (Tab. 3.3, a) show that our method is not just a compromise between the speed of **General** and the quality of **Overfit**, but instead is located much closer to the ideal range (top right corner) than all other methods. We will now discuss each application's results in turn.

Textures. Tab. 3.3 b) shows distribution of error across the texture task for all

Table 3.3: Quantitative results for all methods on all applications. The first row of plots shows the quality-speed continuum spanned by the four methods, with the ideal range (fast inference *and* high quality) being in the top right corner. The second row shows test-set error, individually sorted for each method. The third row shows convergence plots at inference time. Note that very different time-scales are plotted on the same horizontal range, so comparison can only be made in shape, not between fixed values at any point in time.



	TEXTURE		BRDF		SVBRDFSTAT		SVBRDFNONSTAT		ILLUMINATION		TRANSPORT	
	Error	Time	Error	Time	Error	Time	Error	Time	Error	Time	Error	Time
General	0.522	0.022	1.892	0.005	0.436	0.201	0.540	0.040	6.536	0.002	-3.457	0.040
Overfit	0.183	212.421	0.631	141.201	0.229	516.515	0.099	182.292	0.168	133.583	-4.120	362.210
Finetune	0.201	21.210	0.654	9.972	0.256	103.675	0.141	57.221	0.313	26.812	-4.030	42.070
Meta	0.252	0.619	0.720	0.031	0.311	1.484	0.197	0.474	0.352	0.384	-3.970	0.486

exemplars. We see that while both General and Meta struggle more with some specific (not necessarily the same) problem cases, the result is not dominated by outliers. The progress of training is seen in Tab. 3.3, c) where Meta achieves a quality more similar to Finetune than to General, but in a fraction of the time.

Fig. 3.4 shows qualitative results that further confirm these quantitative findings. General often projects the unseen textures into the latent space only approximately, which results in subtle but noticeable differences in features, color and scale. Finetune and Overfit both almost perfectly replicate the original, as both methods conduct a complete optimization run on the current sample. Our Meta-method faithfully replicates all textures with only minor differences in style. For more results, please see Appendix A.

BRDFs. Our Meta-method achieves high fidelity reproduction results across all BRDFs in the MERL-database, as quantified in Tab. 3.3. When rendered under Paul Debevec's St. Peter's Basilica illumination, Meta achieves structural similarity

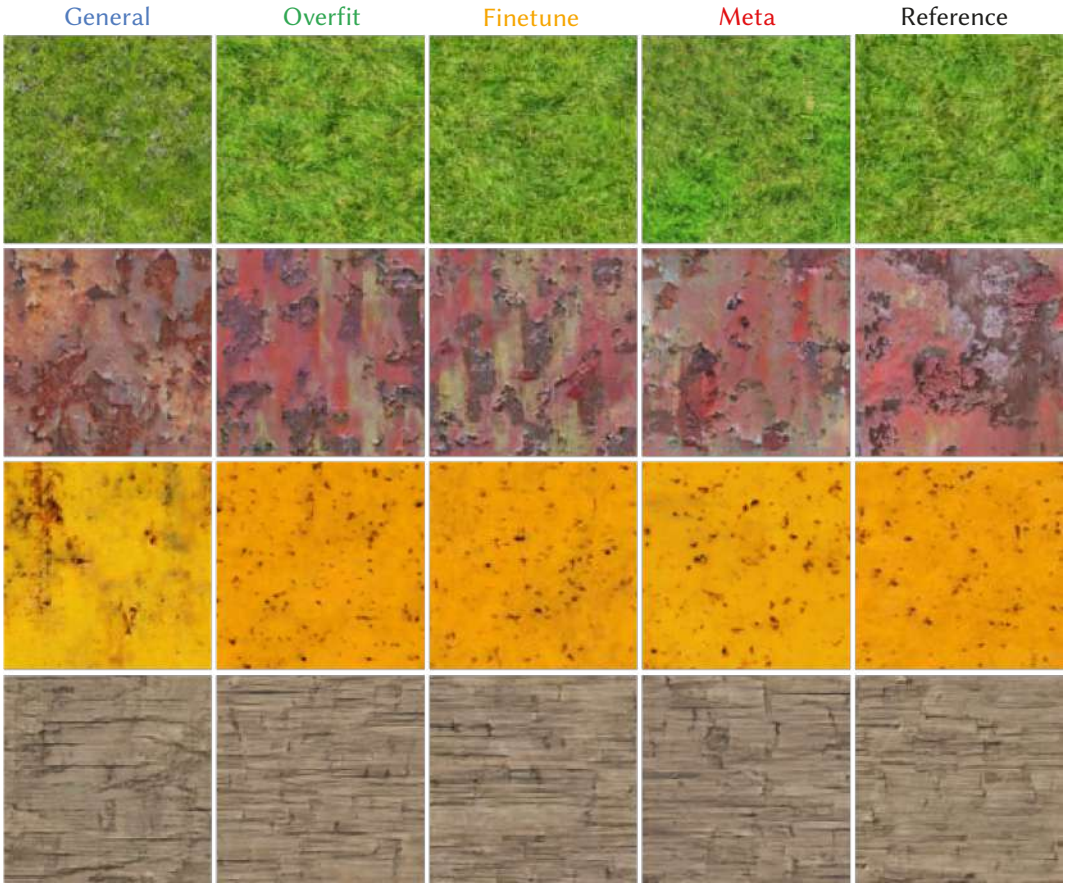


Figure 3.4: Results across the test set for the TEXTURE application. Every result is conditioned on a random process, so not meant to be compared pixel-by-pixel. We report inference time to the respective result.

(SSIM) values of ≥ 0.95 on 99% of all materials. For a visual comparison of the reproduction results of the different approaches, see Fig. 3.5: Our method picks up fine nuances in the BRDF correctly, and we consistently show large improvements over `General`. In some cases, `Meta` even outperform methods `Finetune` and `Overfit`, which both have a time budget several orders of magnitudes larger than our method.

Stationary svBRDFs. We detail quantitative results in Tab. 3.3 and show qualitative results in Fig. 3.6. This application again confirms our previous findings: `General` is fast, but fails to match the input accurately. This becomes evident in Fig. 3.6, where `General` broadly matches the target, but is missing fine details and has slightly tinted colors (top and bottom row) or washed-out highlights (middle rows). Both `Finetune` and `Overfit` match the target well and pick up subtle details

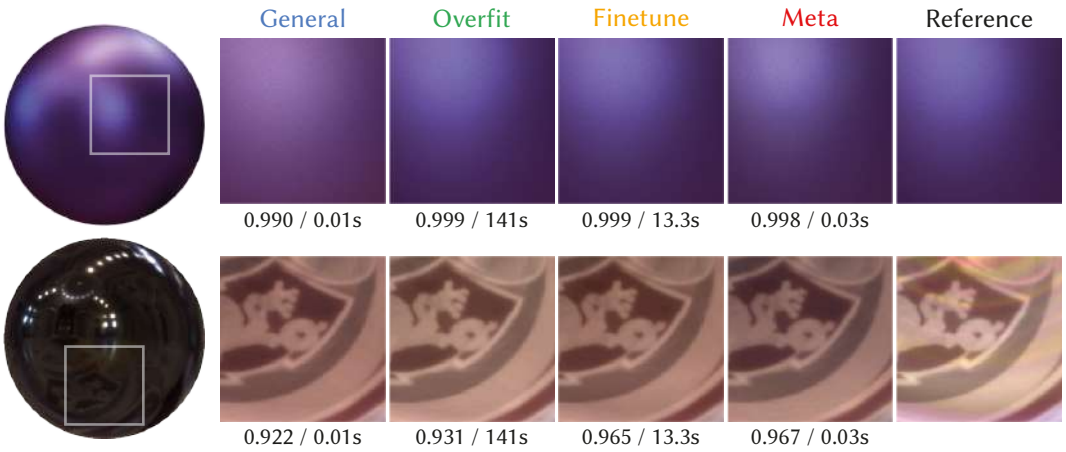


Figure 3.5: Rendered results for unseen BRDFs from the test-set, trained with the different methods. The insets quantify SSIM and inference time.

such as the wood grain and shading cues correctly, but take long to converge.

Our **Meta**-method achieves similar visual quality in a fraction of their runtime and manages to produce correct svBRDF maps in just over a second. We believe that a reason for this is the combination of strong priors built during meta-training (for an example, we refer to Fig. 3.11, left column) and learning how to adapt them optimally. While the quality-speed improvement is still significant, the gain from using **Meta** here is comparatively small, as seen from the position of the red-dot in the quality-speed continuum.

Non-stationary svBDRFs. Qualitative results for non-stationary svBRDFs are shown in Fig. 3.7. We see that **General** is producing highlights not present in optimization-based training schemes, including ours. Out of those optimization-based methods, ours is several orders of magnitude faster, as seen in the numbers and the speed-quality plot in Tab. 3.3, column "svBRDFNonStat". Both **Finetune** and **Overfit** perform well, although the visual comparison in Fig. 3.7 shows **Finetune** to perform slightly better. We presume that this is due to the fact that **Finetune** benefits from the priors developed by **General** (recall, fine-tuning starts at the output of the general model), whereas **Overfit** starts the training from scratch. In heavily ill-posed tasks like svBRDF estimation, the importance of solid priors has been shown to be of great importance for the optimization [95; 78]. This is also part of the explanation for **Meta**'s success on this task, as it can build priors

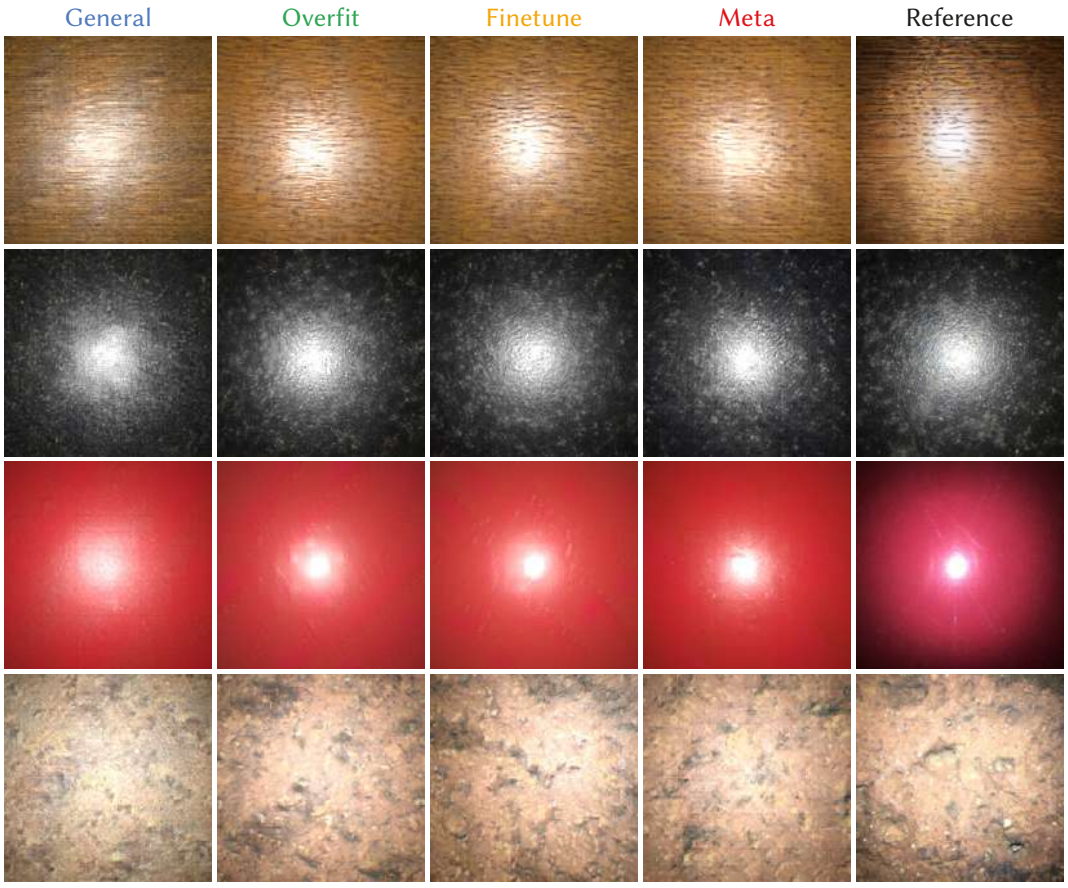


Figure 3.6: Results across the SVBRDFSTAT test-set. Note that every result is a realization of a random process, so not meant to be compared pixel-by-pixel. For re-lit renderings and shading maps, see Appendix A.

over the dataset in the outer loop *and* perform a quick overfit-optimization in the inner loop without diverging too far.

Illumination. We present results for our ILLUMINATION task in Fig. 3.8, where we render the inferred envmaps on a scene with a specular and diffuse object. We compare all methods against a reference image of that same scene rendered under the groundtruth illumination. We note how `General` is able to place sharp shadows (indicating it handles HDR well) but does not manage to exactly match the intensity. Similarly, reflections look plausible, but do not match the reference. Optimization-based methods meet this requirement, but only our meta-trained approach is orders of magnitude faster and achieves comparable quality. This is confirmed by the quality-speed plots in Tab. 3.3, where `Meta` is far-right, indicating that quality is very close to the full optimization-based methods. Tab. 3.3, c shows, that `Meta`

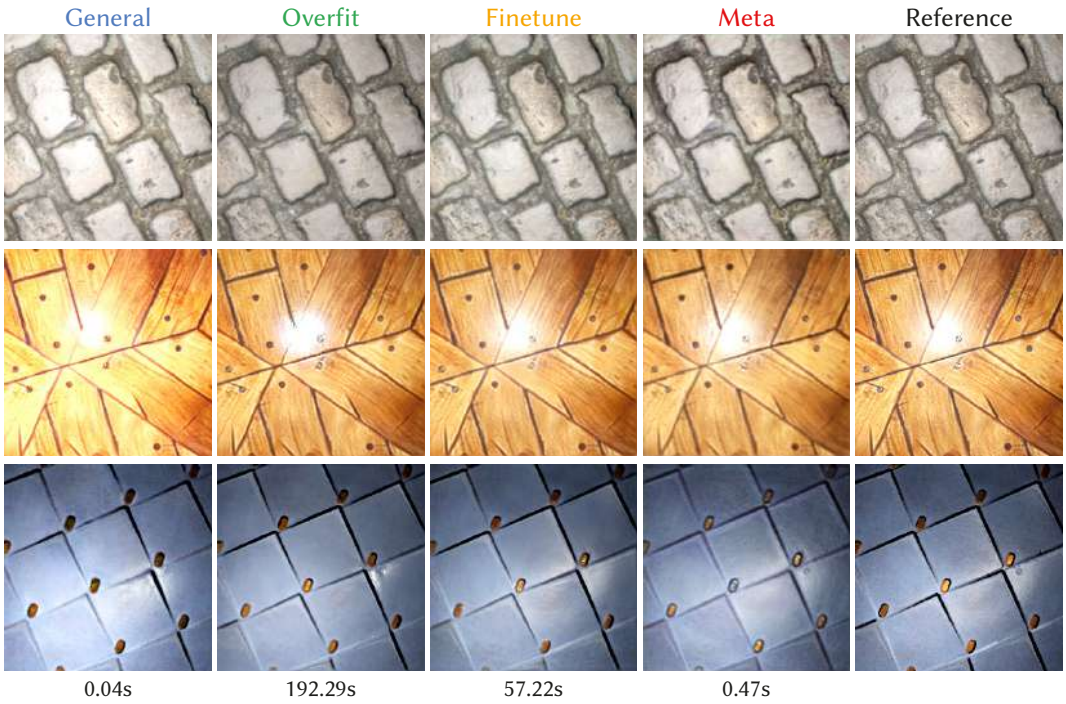


Figure 3.7: Relighting results across the test-set for the SVBRDFNONSTAT task. We render the resulting parameter maps under a different view- and light angle. For more results and the parameter maps, see Appendix A.

converges even faster than for other tasks (the red curve is more concave). From the distribution in Tab. 3.3, b we see that the classic optimization-based methods have no problematic outliers, a property retained by **Meta**.

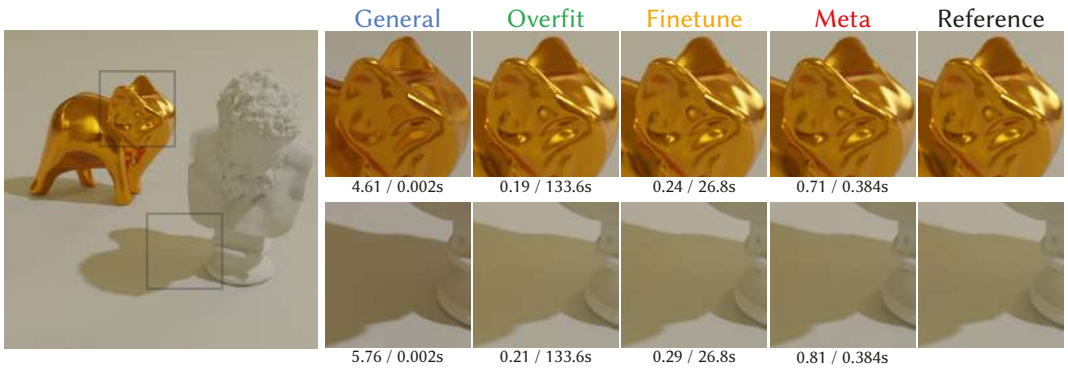


Figure 3.8: Results for an unseen instance from the ILLUMINATION test-set, inferred from a single RGB image (not shown) and used to render a novel scene (left). Quality of illumination is most revealed in reflections (top) and cast shadow (bottom). The sharpness and shape of the shadows is very indicative of the high dynamic range of the regressed envmap. We report MAE $\times 10^2$ and inference time. For direct visualization of the envmaps, see Appendix A.

Light transport. We show the outcome for TRANSPORT in Fig. 3.9. Recall that TRANSPORT uses a resampling of a scene's radiance distribution to learn a model that is used for importance sampling that particular scene. To compare the effectiveness of each approach, we render a novel scene (unobserved during training for `General` and during meta-training for `Meta`) using samples generated by the respective importance-sampling model. We further include an additional baseline, **`Regular`**, for this application. **`Regular`** uses importance-sampling for the geometric term and randomly samples outgoing paths from the hemisphere oriented around the normal. All subfigures are rendered with the same number of samples (4096) and very similar compute time, as querying the importance model can be parallelized on the GPU and hence is fast compared to the tracing of rays (milliseconds vs. minutes). As elaborated earlier (see Sec. 3.4.1), we report the time between model instantiation and the final training step (i.e., when it is ready to produce light path samples) as this is the part that the choice of training scheme can influence, whereas the subsequent path-tracing time is approximately invariant[3] to the origin of the samples. For a comparison of ray-generation and -tracing times, see Appendix Tab. A.4.

Quantitative results are seen in Tab. 3.3, column TRANSPORT: `Meta` is again closest to the top-right corner, indicating that it combines the quality of `Overfit` and the speed of `General`. The qualitative outcome (Fig. 3.9) indicates that all methods successfully reduce variance w.r.t. the **`Regular`** baseline. Again, `General` performs slightly below the iterative optimization-based methods `Meta` is orders of magnitude faster. To quantify how well the instantiated models perform in image space, we rendered the entire test-set with light paths produced by the respective importance model and a fixed sampling budget of 1024 rays per pixel. We display common metrics calculated on these renderings in Tab. 3.4. The results confirm the qualitative inspection in Fig. 3.9 and show that `Meta` again is much closer to `Overfit` and `Finetune` than to the general or regular baseline. For details on the sampling and rendering operations, we refer to Appendix Sec. A.2.6.

[3]We write approximately as the PSS samples created by all *trained* methods are created in Python and must be passed to the C++ renderer, which incurs a time overhead that the **`Regular`** baseline does not suffer. However, this is in the order of milliseconds, and hence can be neglected.

Table 3.4: Mean absolute percentage error and structural dissimilarity (DSSIM) across the TRANSPORT test-set. Lower is better for both metrics.

	Regular	General	Overfit	Finetune	Meta
MAPE	0.516	0.471	0.405	0.409	0.422
DSSIM	0.546	0.479	0.418	0.425	0.430

Please note that we explicitly refrain from discussing which method of importance sampling or path guiding is most appropriate for practical applications and re-iterate that we compare ways of *training* approaches instead of directly comparing the performance of different approaches. We here introduce meta-learning to the importance-sampling and rendering community as a first proof of concept and show that a meta-importance-sampler can generalize across a distribution of Cornell box-like scenes with practical benefits. To our knowledge, this is the first application of meta-learning in rendering, and the first presentation of meta-learned normalizing flows.

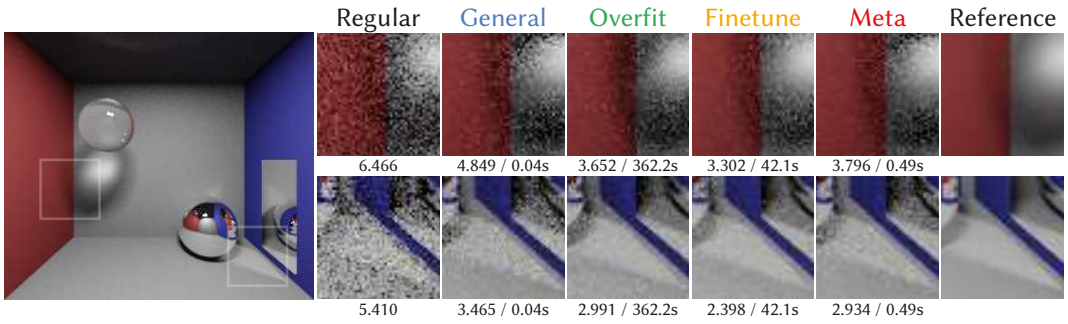


Figure 3.9: Results for an unseen test-scene for the TRANSPORT application. The left image is rendered with 240,000spp to guarantee a noise-free reference. The images to the right are produced by rendering the reference scene with PSS samples produced by our different approaches (equal number of samples, i.e., equal rendering time). We report symmetric mean absolute percentage error (SMAPE) and the respective model inference time.

3.5 Analysis

As the previous section has shown, **Meta** achieves similar quality than optimization-based approaches that take orders of magnitude more training time. To analyze the inner workings of Metappearance, we will next discuss a range of further properties that can be deduced from our experiments: We will ablate our learned components

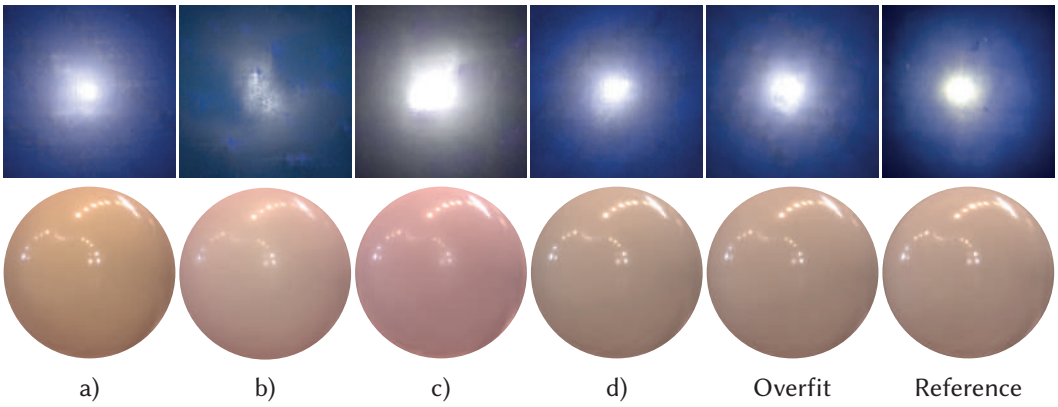


Figure 3.10: We compare the influence of a learned learning rate (column b) and initialization (column c). See the main text for details.

(Sec. 3.5.1), look at convergence of the inner and outer loop (Sec. 3.5.2), explain how Metappearance can be interpreted as model compression and finally discuss how `Meta` can make do with much fewer input observations (Sec. 3.5.3).

3.5.1 Ablations

We meta-learn two hyper-parameters: step size and inititalization, but which of them actually contributes to the success? Fig. 3.10 looks into this question. In column a), we display the output of `General`. In column b), we take this as starting point and use our learned step size to perform $n_\text{l} = 20$ "smart" gradient steps towards the reference. Evidently, this leads to inferior results, as the general model has been trained with a fixed, global learning rate, and hence does not know how to account for a per-parameter learning rate. In column c), we use our meta-learned initialization to perform $n_\text{l} = 20$ steps of conventional Adam optimization (learning rate multiplied by 10 for faster convergence) towards the reference. This again leads to poor results, as our learned initialization normally is adapted through large, non-uniform gradient steps. During meta-training, the initialization was hence moved to a region of the objective space that is approximately equally well-suited for all tasks, but not necessarily easy to navigate with uniform gradient steps. Column d) finally shows the output of our `Meta`-method, where learned initialization and step size are used in combination. Evidently, this outperforms all alternative configurations.

This decay in reproduction quality shows that meta-learning really combines the best of both worlds: By optimizing for optimization directly, the outer optimizer

can discover gradient paths that lead to local minima by not only moving the network weights (as would `General`), but also the step-size with which these weights are updated for a certain number of iterations (as in `Finetune`).

It is tempting to argument that optimizing over optimization itself, e.g., learning the step size, automates time-consuming hyperparameter searches. While this is true to a certain extent, one still must choose the *meta*-hyperparameters, e.g., outer loop learning rate, etc. All our experiments use very similar hyper-parameters (see Appendix Sec. A.1) that were not particularly tuned, but this might be different in different applications or designs (cf. [8]).

3.5.2 Convergence

In Fig. 3.11, we show the convergence of our meta-learned initialization (leftmost column) towards different targets. All intermediate outputs show realistic appearance, and even the meta-initialization could pass as a problem instance (e.g., a texture) on its own. Throughout optimization, our method does not introduce unwanted artifacts, even for the ambiguous single-image svBRDF estimation task.

For maximal quality, we can fine-tune a converged `Meta` model for an additional number of training steps. We explicitly refrained from doing so in our main experiments, as this defies the purpose of Metappearance (achieving high quality *without* fine-tuning). However, `Meta` will converge to a loss difference of less than 1 % relative to `Overfit` in only 65 additional training steps (less than 5 seconds on every application). Related, `General` will not improve from further general training and has been trained to saturation already.

However, it is not our objective to claim superior quality for infinite time and compute resources. Instead, let us consider what often is the case in applications that involve user interaction: operating under the constraint of a finite time budget. Imagine, for instance, an architect wanting to quickly add a real-world svBRDF to his 3D model; imagine an app instantaneously adding a customized deep visual appearance model to a Tik-Tok video. The results of these applications must not only be highly accurate, but also available within split seconds to keep the user engaged, and no other method comes close to ours in this quality-speed trade-off.

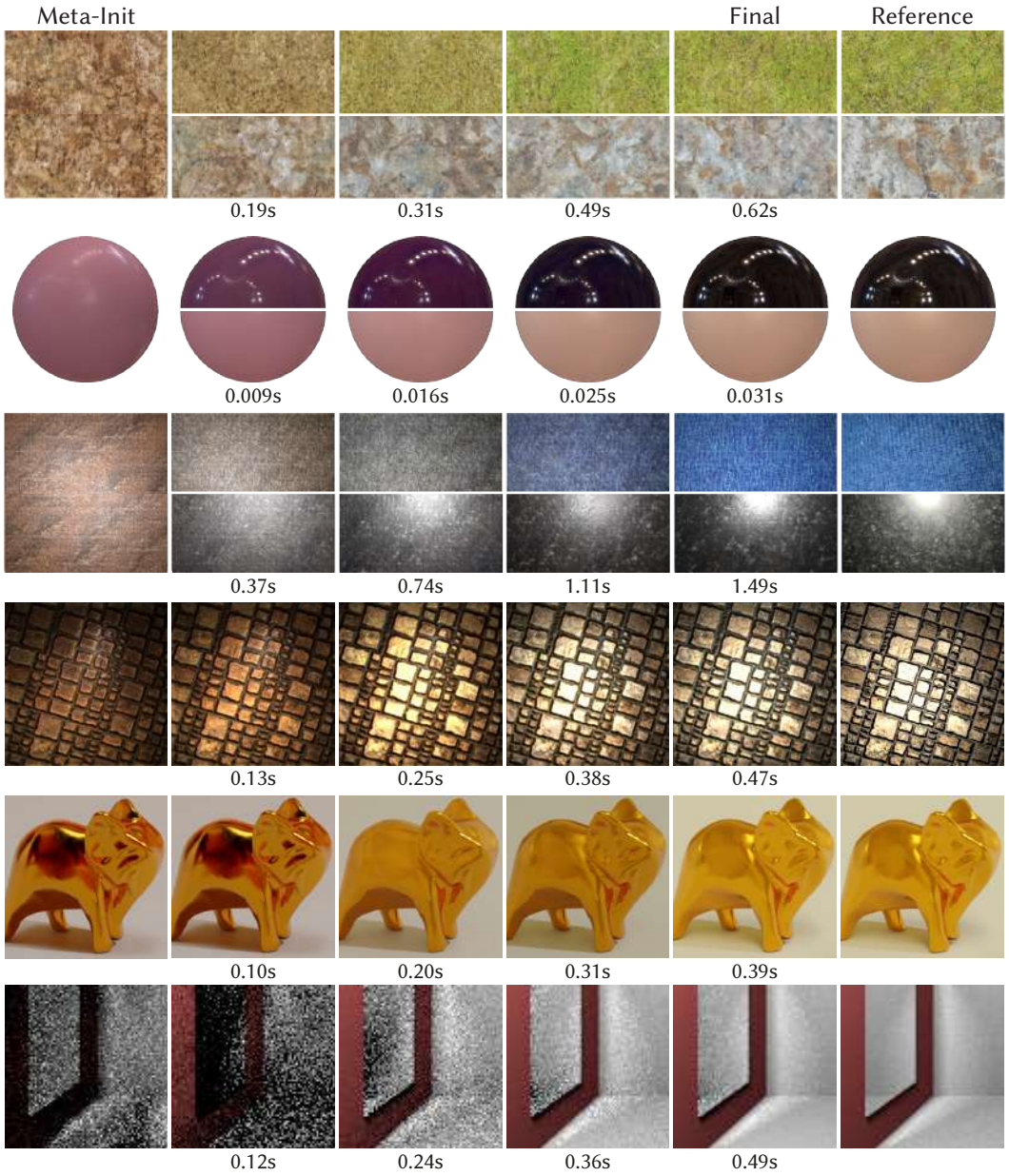


Figure 3.11: Convergence on unseen test-tasks from the meta-init (left) to different targets (right) for our different applications. We report wall-clock inference time and results after approx. 25%, 50% and 75% (columns 2 - 5) of the inner-loop steps. Note that the init itself is a plausible instance and enables the optimization to "branch" to specific, very different goals. The result for TRANSPORT is an equal-spp rendering with the inferred importance model. Note how the noise clears with more meta-iterations although the sample count stays the same.

We hence claim highest quality under the constraint of limited time, and additionally investigate an equal-time comparison between **Finetune** and **Meta**, where both approaches are allowed to perform the same number of gradient steps n_1 that would

Table 3.5: Average error across the respective application's test-set for our QuickFinetune-experiment. For the metrics reported, please see Table 3.2. For convenience, we repeat results for methods `General` and `Meta` from Tab. 3.3.

	n_1	General	QuickFT	Meta
TEXTURE	15	0.522	0.285	0.252
BRDF	10	1.892	1.346	0.720
SVBRDFSTAT	20	0.436	0.351	0.311
SVBRDFNONSTAT	15	0.540	0.289	0.197
ILLUMINATION	15	6.536	5.240	0.352
TRANSPORT	8	-3.457	-3.551	-3.970

normally be used during meta-inference. This effectively is a very quick fine-tuning session, which is why we refer to it as "QuickFinetune", or **QuickFT**[4]. **QuickFT** uses Adam, for which we again increase the learning rate for faster convergence, as we have done for almost all applications in our experiments (the **QuickFT** config used here is the same as in Appendix Tab. A.1), while `Meta` uses both its learned init and the learned stepsize.

As Tab. 3.5 shows, **QuickFT** already offers great gains over the general method on some applications. However, it is outperformed by `Meta` on all applications. For example-specific visualizations of this experiment, including equal-time comparisons to `Overfit`, see Appendix Fig. A.8. This confirms that meta-learning is more than mere general inference followed by fine-tuning and that optimizing over the optimization procedure itself (recall, we learn how to overfit a sample efficiently) really finds a non-trivial optimizer.

3.5.3 Compression and Efficiency

Storage. Our `Meta`-method can further be interpreted as a compression scheme. Consider a rendering or 3D-modeling application, e.g., Blender, that loads a pre-trained model's weights to create new, diverse textures for surfaces. With methods `Overfit` and `Finetune`, such an application would have to store an entire set of weights per texture to be generated (note that, in such a scenario, storing the

[4]To be able to fine-tune, we need to run `General` once. We do not deduct this runtime from **QuickFT**'s time budget, as this makes the comparison stronger and also usually is a rather fast operation.

fine-tuned decoder of method **Finetune** is sufficient), and then is restricted to synthesizing the pre-learned texture exemplars. One could alternatively store the heavy-weight general model, but then would either have to forego accurate high-quality synthesis or fine-tune the result, which is unsatisfying and time-consuming, respectively. With our proposed **Meta**-method, it is sufficient to store two sets of weights only (the model's weights, and the per-parameter learning rate) to achieve high-quality, diverse texture synthesis in interactive runtime, i.e., in less than a second.

Similar arguments can be made for all the applications we have presented in this work. Let w denote the number of disk space required to store the model's weights in methods **Overfit** and **Finetune** (we omit **General** from this comparison as we are concerned with high-quality results only). The total amount of storage required for the efficient and exact synthesis of m textures then equals wm, i.e., one set of weights per exemplar. Our **Meta**-method achieves similar-quality results with *constant* storage requirement $2w$ (weights and per-parameter learning rate), and is not limited to pre-trained or fine-tuned texture exemplars but instead can quickly infer new, unseen exemplars with high fidelity. The compression factor our method achieves hence is $2m^{-1}$, which, in the case of our exemplar texture application with 500 exemplars, equals $1:250$.

We would furthermore like to point out that compression also is an inherent property of using neural networks on certain problem instances. An NBRDF (a BRDF encoded in a network [293]), for instance, has a significantly smaller memory footprint than regular BRDFs (28 kB vs. 34.2 MB in the case of our **Meta**-NBRDF), so we additionally compress the original BRDF with a factor of approx. $1:2000$. However, we do not claim credit for this as a property of our method, but rather of the base approaches we meta-train. In fact, quite the contrary is true – our method requires double the storage of a single network (model weights and per-parameter learning rate). However, we believe **Meta**'s ability to quickly converge to unseen tasks and the resulting, aforementioned compression arguments to outweigh this moderate increase.

Sample Efficiency. For methods that consume single data points, i.e., methods that use an MLP, there is a further compression argument that can be made. To do so, we would like to draw attention to the way **Meta** is trained.

Recall that in meta-learning, the inner loop performs a fixed (small) number of gradient descent steps towards the reference. Naturally, as with most recent optimization algorithms, **Meta** uses *stochastic* gradient descent, i.e., the inner-loop gradients are not calculated per-sample or across all samples, but rather over a randomly selected subset of all available samples. Evidently, as **Meta** only performs n_l inner loop steps, only n_l batches of size b are sampled.

Naturally, this process repeats many times during meta-training and hence eventually samples all data in a task (e.g., all samples in a BRDF). However, this is not the case during meta-inference: Recall that for meta-inference, we merely run n_l gradient steps on a new, unseen task. Evidently, this leads to **Meta** seeing only $n_\text{l} \times b$ samples of a task, while all its competitors have access to *all* the samples in a task – in the cases of **Overfit** and **Finetune**, even repeatedly. Nonetheless, **Meta** delivers quality that is very close to its competitors that have seen all data.

This is no surprise: The ability to make do with scarce data is a core property of meta-learning and has been amply explored in previous works [4; 63]. In Metappearance, this property could be useful in a number of ways: Imagine, for instance, a client-server architecture, where the server stores large amounts of data (e.g., a large set of scene-dependent TRANSPORT radiance distributions), and the client stores the neural representation that will be trained on samples of this data. In order to train a model on a specific radiance distribution, the server would have to transfer *all* of its samples to the client (let us ignore the considerable time cost of training or fine-tuning a network on this data for a second). However, following the above argument, we only need to transfer $n_\text{l} \times b$ samples to infer a converged instance of **Meta**, for which the inference time can *really* be neglected.

In summary, this higher efficiency leads to bandwidth savings of approx. 72.2% for the TRANSPORT application (a full dataset is 288,000 samples, **Meta** consumes only $8 \times 10{,}000 = 80{,}000$ samples). For the case of BRDF encoding, the resulting

reduction in needed data transmission is even more drastic: **Meta** takes n_l =10 inner loop steps with a batchsize of $b = 512$ and hence consumes 5,120 samples, whereas a full MERL BRDF, as is needed by all other methods, consists of $180 \times 90 \times 90 \approx 1.46 \times 10^6$ samples. **Meta** hence achieves a bandwidth saving of 99.6%.

3.6 Conclusion

We have used meta-learning for efficient and accurate appearance-reproduction on a variety of increasingly complex applications. Our model, Metappearance, provides users with results that qualitatively compare well to other training schemes which take orders of magnitude more training iterations or data. We have shown that Metappearance generalizes not only across problem instances of a similar nature, e.g., our variety of Cornell-box scenes, but can also be applied across applications. In terms of implementation effort, the additional code relative to a solution that already uses an existing optimization is small. In fact, as we have shown in Sec. 3.3.5, re-phrasing the loss function is sufficient.

The main point of our experimentation is that while we cannot yet have both perfect speed and perfect quality, we, in several cases, improve substantially over a mere trade-off between the two, as seen from the red dot in Tab. 3.3, a), which has moved much closer to the ideal top-right spot, where visual appearance reproduction aims to be. Directions for future research could include the application of Metappearance to even more complex light-transport algorithms or its extension to meta-learning the objective function or the sampling pattern itself, which would enable even higher accuracy for visual appearance reproduction.

Chapter 4

Plateau-reduced Differentiable Path Tracing

Abstract

Current differentiable renderers provide light transport gradients with respect to arbitrary scene parameters. However, the mere existence of these gradients does not guarantee useful update steps in an optimization. Instead, inverse rendering might not converge due to inherent plateaus, i.e., regions of zero gradient, in the objective function. We propose to alleviate this by convolving the high-dimensional rendering function, that maps scene parameters to images, with an additional kernel that blurs the parameter space. We describe two Monte Carlo estimators to compute plateau-free gradients efficiently, i.e., with low variance, and show that these translate into net-gains in optimization error and runtime performance. Our approach is a straightforward extension to both black-box and differentiable renderers and enables optimization of problems with intricate light transport, such as caustics or global illumination, that existing differentiable renderers do not converge on. The code is available at `https://github.com/mfischer-ucl/prdpt`.

4.1 Introduction

Regressing scene parameters like object position, materials or lighting from 2D observations is a task of significant importance in graphics and vision, but also a hard, ill-posed problem. When all rendering steps are differentiable, we can derive

Figure 4.1: Optimization results with a differentiable path tracer (we use Mitsuba 3 [219]) and our proposed method. The task is to rotate the coffee cup around its z-axis, so that the handle moves right. Due to a plateau in the objective function (when the handle is occluded by the cup), regular methods do not converge.

gradients of the final image w.r.t. the scene parameters. However, differentiating through the discontinuous rendering operator is not straightforward due to, e.g., occlusion. The two main approaches to (differentiable) rendering are path tracing and rasterization.

Physically-based path-tracing solves the rendering equation by computing a MC estimate for each pixel. Unfortunately, MC is only compatible with modern AD frameworks for the case of continuous integrands, e.g., color, but not for spatial derivatives, i.e., gradients w.r.t. an object's position. To alleviate this, Li et al. [169] present re-sampling of silhouette edges and Loubet et al. [183] propose re-parametrizing the integrand, enabling the optimization of primitive- or light positions. For rasterization, differentiability is achieved by replacing discontinuous edge- and z-tests with hand-crafted derivatives [182; 259; 179; 161]. The problem here is that rasterization, by design, does not capture complex light transport effects, e.g., global illumination, scattering or caustics.

Importantly, the mere existence of gradients is no guarantee that descending along them will make an optimization converge [192]. There are surprisingly many cases where they *do not* lead to a successful optimization, due to a plateau in the objective function. An example is finding the orientation of the mug in Fig. 4.1: As soon as the handle disappears behind the cup, no infinitesimally small rotation change will result in a reduced loss. We have hence reached a plateau in the objective function, i.e., a region of zero gradients. We propose a method to remove these plateaus while still having complete, complex light transport.

We take inspiration from differentiable rasterization literature [179; 259; 237; 161], where smoothing techniques are used to model the influence of faraway triangles to the pixel at hand. For rasterization, this simple change has two effects: First, it makes the edge- and z-tests continuous and hence differentiable, and second, in passing (and, to our knowledge, much less studied), it also removes plateaus. In this work, we hence aim to find a way to apply the same concept to complex light transport. Therefore, instead of making the somewhat arbitrary choice of a fixed smoothing function for edge- and depth-tests in differentiable rasterizers, we path-trace an alternative, smooth version of the entire rendering equation (RE), which we achieve by convolving the original RE with a smoothing kernel. This leads to our proposed method, a lightweight extension to (differentiable) path tracers that extends the infinitely-dimensional path integral to the product space of paths and scene parameters. The resulting double integral can still be MC-solved efficiently, in particular with variance reduction techniques we derive (importance sampling and antithetic sampling).

4.2 Background

4.2.1 Rendering equation

According to the rendering equation [140; 304], the pixel P is defined as

$$P(\theta) = \int_{\Omega} f(\mathbf{x}, \theta) \mathrm{d}\mathbf{x}, \tag{4.1}$$

an integral of the function $f(\mathbf{x}, \theta)$ which models the scene and depends on scene parameters $\theta \in \Theta$, over all light paths $\mathbf{x} \in \Omega$. In *inverse rendering*, we want to find the parameters θ^* that best explain the pixels P_i in the reference image with

$$\theta^* = \arg\min \theta \sum_i \mathcal{L}\left(P_i(\theta) - P_i(\theta_{\mathrm{ref}})\right), \tag{4.2}$$

where $\mathcal{L}$ is the objective function and $P_i(\theta_{\mathrm{ref}})$ are the target pixels created by the (unknown) parameters θ_{ref}. Consider the example setting displayed in Fig. 4.2, where we are asked to optimize the 2D position of a circle so that its rendering matches the

reference.

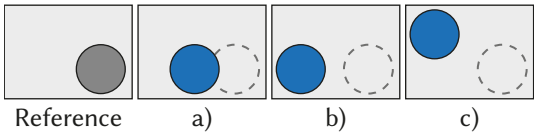


Figure 4.2: An example of a plateau in $\mathcal{L}$: starting the optimization of the circle's position at a) will converge, whereas b) and c) will not. In a)-c), we show the reference dotted for convenience.

Let θ_0 be the initial circle's position. In this simple example, the optimization will converge if, and only if the circle overlaps the reference, i.e., setting a) in Fig. 4.2. The reason for this is that the gradient then is non-zero (a small change in θ is directly related to a change in $\mathcal{L}$) and points towards the reference. However, if there is no overlap between the initial circle and the reference, as in Fig. 4.2 b), a gradient-based optimizer will not be able to recover the true position θ_{ref}. This is due to the fact that there exists a *plateau* in the objective function (for a rigorous definition, see Jin et al. [136]). To visualize this, consider a rendering where the circle is placed in the top left corner, as in Fig. 4.2 c). The scalar produced by the objective function is identical for both b) and c), as $\mathcal{L}$ measures the distance in *image space*. Therefore, the change in $\mathcal{L}$ is zero almost everywhere, leading to zero gradients and to the circle not moving towards the reference position. As we will see in Sec. 4.4, this is surprisingly common in real applications.

Table 4.1: Rendering taxonomoy. See Sec. 4.2.2 and Sec. 4.2.3.

	Rasterizer	Path Tracer	Ours
Differentiable	✓	✓	✓
Light Transport	×	✓	✓
Plateau-free	✓	×	✓

4.2.2 Path tracing

As there is no closed-form solution to Eq. 4.1, path tracing uses MC to estimate the integral by sampling the integrand at random paths $\mathbf{x}_i$:

$$\widehat{P} \approx \frac{1}{N} \sum_i f(\mathbf{x}_i, \theta) \tag{4.3}$$

Gradients: We are interested in the partial derivatives of P with respect to the

scene parameters θ, i.e.,

$$\frac{\partial P}{\partial \theta} = \frac{\partial}{\partial \theta} \int_\Omega f(\mathbf{x}, \theta) \mathrm{d}\mathbf{x} = \int_\Omega \frac{\partial}{\partial \theta} f(\mathbf{x}, \theta) \mathrm{d}\mathbf{x}. \tag{4.4}$$

In order to make Eq. 4.4 compatible with automatic differentiation, Li et al. [169] propose a re-sampling of silhouette edges and Loubet et al. [183] suggest a re-parametrization of the integrand. Both approaches allow to MC-estimate the gradient as

$$\widehat{\frac{\partial P}{\partial \theta}} \approx \frac{1}{N} \sum_i^N \frac{\partial}{\partial \theta} f(\mathbf{x}_i, \theta). \tag{4.5}$$

This is now standard practice in modern differentiable rendering packages [219; 169; 327; 330; 329], none of which attempt to actively resolve plateaus.

4.2.3 Rasterization

Rasterization solves a simplified version of the RE, where for every pixel, the light path length is limited to one. It is often used in practical applications due to its simplicity and efficiency, but lacks the ability to readily compute complex light transport phenomena. Instead, rasterization projects the primitives to screen space and then resolves occlusion. Both steps introduce jump discontinuities that, for differentiation, require special treatment.

Gradients: In differentiable rasterization, both these operations therefore are replaced with smooth functions. Loper and Black [182] approximate the spatial gradients during the backward pass by finite differences. Early, Rhodin et al. [259], often-used Liu et al. [179] and later Petersen et al. [236] replace the discontinuous functions by soft approximations, e.g., primitive edges are smoothened by the Sigmoid function. This results in a soft halfspace test that continuously changes w.r.t. the distance from the triangle edge and hence leads to a differentiable objective. Chen et al. [38] and Laine et al. [157] propose more efficient versions of this, while Xing et al. [322] use an optimal-transport based loss function to resolve the problem. However, most differentiable rasterizers make simplifying assumptions, e.g., constant colors, the absence of shadows or reflections, and no illumination interaction

between objects. Our formulation does not make such assumptions.

Plateaus: Choosing smoothing functions with infinite support (for instance, the Sigmoid), implicitly resolves the plateau problem as well. Our method (Sec. 4.3) draws inspiration from this concept of "differentiating through blurring".

Shortcomings: Consider again Fig. 4.2 a), where the circle continuously influences the rendered image, resulting in a correct optimization outcome. For rasterizers, it is easy to construct a case where this does not hold, by imagining the circle to be the shadow of a sphere that is not seen in the image itself. The smoothed triangles then do not influence the rendering (most differentiable rasterizers do not even implement a shadow test [169; 237]) and can therefore not be used for gradient computation. Analogue examples can be constructed for other forms of multi-bounce light transport.

4.2.4 Other renderers

Other ways to render that are neither path tracing or rasterization exist, such as differentiable volume rendering [301; 106; 194] or fitting NNs to achieve a differentiable renderer [206; 111; 268; 281]. Also very specific forms of light transport, such as shadows, were made differentiable [184]. Finally, some work focuses on differentiable rendering of special primitives, such as points [128; 325], spheres [178], signed distance fields [135; 307; 13] or combinations [40; 121]. While some of these methods also blur the rendering equation and hence reduce plateaus, they remain limited to simple light transport.

4.3 Plateau-free Gradients

Intuition: As differentiable rasterization (see Sec. 4.2.3) has established, the blurring of primitive edges is a viable means for differentiation. But what if there is no "primitive edge" in the first place, as we deal with general light paths instead of simple triangles that are rasterized onto an image? The edge of a shadow, for instance, is not optimizable itself, but the result of a combination of light position, reflection, occlusion, etc. Therefore, to achieve an effect similar to that of differentiable rasterizers, we would need a method that blurs the entire light path (instead of just

Figure 4.3: Optimizing the cup's rotation in the hard (left, blue) and smooth (right, orange) setting (note the blurred handle). The image-space loss landscape is displayed on the right: blurring resolves the plateau.

primitive edges) over the parameter space θ. If this method used a blur kernel with infinite support (e.g., a Gaussian distribution), the plateau in the objective would vanish, as a small parameter change in any direction would induce a change in the objective function.

Example: Let us consider Fig. 4.3, where we again want to optimize the cup's rotation around its z-axis to have the handle point to the right, a 1D problem. As we have seen previously, using an image-based objective function leads to a plateau when optimizing $\mathcal{L}$ in the "hard" setting, i.e., without blur (the blue line in the plot). Blurring the cup's rotation parameter, on the other hand, leads to θ continuously influencing the value of the objective and therefore resolves the plateau (orange line in the plot). Naturally, it is easy to descend along the gradient of the orange curve, while the gradient is zero on the plateau of the blue curve.

4.3.1 The Plateau-free Rendering Equation

Formulation: We realize our blurring operation as a convolution of the rendering equation (Eq. 4.1) with a blur kernel κ over the parameter space Θ:

$$\begin{aligned} Q(\theta) = \kappa \star P(\theta) &= \int_{\Theta} \kappa(\tau) \int_{\Omega} f(\mathbf{x}, \theta - \tau) \, \mathrm{d}\mathbf{x} \mathrm{d}\tau \\ &= \int_{\Theta \times \Omega} \kappa(\tau) f(\mathbf{x}, \theta - \tau) \, \mathrm{d}\mathbf{x} \mathrm{d}\tau \,. \end{aligned} \tag{4.6}$$

The kernel $\kappa(\tau)$ could be any symmetric monotonous decreasing function. For simplicity, we use a Gaussian here, but other kernels would be possible as well. The kernel acts as a weighting function that weights the contribution of parameters θ

that were offset by $\tau \in \Theta$. This means that, in addition to integrating all light paths $\mathbf{x}$ over Ω, we now also integrate over all parameter offsets τ in Θ. We do not convolve across the path space Ω but across the parameter space θ, e.g., the cup's rotation in Fig. 4.3.

Estimation: To estimate the (even higher-dimensional) integral in Eq. 4.6, we again make use of an MC estimator

$$\widehat{Q} \approx \frac{1}{N} \sum_{i}^{N} \kappa(\tau) f(\mathbf{x}_i, \theta - \tau_i)\,, \tag{4.7}$$

which is a practical approximation of Eq. 4.6 that can be solved with standard path tracing, independent of the dimensionality of the light transport or the number of optimization dimensions.

Gradient : Analogously to Eq. 4.5, we can estimate the gradient of Q through the gradient of its estimator

$$\widehat{\frac{\partial Q}{\partial \theta}} = \frac{\partial}{\partial \theta} \frac{1}{N} \sum_{i=1}^{N} \kappa(\tau_i) f(\mathbf{x}_i, \theta - \tau_i)\,. \tag{4.8}$$

Due to the linearity of differentiation and convolution, there are two ways of computing Eq. 4.8: one for having a differentiable renderer, and one for a renderer that is not differentiable. We discuss both options next.

Plateau-free gradients if P is differentiable: With access to a differentiable renderer (i.e., access to $\partial P / \partial \theta$), we can rewrite Eq. 4.8 as

$$\widehat{\frac{\partial Q}{\partial \theta}} = \frac{1}{N} \sum_{i=1}^{N} \kappa(\tau_i) \underbrace{\frac{\partial P}{\partial \theta}(\theta - \tau_i)}_{\text{Diff. Renderer}}\,. \tag{4.9}$$

Therefore, all that that needs to be done is to classically compute the gradients of a randomly perturbed rendering and weight them by the blur kernel.

Plateau-free gradients if P is not differentiable: In several situations, we might not have access to a differentiable renderer, or a non-differentiable renderer might have advantages, such as computational efficiency, rendering features or compatibility with other infrastructure. Our derivation also supports this case, as we

can rewrite Eq. 4.8 as

$$\widehat{\frac{\partial Q}{\partial \theta}} \approx \frac{1}{N} \sum_{i=1}^{N} \underbrace{\frac{\partial \kappa}{\partial \theta}(\tau_i)}_{\text{Diff. Kernel}} \underbrace{P(\theta - \tau_i)}_{\text{Renderer}}, \tag{4.10}$$

which equals sampling a non-differentiable renderer and weighting the result by the gradient of the blur kernel. This is possible due to the additional convolution we introduce: prior work [169; 183] must take special care to compute derivatives (Eq. 4.5), as in their case, optimizing θ might discontinuously change the pixel integral. We circumvent this problem through the convolution with κ, which ensures that, in expectation, θ *continuously* influences the pixel integral.

4.3.2 Variance Reduction

Drawing uniform samples from $\Theta \times \Omega$ will result in a sample distribution that is not proportional to the integrand and hence lead to high-variance gradient estimates and ultimately slow convergence for inverse rendering.

In our case, the integrand is the product of two functions (the kernel κ and the scene function f), which Veach [304] showed how to optimally sample for. As we generally consider the rendering operator a black box, we can only reduce variance by sampling according to the remaining function, the (differentiated) kernel κ (Fig. 4.4b).

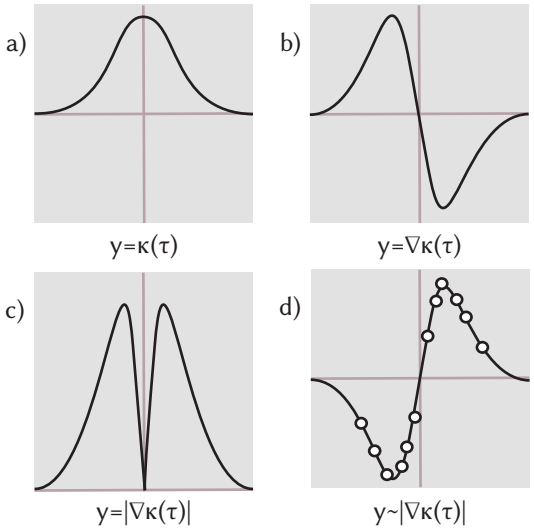


Figure 4.4: Our kernel κ (a), its gradient $\nabla\kappa$ (b), the positivized gradient (c) and samples drawn proportional thereto (d).

While importance-sampling for a Gaussian ($\tau_i \sim \kappa$, required to reduce variance of Eq. 4.9) is easily done, importance-sampling for the gradient of a Gaussian ($\tau_i \sim \partial\kappa/\partial\theta$, to be applied to Eq. 4.10) is more involved.

The gradient of our kernel κ is

$$\frac{\partial \kappa}{\partial \theta}(\tau) = \frac{-\tau}{\sigma^3\sqrt{2\pi}} \exp\left(\frac{-\tau^2}{2\sigma^2}\right), \tag{4.11}$$

which is negative for $\tau > 0$. We enable sampling proportional to its PDF (which is also partially negative) by "positivization" [224], and hence sample for

$$|\frac{\partial \kappa}{\partial \theta}(\tau)|$$

instead (Fig. 4.4c). We note that this function is separable at $\tau = 0$ and thus treat each halfspace separately in all dimensions of τ and σ. In order to sample we use the inverse CDF method. The CDF of Eq. 4.11 is

$$F(\tau) = 0.5\,\mathrm{sgn}(\tau) \exp\left(-\frac{\tau^2}{2\sigma^2}\right) + C,$$

where $C = 1$ on the positive halfspace and 0 otherwise (this originates from the fact that the CDF must be continuous, monotonically increasing and defined on $(0,1)$). Inverting the CDF leads to

$$F^{-1}(\xi) = \sqrt{-2\sigma^2 \log(\xi)},$$

into which we feed uniform random numbers $\xi \in (0,1)$ that generate samples proportional to $|\frac{\partial \kappa}{\partial \theta}(\tau)|$ (Fig. 4.4d).

Obtaining a zero-variance estimator for a positivized function requires sampling at at least two points: on the positivized and the regular part of the function [224]. We note that the function we sample for is point-symmetric around 0 in each dimension and hence use antithetic sampling [100], i.e., for each sample τ, we additionally generate its negated counterpart $-\tau$. Doing so results in a zero-variance estimator, as we can perfectly sample for both parts of the function. In additional experiments, we also found stratified sampling to be more brittle than antithetic sampling.

In previous inverse rendering work, antithetics were applied to BSDF gradients in classic rendering [11; 327] and to improve convergence on specular surfaces [330],

while we use them as a means of reducing gradient variance due to plateau-removal, which is not present in such approaches.

4.3.3 Adaptive bandwidth

Adjusting σ gives us control over how far from the current parameter θ our samples will be spaced out. A high σ may be useful in the early stages of the optimization, when there still is a considerable difference between θ and θ_{ref}, whereas we want a low σ towards the end of the optimization to zero-in on θ_{ref}. Throughout the optimization, we hence decay the initial σ_0 according to a linear schedule, i.e., $\sigma_{t+1} = \sigma_0 - t(\sigma_0 - \sigma_m)$, where σ_m is a fixed minimum value we choose to avoid numerical instabilities that would otherwise arise from $\sigma \to 0$ in, e.g., Eq. 4.11. Fig. 4.5 shows the progression of the blur throughout optimization.

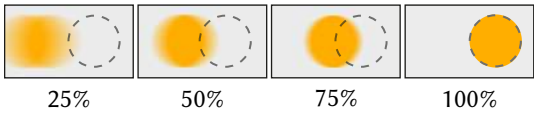


Figure 4.5: We visualize the adaptive spread of the smoothing at $n\%$ of the optimization. The reference position is shown dotted.

4.3.4 Implementation

We outline our gradient estimator in pseudo-code in Alg. 4. We importance-sample for our kernel with zero variance, use antithetic sampling and adapt the smoothing strength via σ. As Alg. 4 shows, our method is simple to implement and can be incorporated into existing frameworks with only a few lines of code. We implement our method in PyTorch, with Mitsuba as rendering backbone, and use Adam as our optimizer. For the remainder of this work, we use all components unless otherwise specified: importance sampling, adaptive smoothing and antithetic sampling. Moreover, we use Eq. 4.10 for computational efficiency (see Sec. 4.4.3) if not specified otherwise.

4.4 Experiments

We analyze our method and its variants in qualitative and quantitative comparisons against other methods and further compare their runtime performance. For the hyperparameters we use for our method and the competitors, please see Appendix Tab. B.1.

Algorithm 4 Gradient estimation of the scene function f at parameters θ with perturbations $\tau \sim \mathcal{N}(0, \sigma)$ at N samples.

```
# Equation 10
procedure EstimateGradient(P, θ, σ, N)
    G := 0
    for i ∈ (1, N/2) do
        ξ ← UNIFORM(0, 1)
        τ ← sqrt(−2σ² log(ξ))
        G ← G + P(θ + τ) − P(θ − τ)
    end for
    return G / N
end procedure
```

4.4.1 Methodology

Methods: For our experiments, we compare four methods. The first is a differentiable rasterizer, SoftRas [179]. Recall that soft rasterizers implicitly remove plateaus, which is why they are included here, despite their shortcomings for more complex forms of light transport. For our method, we evaluate its two variants: The first uses a differentiable renderer and weights its gradients ($\texttt{Our}_{\kappa\partial P}$, Eq. 4.9), while the second one performs differentiation through perturbation ($\texttt{Our}_{\partial\kappa P}$, Eq. 4.10). For both, we use Mitsuba 3 as our backbone, in the first variant using its differentiation capabilities to compute ∂P, in the latter using it as a non-differentiable black-box to compute only P. We run all methods for the same number of iterations and with the same rendering settings (samples per pixel (spp), resolution, path length, etc.).

Metrics: We measure the success of an optimization on two metrics, image-space and parameter-space MSE. As is common in inverse rendering, image-space MSE is what the optimization will act on. Parameter-space MSE is what we employ as a quality control metric during our evaluation. This is necessary to interpret whether the optimization is working correctly once we have hit a plateau, as the image-space MSE will not change there. Note that we are not optimizing parameter-space MSE and optimization never has access to this metric.

Tasks: We evaluate our method and its competitors on six optimization tasks that feature advanced light transport, plateaus and ambiguities. We show a conceptual sketch of each task in Fig. 4.6 - Fig. 4.7 and provide a textual explanation next.

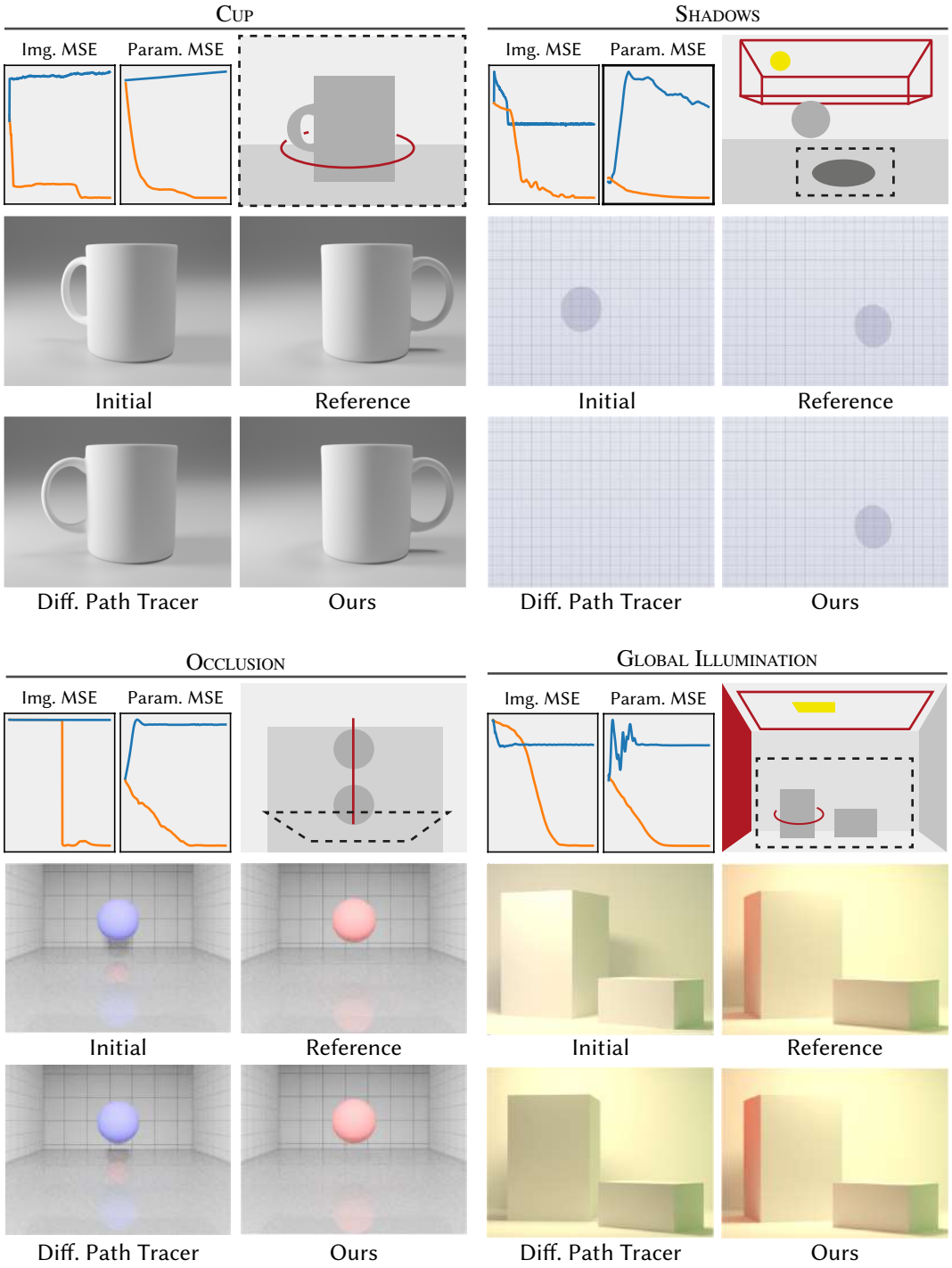


Figure 4.6: We show the optimization tasks and results for $\texttt{Our}_{\partial\kappa P}$ ("Ours", orange) and our baseline Mitsuba ("Diff. Path Tracer", blue) on the tasks Cup and Shadows (top) and Occlusion and Global Illumination (bottom).

4.4.2 Results

Qualitative: We now discuss our main result, see Fig. 4.6 and Fig. 4.7.

Cup: A mug is rotated around its vertical axis and as its handle gets occluded, the

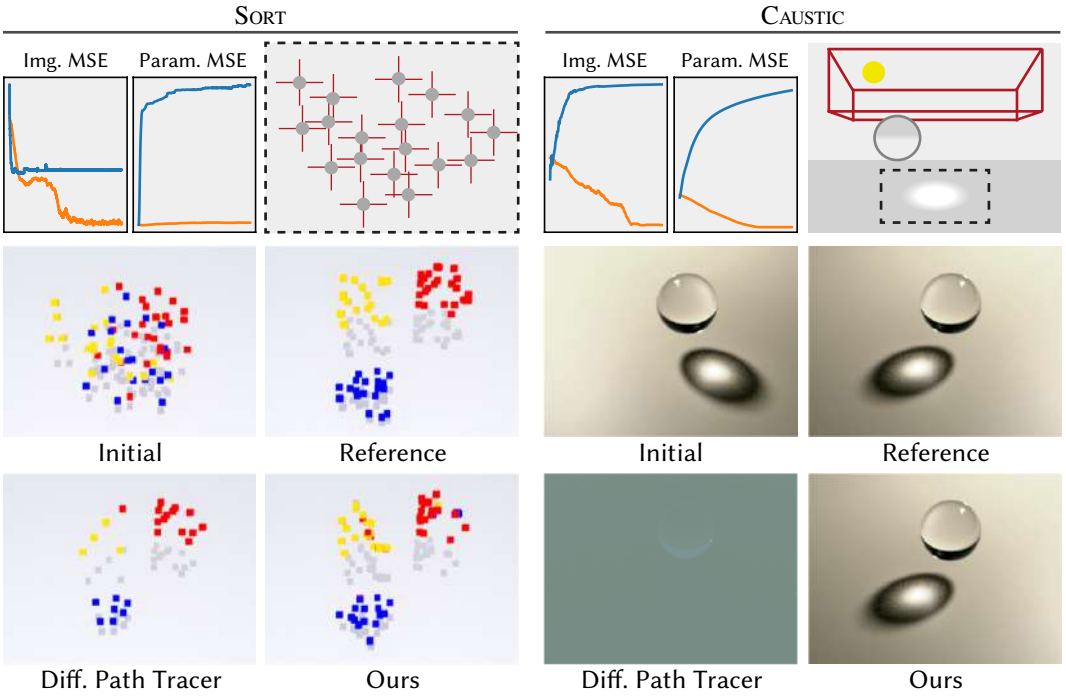


Figure 4.7: We show the optimization tasks and results for $\texttt{Our}_{\partial\kappa P}$ ("Ours", orange) and our baseline Mitsuba ("Diff. Path Tracer", blue) on the tasks Sort and Caustic.

Table 4.2: Image- and parameter-space MSE of different methods (columns) on different tasks (rows).

	Rasterizer		Path Tracer					
	SoftRas		Mitsuba		$\texttt{Our}_{\partial\kappa P}$		$\texttt{Our}_{\kappa\partial P}$	
	Img.	Para.	Img.	Para.	Img.	Para.	Img.	Para.
Cup	3.66×10^{-1}	2.72×10^{-2}	5.49×10^{-3}	0.75×10^{-1}	$\mathbf{4.92\times10^{-6}}$	$\mathbf{4.18\times10^{-7}}$	4.75×10^{-4}	2.77×10^{-1}
Shadows	1.64×10^{-3}	1.42×10^{-1}	1.64×10^{-3}	5.06×10^{-0}	$\mathbf{1.74\times10^{-5}}$	$\mathbf{1.81\times10^{-3}}$	5.12×10^{-4}	1.28×10^{-0}
Occl.	5.33×10^{-2}	7.18×10^{-3}	5.85×10^{-2}	$5.23\times10^{+1}$	$\mathbf{2.34\times10^{-4}}$	$\mathbf{3.29\times10^{-3}}$	5.37×10^{-2}	$1.87\times10^{+1}$
Global Ill.	–	–	3.78×10^{-2}	3.87×10^{-1}	$\mathbf{5.07\times10^{-5}}$	$\mathbf{8.71\times10^{-4}}$	5.88×10^{-2}	2.55×10^{-1}
Sort	1.85×10^{-2}	1.57×10^{-0}	1.18×10^{-2}	6.64×10^{-0}	$\mathbf{3.81\times10^{-3}}$	$\mathbf{4.19\times10^{-1}}$	1.02×10^{-2}	2.24×10^{-0}
Caustic	–	–	3.12×10^{-1}	8.50×10^{-0}	$\mathbf{1.89\times10^{-5}}$	$\mathbf{9.76\times10^{-5}}$	2.42×10^{-1}	4.03×10^{-0}

optimization has reached a plateau. Our method differentiates through the plateau. The differentiable path tracer gets stuck in the local minimum after slightly reducing the loss by turning the handle towards the left, due to the direction of the incoming light.

Shadows: An object outside of the view frustum is casting a shadow onto a plane. Our goal is to optimize the hidden object's position. Differentiable rasterizers can not solve this task, as they a) do not implement shadows, and b) cannot differentiate what they do not rasterize. The plateau in this task originates from the fact that the

shadows do not overlap in the initial condition, which creates a situation akin to Fig. 4.2 b). Again, our method matches the reference position very well. Mitsuba, first moves the sphere away from the plane (in negative z-direction), as this reduces the footprint of the sphere's shadow on the plane and thus leads to a lower error, and then finally moves the sphere out of the image, where a plateau is hit and the optimization can not recover. The blue line in the image-space plot in Fig. 4.6 b) illustrates this problem, as the parameter-error keeps changing very slightly, but the image-space error stays constant.

OCCLUSION: Here, a sphere translates along the viewing axis to match the reference. The challenge is that the sphere initially is occluded by another sphere, i.e., we are on a plateau as long as the occluder is closer to the camera than the sphere we are optimizing. The baseline path tracer initially pushes the red sphere towards the back of the box, as this a) reduces the error in the reflection on the bottom glass plane, and b) lets the shadow of the red sphere (visible underneath the blue sphere in the initial configuration) disappear, which again leads to a lower image-space error. Our method, in contrast, successfully differentiates through both the plateau (which originates from the red sphere having a negligible effect on the objective function) and the discontinuity that arises when the red sphere first moves closer to the camera than the blue occluder.

GLOBAL ILLUMINATION: We here show that our method is compatible with the ambiguities encountered in advanced light transport scenarios. The goal of this optimization task is to simultaneously move the top-light to match the scene's illumination, change the left wall's color to create the color bleeding onto the box, and also to rotate the large box to an orientation where the wall's reflected light is actually visible. The optimization only sees an inset of the scene (black in Fig. 4.6 d) and hence only ever sees the scattered light, and never the wall's color or light itself. The baseline cannot resolve the ambiguity between the box's rotation, the light position and the wall's color, as it is operating in a non-smoothed space. For a plot of the results, not just the inset, see Appendix Fig. B.1. Our method finds the correct combination of rotation, light position and wall color.

SORT: In this task, we need to sort a randomly perturbed assortment of 75 colored primitives into disjoint sets. In this problem, we optimize the x- and y-coordinates of each cube, which leads to a 150-dimensional setting, and hence have a plateau in each dimension, as most of the cubes are initially not overlapping their reference. Mitsuba cannot find the correct position of non-overlapping primitives and moves them around to minimize the image error, which is ultimately achieved by moving them outside of the view frustum. Our method, admittedly not perfect on this task, finds a more correct positions, more similar to the reference.

CAUSTIC: Lastly, the CAUSTIC task features a light source outside the view frustum illuminating a glass sphere, which casts a caustic onto the ground. The goal is to optimize the light's position in order to match a reference caustic. As the sphere does not change its appearance with the light's movement, the optimization has to solely rely on the caustic's position to find the correct parameters. Similar to the GI task, this is not solvable for rasterizers. Our method reaches the optimum position with high accuracy. For the baseline path tracer, we see a failure mode that is similar to the SHADOW task. In this case, the image space error can be reduced by moving the light source far away, as the bulk of the error comes from the caustic not being cast onto the correct position. Moving the light source far away reduces this error, but also leads into a local minimum where there is no illumination at all, resulting in the gray image in Fig. 4.7.

Quantitative: Tab. 4.2 reports image- and parameter-space MSE for all methods across all tasks. The quantitative results confirm what was conveyed visually: regular gradient-based path tracers that operate on non-smooth loss landscapes fail catastrophically on our tasks. `SoftRas` manages to overcome some plateaus, but struggles with achieving low parameter error, as it blurs in image space but must compare to the non-blurred reference (as all methods), which leads to a notable difference between the final state and the reference parameters. To achieve comparable image-space errors, we render the parameters found by `SoftRas` with Mitsuba. Our method $\texttt{Our}_{\partial \kappa P}$, in contrast, achieves errors of as low as 10^{-7}, and consistently outperforms its competitors on all tasks by several orders of magnitude. Interestingly,

$\texttt{Our}_{\kappa\partial P}$ (i.e., using the gradients from the differentiable renderer) works notably worse than $\texttt{Our}_{\partial\kappa P}$ (but mostly still outperforms Mitsuba). We attribute this to the fact that we cannot importance-sample for the gradient here, as we do not know its PDF. Instead, we can only draw samples proportional to the first term in the product, $\kappa(\tau)$, which places many samples where the kernel is high, i.e., at the current parameter value. As we can see from the rigid optimization by Mitsuba, the gradient at the current parameter position is not very informative, so placing samples there is not very helpful.

4.4.3 Timing

We now compare our approach's performance and will see that, while $\texttt{Our}_{\kappa\partial P}$ needs more time to complete an optimization, $\texttt{Our}_{\partial\kappa P}$ on average is $8\times$ faster than differentiable rendering with Mitsuba.

Our method requires the additional step of (over-) sampling the parameter space in order to compute our smooth gradients. However, as shown in Eq. 4.10, our stochastic gradient estimation through the derivative-kernel ($\texttt{Our}_{\partial\kappa P}$) allows us to skips the gradient computation step of the renderer. While there exist techniques like the adjoint path formulation [220] and path replay backpropagation [306], the gradient computation in inverse rendering still is computationally expensive. Additionally, correct gradients w.r.t. visibility-induced discontinuities require a special integrator (re-parametrization or edge sampling) and the creation of a gradient tape or compute graph.

Our method $\texttt{Our}_{\partial\kappa P}$, in contrast, does not need to compute $\partial P/\partial\theta$ and only requires a forward model. We can hence conveniently use the regular path integrator instead of its re-parametrized counterparts, and skip the gradient computation altogether. Moreover, our earlier efforts to develop an efficient importance-sampler will now pay off, as our method converges with as few as one extra sample only. This results in notable speedups, and $\texttt{Our}_{\partial\kappa P}$ hence significantly outperforms other differentiable path tracers in wall-clock time at otherwise equal settings (see Tab. 4.3).

Table 4.3: Timing comparison for the three path tracing variants on all tasks. We report the average time for a single optimization iteration (with same hyperparameters) in seconds, so less is better.

	CUP	SHAD.	OCCL.	GI	SORT	CAUS.
`Mitsuba`	1.12 s	0.64 s	0.37 s	0.44 s	2.88 s	1.02 s
$\texttt{Our}_{\partial\kappa P}$	**0.10 s**	**0.04 s**	**0.09 s**	**0.15 s**	**2.23 s**	**0.08 s**
$\texttt{Our}_{\kappa\partial P}$	2.22 s	1.43 s	0.91 s	1.72 s	32.02 s	4.02 s

4.4.4 Ablation

We now ablate our method to evaluate the effect its components have on the success of the optimization outcome. We will use $\texttt{Our}_{\partial\kappa P}$ from Tab. 4.2 as the baseline and ablate the following components: importance sampling (`noIS`), adaptive perturbations (`noAP`) and antithetic sampling (`noAT`). We hold all other parameters (spp, resolution, etc.) fixed and run the same number of optimization iterations that was also used in Tab. 4.2 and in our previous experiments (e.g., Fig. 4.6).

We report the relative change between the ablations and our baseline in Tab. 4.4. We report log-space values, as the results lie on very different scales. From the averages in the last row, it becomes apparent that all components drastically contribute to the success of our method, while the most important part is the antithetic sampling. We emphasize that importance- and antithetic-sampling are *variance reduction* techniques that do not bias the integration, i.e., they do not change the integral's value in expectation. Therefore, our approach should converge to similar performance without these components, but it would take (much) longer, as the gradient estimates will exhibit more noise.

4.5 Discussion

Related Approaches Other methods proposed blurring by down-sampling in order to circumvent plateaus [255; 157]. The quality upper bound for this is `SoftRas`, which we compare against in Tab. 4.2, as blurring by down-sampling does not account for occlusion, whereas `SoftRas` uses a smooth z-test. Another method that could be tempting to employ is FD. Unfortunately, FD does not scale to higher problem dimensions, as it requires $2n$ function evaluations on an n-dimensional

Table 4.4: Ablation of different components (columns) for different tasks (rows). We report the log-relative ratio w.r.t. $\texttt{Our}_{\partial\kappa P}$, so higher values mean higher error.

	noIS		noAP		noAT	
	Img.	Para.	Img.	Para.	Img.	Para.
CUP	4.75×	6.30×	6.96×	11.89×	3.19×	4.27×
SHAD.	5.27×	5.98×	3.07×	1.27×	5.30×	6.03×
OCCL.	3.18×	3.29×	8.73×	8.62×	8.73×	9.65×
GI	10.38×	10.84×	6.40×	9.15×	5.62×	12.06×
SORT	1.48×	0.70×	1.14×	2.04×	1.59×	1.09×
CAUS.	3.76×	8.35×	0.69×	1.70×	4.24×	9.27×
Mean	4.81×	5.91×	4.50×	5.78×	4.78×	7.06×

problem (on our SORT task, this would increase the per-iteration runtime by $\times 375$). A more economical variant is SPSA, which perturbs all dimensions at once [283]. However, neither FD nor SPSA actively smoothes the loss landscape, as the gradient is always estimated from exactly two measurements, taken at fixed locations, often from a Bernoulli distribution. Our approach, in contrast, uses N *stochastic* samples, where N is independent of the problem dimension. In fact, we use $N = 2$ for most of our experiments (Appendix Tab. B.1contains more details on experiment parametrization). Our method's advantages thus are twofold: first, we do not require cubature over the parameter space, but instead explore the space by stochastically sampling it. Second, our developed formalism allows to interpret this stochastic sampling as a means to compute a MC-estimate of the gradient, and thus enables to simultaneously smooth the space and perform (smooth) differentiation.

Indeed, the formalism developed in Sec. 4.3.1 can be interpreted as a form of variational optimization [285; 286], where one would descend along the (smooth) variational objective instead of the true underlying function. As such, Eq. 4.10 can be seen as an instance of a score-based gradient estimator [292], while Eq. 4.9 can be interpreted as reparametrization gradient [151; 270]. Suh et al. [290] provide intuition on each estimator's performance and align with our findings of the score-based estimator's superiority under a discontinuous objective. It is one of the contributions of this work to connect these variational approaches with inverse rendering.

Limitations As our method relies on Monte Carlo estimation, the variance increases favourably, but still increases with higher dimensions. This can usually be mitigated by increasing the number of samples N. We show examples of a high-dimensional texture optimization in Appendix Fig. B.5. Moreover, a good initial guess of σ is helpful for a successful optimization outcome (see Appendix Fig. B.3and Appendix Tab. B.2). We recommend setting σ to roughly 50 % of the domain and fine-tune from there, if necessary.

4.6 Conclusion

We have proposed a method for inverse rendering that convolves the rendering equation with a smoothing kernel. This has two important effects: it allows straightforward differentiation and removes plateaus. The idea combines strengths of differentiable rasterization and differentiable path tracing. Extensions could include applying our proposed method to path tracing for volumes or Eikonal transport [20; 328] or other fields that suffer from noisy or non-smooth gradients, such as meta-learning for rendering [64; 174]. Our approach is simple to implement, is efficient, has theoretical justification and optimizes tasks that existing differentiable renderers so far have diverged upon.

Chapter 5

ZeroGrads: Learning Local Surrogate Losses for Non-Differentiable Graphics

Abstract

Gradient-based optimization is now ubiquitous across graphics, but unfortunately can not be applied to problems with undefined or zero gradients. To circumvent this issue, the loss function can be manually replaced by a "surrogate" that has similar minima but is differentiable. Our proposed framework, *ZeroGrads*, automates this process by learning a neural approximation of the objective function, which in turn can be used to differentiate through arbitrary black-box graphics pipelines. We train the surrogate on an actively smoothed version of the objective and encourage locality, focusing the surrogate's capacity on what matters at the current training episode. The fitting is performed online, alongside the parameter optimization, and self-supervised, without pre-computed data or pre-trained models. As sampling the objective is expensive (it requires a full rendering or simulator run), we devise an efficient sampling scheme that allows for tractable run-times and competitive performance at little overhead. We demonstrate optimizing diverse non-convex, non-differentiable black-box problems in graphics, such as visibility in rendering, discrete parameter spaces in procedural modeling or optimal control in physics-

Figure 5.1: Our method optimizes arbitrary (black-box) graphics pipelines, which all do not trivially provide gradients, such as rendering (left: finding the box and light position under a discontinuous integrand), modelling (middle: discrete number of vertical and horizontal wicker stakes), and animation (right: optimal control of the engine turn-off time), by fitting a neural network to the loss landscape and then using the network's gradients for parameter optimization.

driven animation. In contrast to other derivative-free algorithms, our approach scales well to higher dimensions, which we demonstrate on problems with up to 35,000 interlinked variables.

5.1 Introduction

Gradient-based optimization has recently become an essential part of many graphics applications, ranging from rendering to find light or reflectance [64; 249; 80], over procedural material modelling [121] to animation of characters or fluids [99; 269]. These methods provide state-of-the-art results, in particular when combined with large amounts of training data and NNs that represent the desired mappings or assets. In order to train these methods via GD, the pipeline needs to be fully differentiable, allowing to backpropagate gradients from the objective function to the optimization parameters. This often is enabled by intricate, task-specific derivations [183; 169] or requires fundamental changes to the pipeline, such as the switch to a dedicated programming language [12; 123; 10; 132].

In practice, the application of these ideas remains limited, as many existing graphics pipelines are black-boxes (e.g., entire 3D modelling packages such as Blender or rendering pipelines such as Renderman [39] or Unity [98]) that do not provide access to their internal workings and hence cannot be differentiated. Further, even if we were given access to the pipeline's internals, the employed functions might not be differentiable (e.g., the step function) or provide gradients that are

insufficient for convergence [192]. The mindset of this chapter is that we only have access to a forward model, e.g., a modeling pipeline, and a reference, e.g., a target image. Using these two, a loss can be computed – its gradients, however, cannot be used.

If a loss is not differentiable in practice, it can be approximated by a "surrogate" loss (Fig. 5.2). The surrogate is a function that has similar minima as the true loss, but also provides gradients that are useful when employed in an optimization. While the concept of surrogate modelling is not new (see Sec. 5.2), it remains unclear how to efficiently find a surrogate loss for any arbitrary graphics pipeline, as here the sampling must be sparse (recall, rendering a sample is expensive), and the optimization problems' dimensionality can vary by several orders of magnitude. In this chapter, we propose *ZeroGrads*, a systematic and efficient way of learning local surrogate losses, requiring no more than a forward model and a reference.

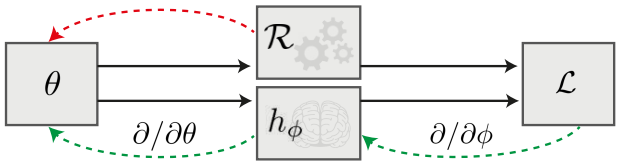


Figure 5.2: Regular forward models $\mathcal{R}$ might not be able to provide gradients w.r.t. their input parameters θ (red arrow, top). Our approach, ZeroGrads, provides this ability via a local learned surrogate h (green arrows, bottom) that maps θ to the associated loss and can be differentiated analytically.

To learn our surrogate loss and use it in an optimization, we follow four key steps: We first smooth the original loss by convolving the cost landscape with a blur kernel, so that it provides gradients that lead to (improved) convergence when used during optimization. We secondly fit a surrogate, a parametric function such as a neural network or a quadratic potential, to that smoothed loss. As the surrogate is differentiable by construction, we can query it to get *surrogate gradients* that drive the parameter optimization. We thirdly constrain this fit to the local neighbourhood of the current parameters, as the global cost landscape is in large parts irrelevant for the current optimization step. By locally updating our surrogate, we allow it to focus on what matters "around" the current solution. However, querying the objective function to create samples for our surrogate fitting is expensive, as each sample requires the execution of the full forward model, such as

a light transport simulation or physics solver. Therefore, fourth and finally, we derive an efficient sampler that reduces the variance of the surrogate's gradient estimates and thus allows us to use ZeroGrads with a low number of sparse samples at tractable runtimes.

In contrast to prior work [122], our local surrogate losses can be trained *online*, alongside the actual parameter optimization, and *self-supervised*, without the need for pre-trained models or pre-computed ground truth gradients. Moreover, in comparison to other gradient estimation techniques [65; 283], our neural proxy allows us to move the noise and variance in the gradient estimates from the *parameter* domain into the *proxy* domain, where it is naturally smoothed by the network's hysteresis, allowing ZeroGrads to scale up to very high dimensions. In summary, our contributions are:

- ZeroGrads, a framework that maps a forward model without usable gradients into a smooth, differentiable surrogate function, such as an NN, with analytic gradients.

- Reducing the variance of surrogate and parameter updates to allow tractable runtimes and unsupervised and successful on-the-fly optimization.

- Optimizing several non-differentiable black-box graphics problems from rendering (visibility) over modelling (discrete procedurals) to simulation and animation (optimal control).

- Showing that our surrogates scale favourably to higher dimensions, with up to several thousand correlated optimization variables, where existing derivative-free methods often struggle to converge.

- A publicly available implementation of our method and benchmarks at `https://github.com/mfischer-ucl/zerograds`

We would like to emphasize that we do not claim superiority (less variance, better convergence, ...) over existing, specialized methods such as Mitsuba [130], Redner [169] or PhiFlow [115], but rather broaden the toolkit of inverse graphics

solvers by now enabling gradient-based optimization on arbitrary, non-differentiable graphics pipelines.

5.2 Previous Work

Optimization in Graphics. Parameters of graphics models are now routinely optimized so as to fulfill user-provided goals. The two main ingredients enabling this are gradient-based optimization and tunable architectures. We will not consider the many different exciting architectures in this work, but focus on the optimization itself. Gradients are typically computed by using a language that allows efficient auto-differentiation, such as PyTorch [231] or JAX [30], often targeting GPUs. Unfortunately, several problems in graphics are not differentiable.

Gradient-free optimization algorithms [260] such as asking a user [185], global optimization [137], direct search [242], SA [152], SPSA [283], particle swarms [230] or GAs [116] have largely fallen from favour in everyday graphics use. This is partially due to the fact that gradient-free optimization – even on smooth problems – often requires a large number of function evaluations before converging [134], and, in general, struggles with convergence as problem dimensionality increases. Additionally, gradient-free optimizers often suffer from high per-iteration cost in higher dimensions (e.g., FD or [331], see Tab. 5.1), require the computation of the Hessian matrix in Newton-type methods, or make use of covariance-matrices (CMA-ES [101]), whose memory requirements grow quadratically with problem dimension and require computationally complex steps like inversion or eigendecomposition. The aforementioned aspects often lead to trade-offs between performance, scalability and runtime that practitioners have to take into consideration. GD, in contrast, has strong convergence guarantees (under convex objectives), is highly scalable, can be parallelized effectively, and has shown superior performance in high-dimensional settings (e.g., NN training). Unfortunately, it cannot be used on many relevant problems (although the cost landscape itself might be smooth), as many graphics pipelines are simply not designed to be differentiable. Our approach circumvents this issue by fitting a model of the cost landscape, which can then be differentiated to

Table 5.1: Comparison of optimization algorithms. n is the problem dimensionality, p is the population size in evolutionary algorithms, k is the state's size in stateful algorithms, b is the batchsize in multi-sample algorithms, t and t_i are the times required for a function evaluation and a state update, respectively. Stateful denotes whether the method maintains a state other than the current optimization parameter.

	Memory	Gradient	Iter. Cost	Scalable	Robust	Stateful
Ours	$n+k$	✓	$b \cdot t + t_{\text{GD}}$	✓	✓	✓
SPSA	n	✓	$2t$	✓	✓	×
FR22	n	✓	$b \cdot t$	✓	?	×
CMA-ES	$n^2 + \text{k}$	×	$p \cdot t + t_{\text{CMA}}$	✓	✓	✓
GA	n	×	$p \cdot t + t_{\text{GA}}$	×	×	✓
FD	n	✓	$2n \cdot t$	×	×	×
SA	n	×	t	×	×	×

provide *surrogate* gradients that GD can work with. Even for non-smooth problems, our formulation makes the problem smooth and hence amenable to GD.

Differentiation. Most graphics pipelines used in production (e.g., Blender, GIMP, Photoshop) are not differentiable, as they are not implemented in a differentiable programming language. The deeper underlying mathematical problem is that their output often relies on integration – however, differentiation and the typical integral estimation through MC cannot be interchanged without further considerations. A prominent example are discontinuities in rendering, which have sparked a body of work by Lee et al. [163]; Bangaru et al. [12]; Loper and Black [182]; Kato et al. [143]; Li et al. [169]; Liu et al. [179]; Rhodin et al. [259]; Xing et al. [322] or Loubet et al. [183], to only name a few salient ones. Similar problems appear in vector graphics [170], signed distance functions (SDFs) [307; 13], entire programs [33] or physics [123; 35; 198]. All these approaches require access to the internals of the graphics pipeline in order to replace or change parts such that gradients can be backpropagated. Our approach, in contrast, assumes the pipeline to be a black-box and does not make any assumptions or approximations to the internals.

Gradient Smoothing. Another approach for optimizing non-differentiable problems has been proposed as *stochastic* optimization. Here, discontinuities and plateaus are

smoothed out by optimizing over the expected value of a distribution (generally MC-approxim. via randomized sampling) instead of a rigid parameter [22; 36; 57; 286], with the resulting gradient sometimes being referred to as *zeroth-order* [290]. This has successfully been used to smooth out plateaus and discontinuities in rendering ([65; 161]), contact dynamics in robotics ([291; 196]) and policy optimization in reinforcement learning ([318; 290]), albeit without the explicit learning of a proxy cost model.

Proxies. A key insight is that we only need the loss' gradients, and not those of the entire pipeline. Hence, if a part in a conventional graphics pipeline cannot be differentiated, we search for a similar function that we can differentiate instead, our *proxy*. As the proxy is an analytic function, the gradients w.r.t. its input can readily be computed via AD and then be used for optimization. Graphics is a good fit for neural proxies, as we can freely sample the objective in many cases, e.g., by rendering an image or running a simulation (a "simulation-optimization" setting [159]). While easy to do, creating a sample is expensive.

The concept of "Neural Proxies" was pioneered for physics by Grzeszczuk et al. [90] and is now applied to problems such as material editing [121], photo editing [300; 68], hardware and design [299], software synthesis [62], simulation [204; 257; 7] or animation [276; 208; 278]. Rendering itself becomes differentiable when replaced by a NN proxy [206], however, having a NN emulate the complex behaviour of a full graphics pipeline might not scale to complex assets.

Surrogate losses. Surrogate losses [246] (sometimes also called meta-models [16; 28]) extend the idea of proxies by providing an approximation to the *entire* forward model's behaviour by only emulating its response (or loss landscape), without necessarily replicating all the internals of the mdoel. Differentiating the surrogate will provide gradients which can then be used for (gradient-based) optimization. Surrogates are especially popular when taking a sample is expensive, like in airplane design [75] or neural architecture search [336], and can be modelled in a number of ways, e.g., through polynomials [137], radial basis functions (RBFs) [96] and recently neural networks [89; 233]. Most of these methods learn surrogates for

the entire cost landscape (typically in a simplified setting, e.g., classification), with the exception of response surface maps (RSMs), which fit a first- or second order polynomial to the local neighbourhood, but are known to not converge on higher-dimensional problems [311]. More crucially, aside from global fitting, most methods assume the availability of a large set of data samples, e.g., a curated image collection like ImageNet [48]. In our setting, in contrast, sampling is expensive, which is why we sample *sparsely and locally* and fit the surrogate in-the-loop, during optimization. Our method hence extends the family of surrogate-based optimizers, and, in contrast to previous work, scales to a wide range of optimization tasks in high dimensions.

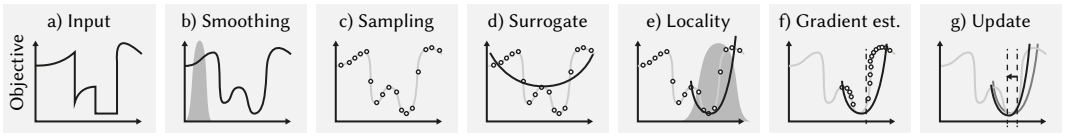


Figure 5.3: A conceptual illustration of our approach; each subplot shows a one-dimensional cost landscape. The x-axis denotes the parameter position. For details please refer to Sec. 5.3: Overview.

Gradient estimation. The surrogate itself is updated gradient-based, by sampling the objective function a finite number of times and then estimating the surrogate's gradient from its prediction error. For general learning, building gradients is fundamentally a MC estimation problem [195], akin to what graphics routinely is solving for rendering [304]. We identify the similarity of estimating proxy gradients and simulating light transport (high dimensionality, sparsity, product integrands) and employ variance reduction based on importance sampling [304; 139; 140] the local parameter neighbourhood to increase the efficiency of our surrogate gradient estimates.

5.3 Our approach

Given a scalar objective function $f(\theta) : \Theta \rightarrow \mathbb{R}_0^+$ over an n-dimensional parameter space Θ, we would like to find the optimal parameters θ^* that minimize f. Typically, gradient-based optimizers such as SGD or ADAM are used for such a task. However, in the setting of this work, their direct use is not possible, as the objective's gradients $\partial f / \partial \theta$ are either not accessible (in a black-box pipeline), undefined (at

discontinuities), zero (on a plateau) or too costly to compute (e.g., when appearing in an integral). We are, however, able to sample this objective function by sampling a set of parameters and then comparing the resulting output with the reference. We propose to now *locally* fit a tunable and differentiable surrogate function $h(\theta, \phi)$ to those samples, whose derivative $\partial h/\partial \theta$ will act as *surrogate gradient* and drive the optimization.

Overview Our approach is summarized in Alg. 5 and Fig. 5.3. Given an arbitrary (potentially non-smooth and non-convex) objective function (Fig. 5.3a, also called loss landscape) and a randomly initialized parameter state θ, we first smooth the objective via convolution with a Gaussian kernel in order to reduce plateaus (Sec. 5.3.1 and Fig. 5.3b). We subsequently fit our surrogate h (Sec. 5.3.2) to this smooth objective. However, sampling is expensive (requiring a full rendering or simulator run), and global sampling and surrogate fitting would be very approximate (Fig. 5.3c, d), which is why we enforce locality via another Gaussian kernel (Fig. 5.3e and Sec. 5.3.3) and hence encourage the surrogate to focus on what matters at the current optimization iteration. Unfortunately, we do not have supervision gradients to train our surrogate on, which is why we must estimate the surrogate's gradient (Sec. 5.3.4). While some samples are more informative than others, it is unclear how to find those, i.e., how we can efficiently sample this convolved space. To this end, we derive an efficient importance-sampler (Fig. 5.3f and Sec. 5.3.5) that samples according to the locality terms and thus reduces the variance of the estimated surrogate gradient. Finally, we use this estimated gradient to update our surrogate (Fig. 5.3g) in order to improve its fit to the objective. We can then descend along the surrogate's gradient $\partial h/\partial \theta$ (readily available via AD) to update the optimization parameter θ and repeat the process from b) onward, i.e., the surrogate is updated from its previous state instead of re-fit. Lines 5 and 6 in Alg. 5 illustrate this, where OPTIMIZE performs gradient descent steps on a variable.

5.3.1 Smooth objective

As the objective might not always be differentiable (or provide gradients that are of little use [192]), we seek to find a function that has similar optima and is differentiable.

Figure 5.4: Samples of the smooth objective (bottom row) on which we learn our surrogate: Perturbing the rigid scene parameters (top row) smooths discontinuities, e.g., the binary on/off for the LED task (an inset is shown).

In practice, the issue is not so much in non-differentiable point singularities (which are even present in the popular ReLU activation), but regions with zero gradients ("plateaus"). These can be removed by convolving the objective with a blur kernel. Similar ideas have been applied to rasterization [179; 237] and path-tracing [65], which we scale to arbitrary spaces. We define the smooth objective $g(\theta) : \Theta \to \mathbb{R}_0^+$ as

$$g(\theta) = \kappa * f(\theta) = \int_\Theta \kappa(\tau) f(\theta - \tau) \mathrm{d}\tau, \tag{5.1}$$

a convolution of the objective f and a smoothing kernel κ, which we choose to be a Gaussian. Convolution with a Gaussian kernel has several desirable properties, e.g., convexity is preserved, it holds that $L_g \leq L_f$, i.e., the smooth objective is stronger Lipschitz-bound than f, and the gradient ∇_g is Lipschitz-continuous even when ∇_f is not [209], as is the case on many of our problems (e.g., Fig. 5.3a)). We show visualizations of Eq. 5.1 in Fig. 5.4.

Algorithm 5 High-level pseudo-code of ZeroGrads.

Input: objective f, surrogate h
Output: optimized parameters θ minimizing f

procedure ZEROGRADS(f, h)
  ϕ := INIT() ▷ surrogate parameters
  θ := UNIFORM() ▷ optimization parameters
  for i **do**
    ϕ = OPTIMIZE$_1$(ϕ, ESTIMATEGRADIENT(ϕ, θ f, h))
    θ = OPTIMIZE$_2$(θ, $\partial h / \partial \theta$)
  end for
  return θ
end procedure

5.3.2 Surrogate

The key ingredient, the surrogate $h(\theta,\phi)\colon \Theta \times \Phi \to \mathbb{R}_0^+$, consumes θ (like f and g, which it emulates), but also takes the tunable parameters ϕ from the m-dimensional surrogate parameter space Φ.

We encode our surrogate in a differentiable proxy function of variable form, which can take the form of polynomials, RBFs or NNs (see Sec. 5.4.4 for examples) and whose analytic parametrization allows to easily get the surrogate gradient $\partial h/\partial\phi$ and the parameter gradient $\partial h/\partial\theta$ via automatic differentiation. In contrast to linear methods (e.g., [331; 65; 283]), our continuous surrogate formulation allows us to perform one or more gradient descent steps on the surrogate and to evaluate the estimated loss surface at a new position without re-running the forward model.

5.3.3 Localized surrogate loss

Matching h to g across the entire domain Θ might be too ambitious and furthermore is unnecessary, as most first-order gradient-based optimizers only ever query values at or around the current parameter θ. Instead, we create a surrogate that is focused around the current parameters by locally sampling the objective function.

The loss of the surrogate parameters ϕ "around" θ hence is

$$l(\theta,\phi) = \int_\Theta \lambda(\rho,\theta)\,(g(\rho) - h(\rho,\phi))^2\,\mathrm{d}\rho, \tag{5.2}$$

where λ is a weighting function that chooses how much context is considered around the current solution and ρ is from the parameter space Θ. We again choose a Gaussian with mean θ here, which is not to be confused with the smoothing kernel κ. Eq. 5.2 illustrates how the surrogate never has access to the gradients of the true objective – these might not even exist –, but learns self-supervised by only sampling the (smoothed) loss g.

5.3.4 Estimator

Combining the smoothed loss in Eq. 5.1 and the localized surrogate loss in Eq. 5.2, we arrive at the following expression:

$$l(\theta,\phi) = \int_\Theta \underbrace{\lambda(\rho,\theta)\left(\left[\int_\Theta \kappa(\tau) f(\rho-\tau)\mathrm{d}\tau\right] - h(\rho,\phi)\right)^2}_{:=I(\rho,\phi)} \mathrm{d}\rho. \tag{5.3}$$

We are now interested in the gradient of this expression w.r.t. the surrogate parameters ϕ, i.e.,

$$\frac{\partial}{\partial\phi} l(\theta,\phi) = \frac{\partial}{\partial\phi}\int_\Theta I(\rho,\phi)\mathrm{d}\rho = \int_\Theta \frac{\partial}{\partial\phi} I(\rho,\phi)\mathrm{d}\rho. \tag{5.4}$$

The above equality holds, according to Leibniz' rule of differentiation under the integral sign, if, and only if, the integrand is continuous in ϕ and ρ [169]. Our Gaussian locality weight λ fulfills this and $h(\rho,\phi)$ is continuous by definition, as it is a NN or quadratic potential. Through our previously introduced convolution (Eq. 5.1), the – originally discontinuous – objective f becomes smooth and hence leads to the inner integral being continuous in ρ. Leveraging the fact that a composition of continuous functions also is continuous, we can say that $I(\rho,\phi)$ is continuous in ϕ and hence use Eq. 5.4 as our gradient estimator:

$$\frac{\partial}{\partial\phi} l = \int_\Theta \frac{\partial}{\partial\phi}\lambda(\rho,\theta)\left(\left[\int_\Theta \kappa(\tau) f(\rho-\tau)\mathrm{d}\tau\right] - h(\rho,\phi)\right)^2 \mathrm{d}\rho \tag{5.5}$$

$$= \int_\Theta 2\lambda(\rho,\theta)\frac{\partial h(\rho,\phi)}{\partial\phi}\left(h(\rho,\phi) - \int_\Theta \kappa(\tau) f(\rho-\tau)\mathrm{d}\tau\right)\mathrm{d}\rho. \tag{5.6}$$

Here, the inner integral over τ is conditioned on the outer variable ρ, leading to a nested integration problem. A general, unbiased solution would estimate the inner and outer integrals with N and M samples, respectively, where $M \propto N$, leading to quadratic complexity. However, in this special case the function acting on the inner integral is linear, and the nested estimator thus remains unbiased even for constant N (see [251], Sec. 5 & Fig. 2).

We now aim to re-arrange Eq. 5.6 into a double-integral of a product, a form that

is reliably solvable by the well-known approaches that solve the rendering equation [304]. We hence write Eq. 5.6 as

$$\begin{aligned}\frac{\partial}{\partial\phi}l = \int_\Theta\int_\Theta 2\lambda(\rho,\theta)\frac{\partial h(\rho,\phi)}{\partial\phi}h(\rho,\phi)\mathrm{d}\tau - \\ \int_\Theta 2\lambda(\rho,\theta)\frac{\partial h(\rho,\phi)}{\partial\phi}\kappa(\tau)f(\rho-\tau)\mathrm{d}\tau\mathrm{d}\rho,\end{aligned} \tag{5.7}$$

where we use the fact that integrating an expression that is independent of the integration variable (here τ) reduces to multiplication by the volume of the integration domain, which we here assume to be normalized to unit volume, i.e., $\int_\Theta d\tau = 1$. Now the two inner integrals can be written as one, and factored as

$$\frac{\partial}{\partial\phi}l = \iint_\Theta 2\lambda(\rho,\theta)\frac{\partial h(\rho,\phi)}{\partial\phi}\left(h(\rho,\phi)-\kappa(\tau)f(\rho-\tau)\right)\mathrm{d}\tau\mathrm{d}\rho. \tag{5.8}$$

This integral spans the product space $\Theta\times\Theta$ and has the following unbiased estimator with linear complexity $\mathcal{O}(N)$:

$$\frac{\partial}{\partial\phi}l \approx \frac{1}{N}\sum_i^N \frac{2\lambda(\rho_i,\theta)}{p(\rho_i,\tau_i)}\frac{\partial h(\rho,\phi)}{\partial\phi}\left(h(\rho_i,\phi)-\kappa(\tau_i)f(\rho_i-\tau_i)\right), \tag{5.9}$$

$$= \frac{\partial}{\partial\phi}\frac{1}{N}\sum_i^N \frac{\lambda(\rho_i,\theta)}{p(\rho_i)p(\tau_i)}\left(h(\rho_i,\phi)-\kappa(\tau_i)f(\rho_i-\tau_i)\right)^2. \tag{5.10}$$

It seems somewhat contrived to go through all the above reformulations to arrive from Eq. 5.4 at Eq. 5.9. Note, however, that differentiating *inside* the integral, enabled by Eq. 5.4, removes the non-linear function acting on the inner integral in Eq. 5.3 and hence enables us to formulate an *unbiased* estimator of the smoothed, localized loss in $\mathcal{O}(N)$. This re-formulation is only possible in special cases (here, for a second-order polynomial acting on the inner integral, see [251] Sec. 4) and would not be possible for other popular distance measures between h and f, e.g., the Kullback-Leibler (KL)-divergence or the Hinge- or Exponential-losses, which would introduce bias into the optimization due to non-linearities in their derivatives [49; 216].

Algorithm 6 Estimating the surrogate's gradient

Input: surrogate parameters ϕ, optimization parameter θ, objective f, surrogate h
Output: surrogate gradient $\nabla\phi$

procedure ESTIMATEGRADIENT(ϕ, θ f, h)
  $\nabla\phi := 0$
  for N **do**
    $\rho, \tau :=$ NORMAL(σ_o), NORMAL(σ_i)
    $s := (h(\rho, \phi) - f(\rho - \tau))^2$
    $\nabla\phi$ += GRAD(s, ϕ)
  end for
  return $\nabla\phi / N$
end procedure

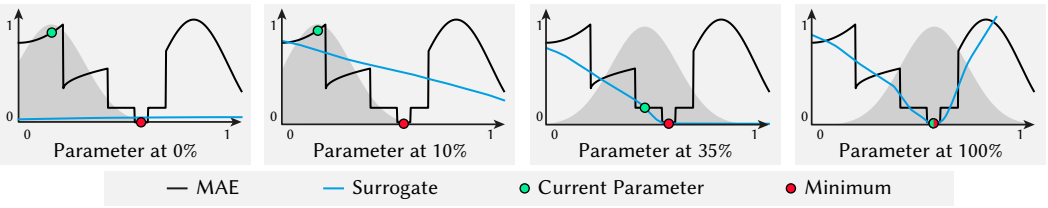


Figure 5.5: A 1D-example with the function from Fig. 5.3a) and our neural surrogate (blue), which learns a local approximation of the loss (black, MAE) and provides gradients for the optimization parameter (green). The sampling distribution is displayed in grey, state is shown at 0%, 10%, 35% and 100% total iterations.

5.3.5 Sampling

For uniform random sampling, the MC estimator in Eq. 5.9 will exhibit substantial variance, leading to slower convergence and hence longer runtimes. Instead, we would like to sample in a way that maximizes a sample's importance and therefore produces more meaningful gradients for the same sampling budget. Thanks to our reformulation of the nested integral, it is evident that the integration domain now is $\Theta \times \Theta$, and that the magnitude of the gradient – the quantity we would like to estimate – is determined by three factors: the difference between the surrogate's prediction and (smooth) objective f, and the locality terms λ and κ.

While we cannot trivially compute the PDF of the surrogate's prediction error, our local surrogate formulation shows that we can reduce the variance by importance sampling [304] for both locality terms. This again allows us to focus our surrogate on the regions of the parameter space that matter at the current optimization iteration while ignoring large amounts of space, which is especially helpful in higher dimen-

sions. The parameter σ_o of the locality λ determines how far the current solution's sampling radius is spread out, while σ_i determines the amount of smoothing, and is generally set to 15% of σ_o (for details, please see Appendix C.1.1). We display the resulting gradient estimator in Alg. 6.

5.3.6 Summary

In combination, the above elaborations allow us to optimize the objective function $f(\theta)$ through our surrogate's *surrogate gradients* $\partial h/\partial \theta$. We emphasize that, in contrast to prior work [121; 233; 89], our surrogate is learned self-supervised, without any ground truth supervision in the form of pre-computed gradients, and is optimized on-the-fly, alongside the parameter θ. This is made possible by a low number of samples which we achieve through our efficient estimators. As such, it allows the application of our method to systems where only a forward model is given. In the following sections, we will detail and evaluate some exemplary applications. Further, we provide a simple, illustrative 1D-example and visualize our learned surrogate loss over the course of the optimization in Fig. 5.5.

5.4 Evaluation

Our evaluation compares different methods on a range of tasks (see Sec. 5.4.3 and Appendix C.2for detailed task descriptions): we validate our design choices through ablations on lower-dimensional tasks in Sec. 5.4.3 and compare against established derivative-free optimizers on higher-dimensional, real-world tasks in Sec. 5.4.5. The reason for this is twofold: first, in the higher-dimensional regime, the task complexity makes it non-evident to see which of the ablated attributes lead to the method failing, and second, the derivative-free algorithms could solve some of the lower-dimensional tasks by simply brute-forcing the solution (which is a valid way of solving the problem, but besides the scope of interest here).

As our objective f, we use the MSE between the current rendered state and the target, if not otherwise specified. For details on our proxy's architecture and hyperparameters, please see Appendix C.1.

5.4.1 Methods

We compare our approach to several ablations and variants, out of which some correspond to existing published methods. All methods operate in *image space* only and do not have access to any ground truth supervision or parameters. We structure the space of methods by the type of i) smoothing, ii) surrogate, and iii) sampling. Our full method, `Ours`, implements Alg. 6: it smooths the loss via convolution with a Gaussian (Eq. 5.1), uses a neural proxy and draws samples by importance-sampling the locality terms. We compare our full approach to the following methods and ablations:

`NoSmooth`, where we ablate the smoothing convolution (Eq. 5.1) and directly sample the non-smooth loss.

`NoNN`, where we replace the NN in the surrogate by a quadratic potential function of the form $(\mathrm{x}, 1)^\intercal \mathsf{M} (\mathrm{x}, 1)$, where, for an n-dimensional problem, M is a symmetric matrix in $\mathbb{R}^{n+1 \times n+1}$.

`NoLocal`, where we ablate the locality by drawing uniform random samples from the domain. For unbound domains, we manually set reasonable boundaries.

`FD`, our implementation of finite differences, with an optimally chosen $\pm\varepsilon$ in each dimension.

`FR22`, which re-implements the approach presented by Fischer and Ritschel [65], who derive gradients by stochastically perturbing the optimization parameters θ at every iteration and weighting the resulting loss values by a gradient-of-Gaussian kernel, effectively creating a linear gradient estimate akin to a stochastic multi-sample version of Spall [283].

5.4.2 Protocol

The results we report are the median values over an ensemble of 10 independent runs of random instances of each task. To ensure fairness, all methods are run in their best configuration. For each run, the parameter initialization and ground truth (where possible) are randomly re-sampled and the surrogate, optimizer and all other stateful components are re-initialized from scratch.

Table 5.2: An overview of our method and its ablations and competitors. To the right, we show the competitors' relative error ratio at the iteration where our method achieves 95 % error reduction - i.e, how much others are behind.

Method	Smoothing	Surrog. h	Sampler	Rendering	Modeling	Animation
`NoSmooth`	None	NN	Gauss	1.2×	3.9×	1.5×
`NoNN`	Gauss	Quad.	Gauss	8.9×	792.4×	16.3×
`NoLocal`	Gauss	NN	Uniform	12.3×	613.4×	22.4×
`FD`	None	Linear	Box	24.5×	654.3×	10.2×
`FR22`	Gauss	Linear	Import.	11.0×	323.6×	3.3×
`Ours`	Gauss	NN	Gauss	1.0×	1.0×	1.0×

5.4.3 Tasks

We validate our method on two sets of tasks: the first set consists of lower-dimensional tasks from rendering, animation, modelling and physics, all of which are not trivially amenable to gradient-based optimization due to (partially) discrete parameter spaces, discontinuous integrals or non-differentiable frameworks, and serves the purpose of evaluating our approach's design decisions. We provide a short description of each task in the following section and refer the reader to Appendix C.2 for more details on task setup and to Fig. 5.7 for task-specific visualizations.

Additionally, we evaluate how well our method scales to more realistic, higher-dimensional inverse optimization problems on a second set of tasks in Sec. 5.4.5.

Differentiable Rendering. Discontinuities in rendering arise from the visibility function that appears inside the integral, in which case the gradient of the integral cannot be computed as the integral of the gradient. Typical solutions include re-parametrization or edge-sampling for path tracing [183; 169] or replacing step functions with soft counterparts in rasterization [237; 179] and use framework-specific implementations.

CORNELL-BOX We optimize the light's horizontal and vertical translation and the axial rotation of both boxes inside the Cornell box. The discontinuities arise from visibility changes at the silhouette edges of the moving boxes and light.

BRDF Here, we optimize the material properties (RGB reflectance and index of refraction (IOR)) of a material test-ball illuminated under environment illumi-

nation. Optimizing properties of ideal specular objects, such as their IOR, often is challenging even for modern differentiable path tracers [130]. Our method does not make any assumptions on the underlying function, so we can successfully optimize these cases, too.

MOSAIC In this task, we simultaneously optimize the vertical rotation of 320 cubes to match a reference. Again, the discontinuities arise for pixels at the silhouette edges that change their color discontinuously with the occluding cube's rotation.

Procedural Modeling. Procedural modeling mainly uses nodes from two categories, *filtering* nodes (that are differentiable by construction) and *generator* nodes, that operate on discrete parameters, such as a brick texture generator. Node relations often are highly non-linear due to complex material graphs and their interplay with other pipeline parameters. Moreover, the connections inside the nodegraph are a combinatorical problem with a highly discontinuous loss landscape. While previous research has made great progress in this field [121; 92; 275; 122; 166], it still often either involves lengthy, framework-specific pre-training, or relies on the existence of differentiable material libraries.

WICKER A procedural modelling scenario where we simultaneously optimize the parameters of a node graph that creates a woven wicker material. We optimize the number of horizontal and vertical stakes and the number of repetitions across the unit plane.

LED Given a target setting, we optimize each LED panel in a digital display to either be on or off. The display consists of 12 panels, each with 28 elements, leading to a 336-dim. binary problem.

NODE-GRAPH In this task, we directly optimize the connectivity of a material graph - a mixture of a combinatorical and procedural modelling problem. We encode all possible edges for a given set of nodes in a connection matrix and optimize the matrix entries. If an entry rounds to 1, the corresponding edge is inserted into the node graph, else removed. Our test graph has 8 nodes (several inputs and outputs each, 24 valid connections and matrix entries). Some graph edges constrain each other, e.g., when a shader node already is connected, a new connection will not result

in an updated image, making this an even more inter-linked problem.

Animation. Accurate animation often relies on differential equations to solve the underlying physics equations which govern a character's behaviour or movement. Oftentimes, the forward model of such an animation is inherently discontinuous (for instance, due to collisions) while at the same time, the underlying physics solver is not differentiable, as its output is the solution of an integral approximated by discrete sampling locations (the time steps). This is similar to the problem encountered in differentiable rendering: if the integrand (e.g., the time at which a force starts acting) is discontinuous, it is incorrect to simply differentiate the integral estimate to get a gradient estimate (see Appendix Fig. C.4).

ROCKET A physical simulation where we optimize the discrete event time at which a rocket's engine must be turned off in order to reach a certain target point. We simultaneously optimize 10 rockets flying in parallel. As the forward model is solved with a finite number of time steps, a small change does not necessarily translate to a different outcome, leading to zero gradients almost everywhere.

GRAVITY This task runs a physics solver to infer the collision behaviour of three cubes that are dropped onto each other. If optimized correctly, the cubes will form a tower after being dropped. We optimize the two upper cubes' initial translation and their coefficients of restitution, the "bounciness", which we provide as the input to the solver.

5.4.4 Results

We display the results of all methods listed in Tab. 5.2 in Fig. 5.6, with one subfigure per task, and show a quantitative analysis in the right part of Tab. 5.2. We group the results into rendering (CORNELL BOX, BRDF, MOSAIC), modeling (WICKER, NODE-GRAPH, LED) and animation (ROCKET, GRAVITY), which correspond to the three rightmost columns in Tab. 5.2 and the rows in Fig. 5.6, respectively.

From the convergence plots in Fig. 5.6, it becomes evident that `NoLocal` works in select cases, but often struggles to find the correct solution, especially where the domain is higher-dimensional (e.g., MOSAIC), as a uniform random sampling of the parameter space introduces substantial variance in the gradient estimates. Similarly,

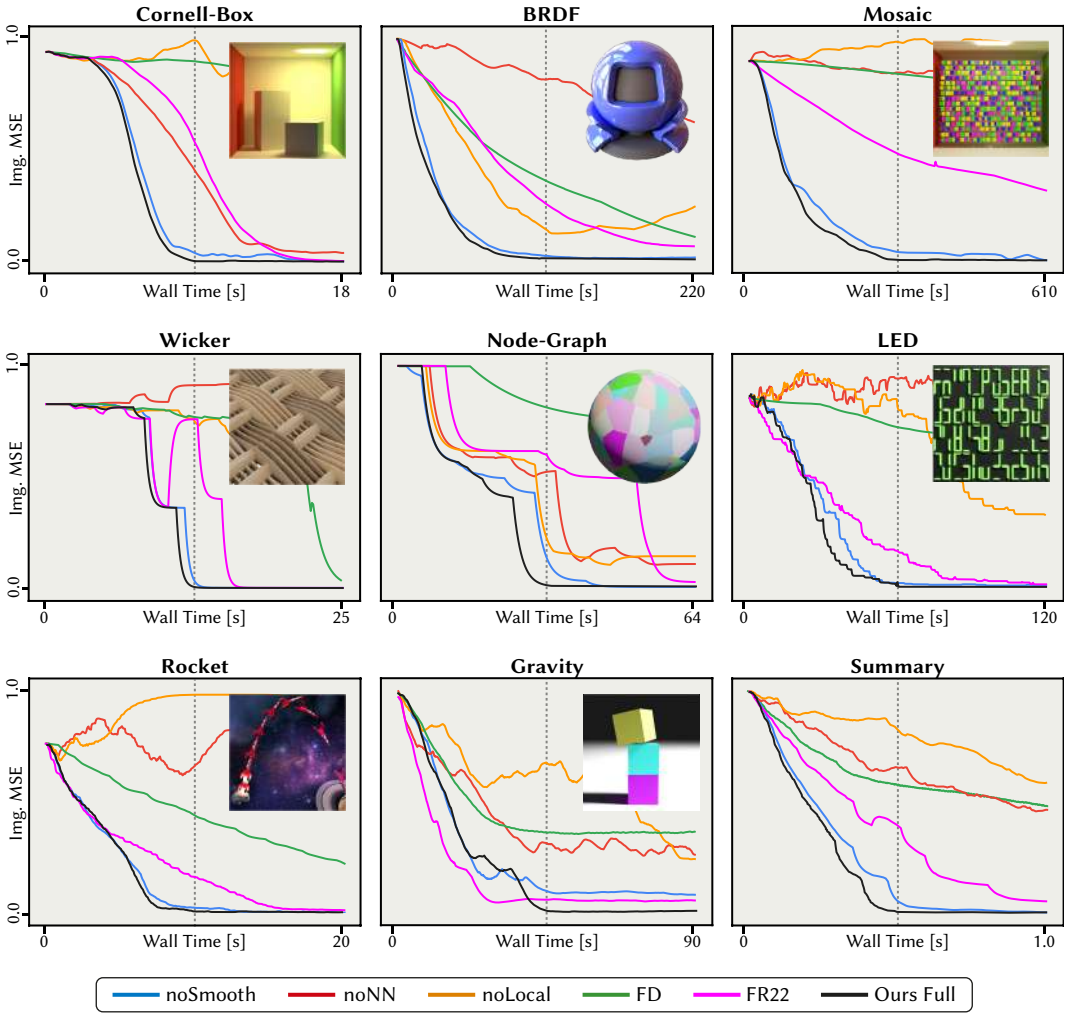


Figure 5.6: We show convergence plots of all methods (wall-clock time in seconds, method colors are consistent with Tab. 5.2) for all tasks, ranging from differentiable rendering (top row) over procedural modelling (middle row) to animation and simulation (bottom row). For all experiments, we let our method run until convergence (the dashed vertical line in each subfigure) and then allocate *twice as much* time for the other methods to converge. All results are reported across an ensemble of 10 independent runs for all methods. For convenience, we show a summary across all tasks in the right bottom subfigure (mean across all methods, normalized and resampled). For task-specific visualizations, please see Fig. 5.7.

NoNN is challenged in higher-dimensional cases and does not converge reliably, which we attribute to the reduced expressiveness of the quadratic potential. However, this shows that for simple tasks (e.g., CORNELL-BOX), the proxy does not need to be overly complicated. Finite Differences (FD) works well and makes steady progress towards the target, but does not scale well to higher dimensions, as an n-dimensional

problem requires $2n$ function evaluations for a single gradient step (see the wall-time plots in Fig. 5.6 and Tab. 5.1). `FR22` works reliably on all tasks, but often converges slower than our method. Surprisingly, we find that ablating the inner smoothing

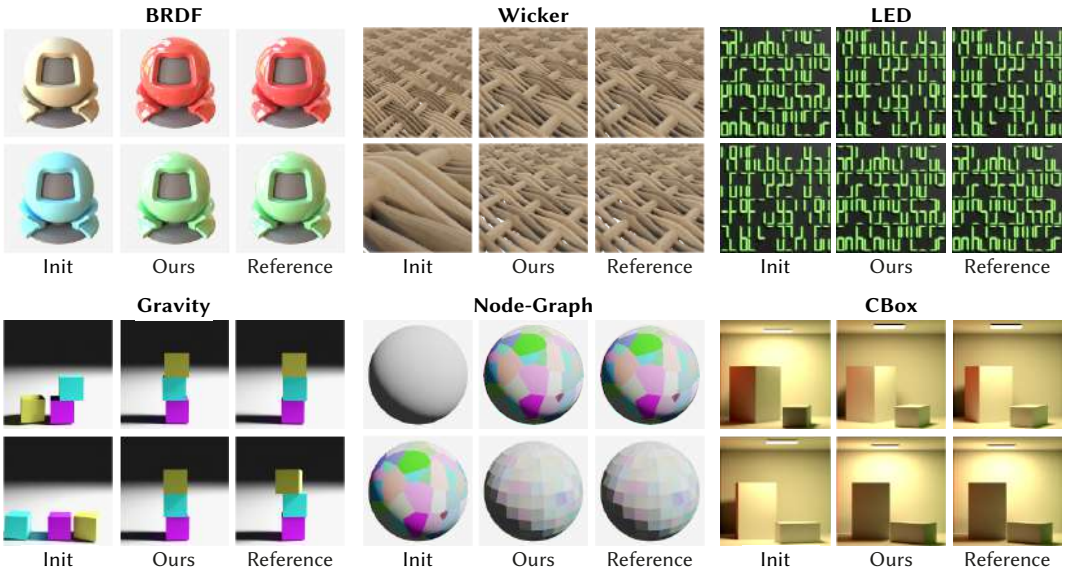


Figure 5.7: Visualizations of instances of our tasks. We visualize the initial configuration in the left column of each subfigure, and our outcome and the reference in the middle and right column, respectively. For our method's convergence behaviour on these tasks, please see Fig. 5.6.

operation in `NoSmooth` only slightly impedes performance (ca. 2x), which we partly attribute to the implicit smoothing introduced by the surrogate fit. In almost all cases, `Ours` works best and faithfully recovers the true parameters. The overhead of our method compared to its competitors is small: for smoothing, it suffices to slightly perturb the current state, i.e., no additional evaluation of f is required. The NN query is very efficient as it can be parallelized on the GPU, and the surrogates are relatively shallow (for implementation details see Appendix C.1), providing `Ours` with the best quality-speed relation, as is evident from the right bottom subfigure in Fig. 5.6, where we show the (normalized and re-sampled) mean performance of all methods.

5.4.5 Higher Dimensions

We here evaluate our method on higher-dimensional problems from the inverse rendering literature and compare our approach against the established derivative-free optimizers genetic algorithms (`GA`) [116], simulated annealing (`SA`) [152],

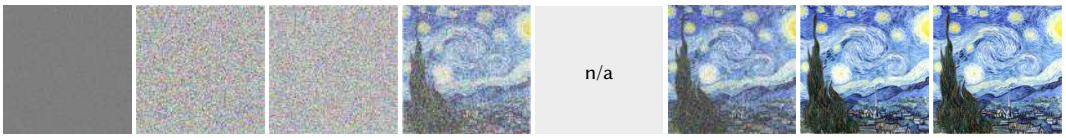


Figure 5.8: We show an equal-sample comparison (i.e., the same budget of function evaluations) for the task of optimizing a $256 \times 256 \times 3$ texture. CMA-ES cannot be run on this example due to its quadratic memory complexity causing out-of-memory errors on our 64 GB RAM machine.

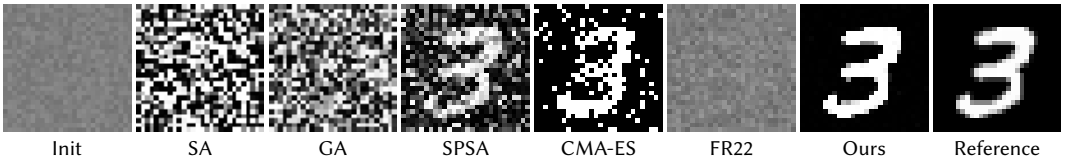


Figure 5.9: We show an equal-sample comparison for the task of optimizing the $35,152$ weights of a MLP such that it encodes digits from MNIST [162].

simultaneous perturbation stochastic approximation (`SPSA`) [283] and `CMA-ES`.

Due to the high dimensionality of these experiments, we found it necessary to increase the surrogate capacity and the gradient batchsize (for implementation details, please see Appendix C.1). All results we show are equal-sample comparisons, i.e., achieved after the same number of function evaluations, disregarding the fact that `CMA` requires significantly (multiple times) more runtime than all other approaches. To avoid clutter in the main manuscript, we show the outcome of our ablated methods on theses tasks in Appendix Fig. C.2.

TEXTURE First, we optimize the 256×256 RGB pixels of a texture in Fig. 5.8,

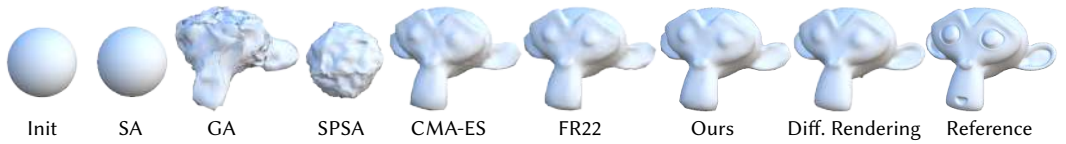


Figure 5.10: We show an equal-sample comparison for the task of optimizing the 3D positions of a tessellated sphere with $2,562$ vertices to match a rendered reference shape. "Diff. Rendering" uses the analytical gradients from Nicolet et al. [215].

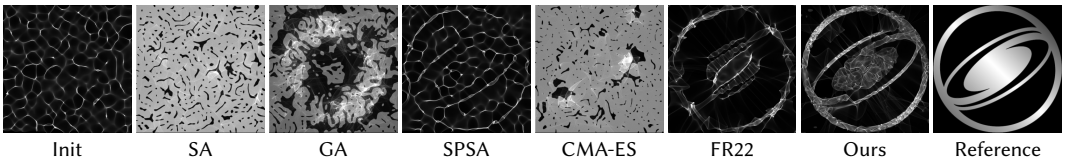


Figure 5.11: We show an equal-sample comparison for the task of optimizing a 1,024-dim. heightfield such that the resulting caustic resembles the reference image.

a relatively simple task with a smooth cost landscape and no correlation between the optimization variables. Our surrogate gradients lead to a successful optimization outcome, while the derivative-free optimizers GA and SA fail to converge due to the high problem dimensionality. CMA cannot be run on our 64GB RAM machine because of the quadratic memory requirements of the covariance matrix. Both SPSA and FR22 make progress towards the target, but require more time to converge.

MLP To increase the correlation between the optimization variables from the previous task, we repeat a similar experiment in Fig. 5.9, where we use our method to optimize the weights of a MLP such that it overfits single digits from the MNIST [162] dataset, i.e., learns a mapping from continuous 2D coordinates in (0,1) to a monochrome color value at the corresponding pixel location. The results are similar: our method has already converged, while SPSA and CMA make progress but require more function evaluations, and both GA and SA do not converge at all. Interestingly, FR22 does not converge either. This is in line with recent findings that show that perturbation-based methods do not perform well on emulating backpropagation in neural networks [209; 32; 19].

Figure 5.12: We sample the latent space of our trained variational auto-encoder (VAE) and show a variety of style transformations (rows), enabled by the spline formulation, on three digits per MNIST class. The first row displays the output of the spline renderer on which the surrogate operates.

MESH Next, we optimize the 3D position of 2,562 vertices of a triangle mesh, as in Nicolet et al. [215], to match a reference (Fig. 5.10). This problem is already

much harder, as the vertices are interlinked and the loss landscape exhibits discontinuities due to the rasterization process. On this task, `GA` and `SPSA` fail to converge to the correct result, while `SA` does not move from the initial configuration. In contrast, `CMA`, `FR22` and `Ours` find the correct solution, with `FR22` and `Ours` achieving the lowest final optimization errors (0.0018 and 0.0013, respectively). For completeness, we also show the differentiable rendering solution proposed by [215].

CAUSTIC In caustic optimization, a classic task from inverse rendering [225; 272; 153], the goal is to optimize an initial height-field such that it refracts incoming light into a caustic that resembles a provided reference image. While previous systems have gone to great effort to accurately capture the intricate behaviour of the refracted light, we use a simple rasterization-based renderer inspired by [320] (for details see Appendix C.2). Our heightfield is parameterized by a cubic B-Spline with resolution $1,024$. This example is interesting as the optimization variables have a highly non-local influence on the observed image pixels. We show the resulting optimization outcomes in Fig. 5.11 and observe that all traditional gradient-free optimizers fail to correctly recover the target image. Again, `SPSA` and `FR22` achieve a caustic that roughly resembles the reference, while our method achieves a plausible outcome.

SPLINE GENERATION Finally, we use our method to train a *generative model* on a *non-differentiable* task. Here, we use our surrogate gradients to train a VAE [148] that encodes digits from the MNIST [162] dataset and outputs the support points of a spline curve, which are then rendered with a non-differentiable spline renderer. As in all tasks, we operate in *image space* only, so we cannot simply backpropagate through the spline renderer, but must query our surrogate for gradients.

This again is a very high-dimensional, non-local and interlinked problem, as all optimization variables (the NN weights) directly influence the spline's final support points. For details on the training and model architecture, please see Appendix C.2 In Fig. 5.12, we sample the latent space of the model after training. We render the generated spline in different styles, which can easily be applied in post-processing due to the control-point formulation. As is evident from the figure, our method is the

only approach that achieves an output that resembles actual digits across all numbers, with FR22 achieving satisfactory results on simple cases (1, 3, 5, 7), and the other derivative-free optimizers failing completely. To our knowledge, this is the first generative model that is trained on a non-differentiable task, which again highlights the generality of our proposed approach, ZeroGrads, and gives rise to an exciting avenue of future research.

5.4.6 Gradient Variance Analysis

While our method works well in all the previous tasks, its benefits are most pronounced when the loss landscape exhibits stochasticity or noise, e.g., in the MLP and SPLINES tasks. We hypothesize that this can be explained by the centerpiece of our approach, the *neural proxy*: in contrast to FR22 and SPSA, ZeroGrads uses a neural network as proxy function, whose state acts as hysteresis and endows our method with a certain inertia, limiting the estimated loss landscape's spatiotemporal change by the network's adaptability. This behaviour is further reinforced by the spectral bias of neural networks [248; 295], which has been shown to encourage the learning of low-frequent, Lipschitz-continuous functions over those characterized by rapid changes.

FR22 and SPSA, in contrast, re-build a (linear) gradient estimate during every iteration of the optimization, effectively ignoring information about the loss landscape from previous iterations. As this gradient estimate is a stochastic approximation, it will exhibit noise and variance, which highlights the main difference between the discussed (gradient-based) approaches: while SPSA and FR22 estimate the *parameter gradient* ∂_θ (subject to variance), ZeroGrads estimates the surrogate gradient ∂_ϕ, but *analytically* computes the parameter gradient ∂_θ. This allows us to move the higher-variance estimate into the neural network's parameter update, where the estimate's noise is smoothed by the aforementioned hysteresis.

To analyze this behaviour, we plot the variance of the gradient-magnitude over the course of the optimization in Fig. 5.13 for all three[1] gradient-based optimizers

[1]We exclude finite differences from this comparison due to its intractable per-iteration cost in higher dimensions.

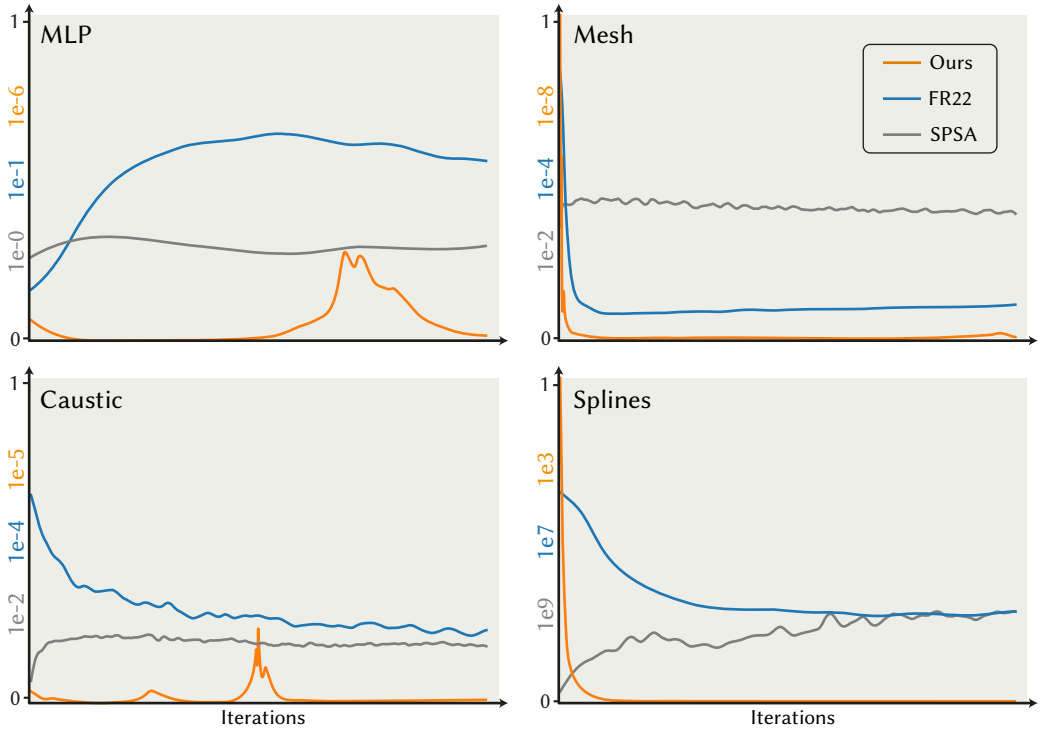


Figure 5.13: We plot the variance (smoothed for ease of visualization) of the gradient magnitude over the course of the optimization. Note that the scales are vastly different, as denoted on the left axis. Our method consistently produces gradients with lower-variance magnitude, which we attribute to the neural proxy's state and smoothness, resulting in less gradient noise.

`SPSA`, `FR22` and `Ours`. The vastly different scales on the y-axes of each subplot confirm our hypothesis: the variance in the gradient-magnitude of our method is consistently orders of magnitude lower than that of the other approaches. While this does not allow reasoning about the *correctness* of the derived gradients, it explains why our approach outperforms the competitors in the provided examples.

5.4.7 Comparison to specific solutions

There exist many specialized solutions that enable gradient computation in graphics, and a full study of *all* is beyond the scope of this work. We compare qualitatively to two of these methods, rendering (Fig. 5.14) and procedural modelling (Fig. 5.15), where the common theme is that our neural surrogates are capable of optimizing their specific problems as well, and sometimes even go beyond. In Fig. 5.14, top row, we show that we can optimize a material's IOR, a feature for which backpropagation through detached sampling has not yet been implemented in Mitsuba. As our method

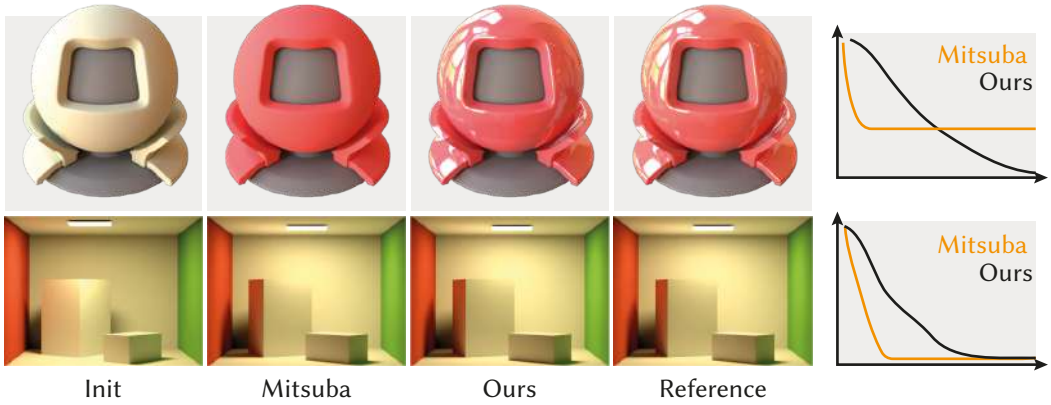


Figure 5.14: Comparison with Mitsuba 3 [130] on the BRDF and CORNELL-BOX tasks, top and bottom row, respectively. Note that the incorrect IOR in the top row is due to Mitsuba not yet implementing this feature instead of failing during optimization (see Sec. 5.4.7).

only needs a forward-model, we can simply combine Mitsuba's forward path tracer with our surrogate gradients and thus are able to optimize the IOR as well. In Fig. 5.15, we optimize a node graph towards the target patterns, using a mixture of VGG and MSE loss, which nicely shows our surrogate's flexibility w.r.t. to the objective f. Moreover, our method also works in extreme parameter ranges, as is evident from the bottom row in Fig. 5.15, where the pre-trained proxy from Hu et al. [121] breaks due to the parameter value being out of the range it encountered during training. In summary, although our method might sometimes come at the expense of higher compute or variance (e.g., compared to Mitsuba in Fig. 5.14, see the convergence plots to the right), the strength of our approach lies in its generality, i.e., in that it can be applied to arbitrary forward graphic models, and that it successfully optimizes high-dimensional, interlinked problems.

5.4.8 Limitations and Failure Cases

Our method inherits the limitations of gradient descent, namely that it can get stuck in local minima (although we do our best to avoid this via the smoothing convolution), move slowly in regions of shallow slope (see the experiments on the Rosenbrock function in Appendix Fig. C.1) and that it introduces additional hyper-parameters (Appendix C.1.1). Additionally, our method "wastes" samples during an

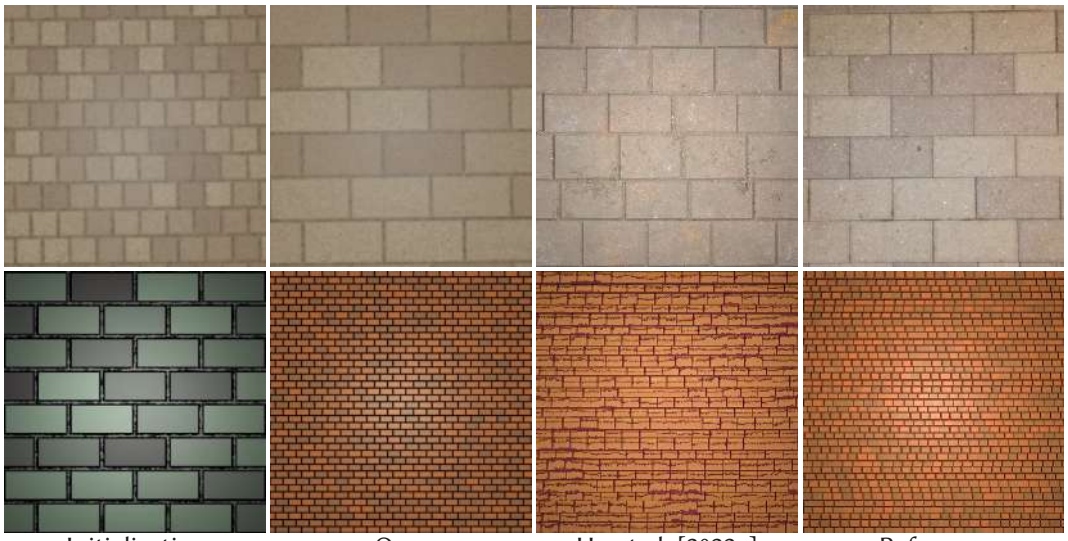


Figure 5.15: Comparison between Ours and Hu et al. [121] (result and reference taken from their publication). In the lower row, their pre-trained brick generator fails, as the parameter lies outside the training domain.

initial warm-up phase, in which the (initially random) network weights first adapt to the loss landscape. Moreover, while our derived gradients have lower variance than competing approaches (Fig. 5.13), they have higher variance than analytical gradients and therefore typically under-perform relative to problem-specific methods, where they are available (see the convergence plots in Fig. 5.14). Finally, on lower-dimensional discrete problems, it can potentially be faster to simply brute-force the solution by trying all possible combinations, akin to what genetic algorithms would do with a high-enough sample budget. However, this quickly becomes infeasible as dimensionality increases.

In addition, we show a failure case in Fig. 5.16. The task is inspired by PSDR-Room [323; 211] and the optimizer is asked to replicate a scene layout and materials from a single photograph. For each piece of furniture or shrubbery, the optimizer can make a discrete choice from 10 objects (left column in Fig. 5.16) and additionally adapt their continuous 3D position in the scene and the wall's color (right column in Fig. 5.16). We optimize MSE in the single-stage, single-resolution setting. While successful in the type-only setting, ZeroGrads fails to correctly optimize both type and position in the right column of Fig. 5.16. We assume this is because the optimizer must cycle through a number of objects before encountering the correct one, while

Figure 5.16: A limitation of our method: when the loss landscape is too complex (right column: mixture of plateaus and discrete spaces), the proxy cannot encode it accurately and the optimization stalls in a local minimum.

simultaneously working in discrete and continuous space, and moreover dealing with plateaus in image-space which stem from the objects not overlapping their reference counterparts. Not even the smoothing operation can make this task easier, as the optimizer has the opportunity to reduce the image error by simply pushing the objects out-of-frame (as is happening here) and then falls into a local minimum from which it cannot recover. We conclude that more research is needed in this direction, e.g., through self-adapting proxy configurations or advanced hybrid approaches.

5.5 Conclusion

We have proposed ZeroGrads, a method for practical computation of surrogate gradients in non-differentiable black-box pipelines, as are found across many graphics domains, ranging from rendering over modelling to animation. Our key ideas are the

smoothing of the loss landscape, a local approximation thereof by a NN, and a low-variance estimator based on sparse, local sampling. We have favourably compared to several ablations and published alternatives and shown results for a wide variety of tasks. Additionally, our neural surrogate allows us to transform the noisy gradient estimate into an update on the network's parameters, where the noise is smoothed by the network's hysteresis. We therefore can show that our surrogate gradients scale to high dimensionality, where traditional gradient-free optimization algorithms often do not converge.

In future work, we plan to further explore the interplay of the inner surrogate loss and the outer optimizer and to find ways to automatically determine the required surrogate network's complexity. Moreover, it would be interesting to leverage the fact that our surrogate provides a continuous model of the cost landscape, for instance by lookahead-training or approximate second-order methods.

Most of the things that enable our approach are known in the optimization community that routinely uses proxies and surrogates. Yet, these ideas are rarely used in graphics, where specific solutions were developed and rarely compared against what the optimization literature offers. Our work combines graphic-specific features (e.g., MC-estimating the gradient, sampling the objective through simply rendering) and graphics-inspired improvements (such as variance reduction through importance sampling) to match requirements of graphics with general optimization.

Chapter 6

Discussion and Outlook

The previous three chapters have presented the core topics of this thesis: efficient, inverse rendering and the estimation of the therefore necessary gradients. However, as initially remarked, the mere availability of an efficient optimizer or gradient does not guarantee a successful optimization outcome - in fact, as remarked by Metz et al. [192] and listed in Chapter 2, there is a plethora of reasons for which optimization still could diverge, such as incorrect or undefined gradients or plateaus in the cost landscape. This section will thus illustrate limitations of current techniques, present potential remedies and shine light on new directions and angles on inverse and differentiable rendering.

6.1 Limitations of the discussed methods

Hyperparameter sensitivity. For both gradient estimation approaches presented in this thesis, PRDPT and ZeroGrads, the hyperparameter σ, which denotes the standard deviation of the Gaussian sampling distribution, plays a crucial role. While correct choice of the sampling radius is also an issue for the more traditional optimizers finite differences and SPSA, for the case of PRDPT and ZeroGrads it is of particular importance since it additionally denotes the *blur* radius that is used to remove plateaus. Too little blurring will not sufficiently remove plateaus for convergence (since we will always sample parameters *on* the plateau), and too much blurring will lead to oversmoothing, which could potentially lead to divergence.

To exacerbate this problem, σ should ideally vary *per dimension*: consider the

case of optimizing the number of bricks in a brick texture alongside their color. The number is a discrete parameter in the interval $[1,k]$, with $k \gg 1$, while the RGB color is limited to the $[0,1]$ interval. Both PRDPT and ZeroGrads address this problem via normalization of the ranges, which however often is non-trivial and non-intuitive, such as in this case here. Even when normalization can be easily applied, there are no guarantees that equal perturbations along the dimensional axis result in equal image-space appearance variations: some parameters might be hidden by occluders and thus do not influence the image-space error at all, while others, such as light sources, will exhibit strong, non-local influence for even minor perturbations.

The optimal choice of perturbation radius, or per-dimension σ_i, thus is an open problem. A simple and effective solution is the one-at-a-time (OAT) method [44; 42], where the optimal per-dimension sampling radius is determined iteratively: vary σ_i by a small perturbation ε_i and see if the resulting image changes - if not, increase ε_i. However, this strategy, while effective, is not *efficient* – in fact, it is prohibitively expensive, since an n-dimensional problem will require *at least* $2n$ function executions to determine σ, and this is only the *precursor* to the gradient estimation step, which will require additional renderings. While it is not inconceivable that this might still pay off in total optimization time if the found "optimal" σ sufficiently improves the subsequent gradient estimate, this strategy seems naïve and provides an interesting starting point for future research, with links to parameter sensitivity analysis.

Gradient sparsity. An additional point of improvement lies in the way in which the perturbations are rendered from parameter- into image-space. There exist two possibilities: in the *per-image* variant, the entire image is rendered with a single perturbation, and several images are combined to achieve a blurred average image; in the *per-pixel* variant, only one image is rendered, but each pixel's color is rendered under a random perturbation of the optimization parameter (Fig. 6.1 shows an example using Gaussian perturbations), leading to a more evenly space-out parameter influence.

However, per-pixel variations are not always straightforward to implement and parallelize over – existing frameworks such as NVDiffRast or

Mitsuba assume a relatively static scene definition in their rendering calls. Both PRDPT and ZeroGrads therefore use *per-image* perturbations for ease of implementation and computational efficiency, which leads to competitive wall-time optimization performance. However, as becomes evident from the left column in Fig. 6.1, per-image perturbations lead to less well approximated blur (Gaussian here), and therefore to sparser gradients, which result in inferior optimization performance. Preliminary toy experiments conducted after completion of the work have confirmed this behaviour – however, it is unclear how per-pixel perturbations would be realizable for the arbitrary graphics models discussed in Chapter 5 due to their potential black-box nature. An efficient way of realizing such per-pixel perturbations on modern graphics pipelines would therefore benefit optimization convergence.

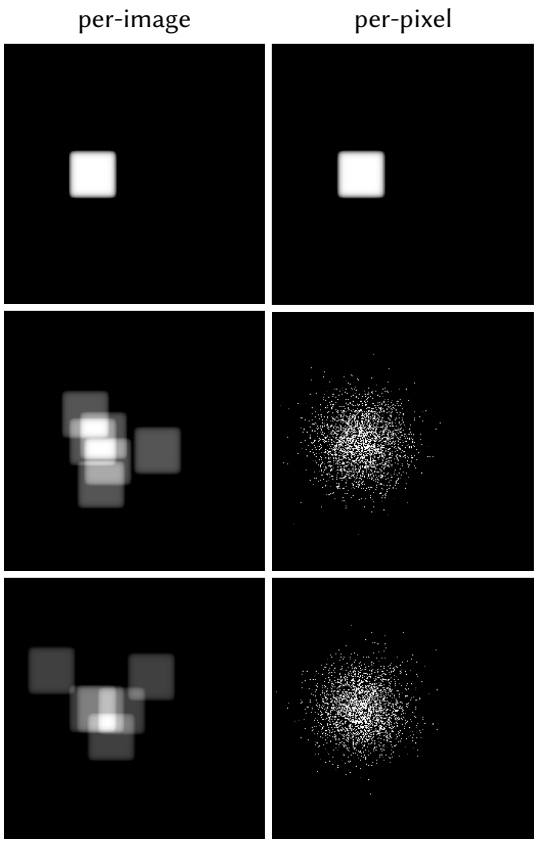


Figure 6.1: Different parameter perturbation techniques (columns) and random seeds (rows) for the square's initial 2D position (top row).

Uncertainty estimation. For all three presented approaches, the sampling patterns during training and optimization were chosen to either be random uniform across the training set (Chapter 3) or from the Gaussian distribution and its derivatives (Chapter 4, Chapter 5). The samples therefore were placed in space without taking into account how well the current parameters perform, and how much we know about their neighbouring regions. In a follow-up work to Metappearance, Liu et al. [174] show that one can meta-learn the sampling pattern during BRDF acquisition to improve the performance of a subsequent (neural) BRDF model by orders of magnitude. In a similar spirit, Ansari et al. [7] and Goli et al. [85] show that

incorporating sample uncertainty improves model estimation. It therefore stands to reason that one could incorporate the uncertainty of the current state estimate into the sampling pattern of a (variational) optimizer to improve convergence, by placing samples in (local) regions where the current fit deviates most from the underlying cost landscape, i.e., is not yet sufficiently accurate. Such sampling schemes could additionally help to mitigate the problems that are introduced in higher dimensions due to added noise and estimator variance.

Disentanglement of smoothing. In Chapter 5, our experiments have shown that we need to increase our surrogate's capacity for learning the loss landscape on higher dimensional problems. This is no surprise, as a bigger network can capture more intricate and higher-dimensional (non-linear) relationships. However, the current surrogate models have been hand-designed in an empirical trial-and-error process. While we have found configurations that work across problems (one for low- and one for high-dimensional problems, respectively), as of yet, there is no clear intuition on how complex the surrogate needs to be for an arbitrary new optimization problem. Additionally, this problem transcends mere fitting capacity and is interlinked with the employed smoothing: choosing a higher-capacity surrogate will lead to less low-pass filtering, resulting in higher frequencies being present in the learned loss landscape (the same goes for all frequency-increasing techniques, e.g., positional encoding [193] or Fourier Features [295]), hence diminishing the smoothing properties of the network and in turn requiring more Gaussian smoothing. While research from core machine learning has investigated the *theoretical* required network capacity for a certain fitting quality [229; 274], it is unclear how one would disentangle the *actually achieved* smoothing from the smoothing via Gaussian convolution. Disentangling these properties could be interesting for the broader community and also help with the aforementioned sensitivity to the blur radius σ.

6.2 Limitations of current inverse rendering setups

Pixel-wise loss functions. A problem that transcends the scope of this thesis and is commonly found in inverse rendering setups is the use of pixel-wise loss functions

such as the mean absolute or square error. In fact, the use of these losses is what creates plateaus in the cost landscape and motivated Chapter 4 in the first place. Since they compare the current state and the reference pixel-wise, they are unable to capture long-range spatial dependencies, necessitating the use of plateau-reduction or blurring techniques. As we have seen, plateau-reduction can either operate in image-space, using, e.g., image blur kernels or optical flow [322], or in parameter-space (Chapter 4, Chapter 5, [47]), often at the expense of additional renderings.

An additional, little-discussed remedy to plateaus is the optimization in a different basis. Instead of optimizing in pixel-space, we could optimize in a novel space that is free from plateaus. One idea would be the activation space of a neural network, which is by design smooth. In this setting, the network consumes the current image and the reference, and the optimizer would act based on the difference between the produced features. The challenge is to find a latent- or feature-space that accurately reflects small details (e.g., the pixels of a texture) without over-smoothing, while still being free of plateaus. A promising direction could be the attention-values computed in vision transformers (ViTs), which have been shown to correlate well with semantic- and positional image information [289] and which, by design of the attention mechanism, capture global image information [303] as well as high-level features, suitable for, e.g., material selection [93; 69; 273]. While in vanilla ViTs these features suffer from a lack of resolution due to the ViT's relatively coarse patchify-operation and input resolution, recent work has found ways to mitigate this via additional computations [5; 70] or architectural changes [267; 27].

A more fundamental problem with image-space approaches is that they can only optimize what is being rendered on the screen, and have no straightforward way of differentiating "hidden" parameters that do not influence the visible image, for instance due to occlusions. Extending such approaches to hidden parameters while retaining their smoothness would be an interesting direction for future research with significant impact potential.

Specialized optimizers. Additionally, little attention has been paid to using optimizers that are specifically designed for inverse rendering. The de-facto standard,

Adam [149], has been designed for neural network training – it is thus not obvious that it is also the optimal choice for inverse rendering. In fact, Ling et al. [173] point out one of Adam's shortcomings by showing that its per-parameter normalization does not preserve rotation equivariance. A promising research area would therefore be problem-dependent optimizers in inverse rendering that take advantage of their specific sub-problem's characteristics for improved convergence.

As an example, one could conceive an optimizer specifically designed for 3D Gaussian splatting (3DGS). In addition to regularization over time, as is currently done by Adam, such an optimizer could take the spatial correlation of the optimization variables (the Gaussians) into account when normalizing or propagating the gradients: Gaussians that lie closely together in 3D space will likely receive similar gradients, while additional spatial (gradient) filtering could be done post-projection in the 2D image plane. Similarly, the Gaussian's lower spherical harmonics (SH) bands, encoding non-local illumination, are not likely to change with high frequency; something the optimizer additionally could exploit. As proposed by Chandra et al. [34], one could even stack such optimizers, or meta-learn their free parameters over a range of 3DGS reconstruction tasks.

The above points, while only giving rough outlines of potential improvements, make it evident that current inverse rendering pipelines still show significant room for improvement. The following section will outline more exotic directions for inverse rendering which markedly deviate from the currently employed methods.

6.3 New directions for inverse rendering

Inverse rendering as RL problem. In reinforcement learning (RL), a learner (or agent) can interact with the world through a set of actions and observe the effect of these actions on its environment over time. The learner's goal is to take a sequence of actions which maximize an expected reward. Instead of the traditional, gradient-based optimization, we could frame iterative optimization as a RL problem by choosing the actions to be parameter updates and formulating the reward as the similarity between current rendering and target image. This formulation has

the significant advantage that in RL, the loss function or forward pipeline *need not be differentiable*. Instead, the so-called policy-gradient is formulated via the REINFORCE estimator [318], expressed as

$$\nabla_\theta \, \mathbb{E}_{\pi\sim P_\theta}[R(\pi)] = \mathbb{E}_{\pi\sim P_\theta}[(\nabla_\theta \log P_\theta(\pi)) \cdot R(\pi)] \tag{6.1}$$

$$\approx \frac{1}{N}\sum_i^N (\nabla_\theta \log P_\theta(\pi^i)) \cdot R(\pi^i)\,, \tag{6.2}$$

where θ is the optimization parameter, π denotes a trajectory of chosen actions over time, $R(\pi)$ is the total combined reward for this trajectory, and $P_\theta(\pi)$ is the trajectory's probability. Importantly, this formulation does not depend on the gradient of the reward function w.r.t. θ, which – in the case of differentiable rendering or a non-differentiable loss function – we might not be able to compute, but instead constructs an *unbiased* Monte Carlo estimator of the gradient via separate evaluation of the sampled trajectories π^i [318; 271][1].

Indeed, Pinto et al. [238] show that this trick can be leveraged to optimize vision models to improve on non-differentiable metrics such as average recall, while Li et al. [167] exploit the REINFORCE estimator to optimize an inverse procedural material model with a non-differentiable graph renderer in the pipeline.

The problem with RL-based approaches is their reliance on a fixed problem domain or parameter structure, for instance, the fixed number of graph parameters in [167] or the fixed number of NN weights in [238]. However, a perfect "differentiable rendering agent" would need to operate on unstructured, arbitrary domains and variable-length input data, e.g., the first three (of n) parameters could be the diffuse component of an optimized BRDF, the RGB emission values of a light source or the 3D position of a piece of furniture. Handling such domains in differentiable rendering is non-trivial; a potential solution could take inspiration from natural language processing (NLP), where RL has successfully been used [223] to fine-tune large transformer models (e.g., ChatGPT) which operate on similarly unstructured data and general-purpose domains.

[1] From Eq. 6.2, it becomes evident that the gradient estimator proposed in PRDPT (Chapter 4) can be interpreted as a special case of the REINFORCE estimator.

Oracle rendering. Extending this idea, another novel direction could be the use of a "differentiable rendering oracle" in the form of a large vision-language model (VLM) such as PaliGemma [23] or LLaVA [177]. This would allow combining image- and natural language information: the oracle would consume the current rendering and the reference image (or potentially just the difference image), as well as the current set of parameters and a prompt telling it to "vary the parameters to reduce the difference between the images". The big advantage of this proposal is that VLMs, by design, can work with the unstructured, variable-length input sequences encountered in general-purpose differentiable rendering. The language component of the input could additionally inform the VLM about the purpose of each parameter (e.g., "θ_0 is the intensity of the light source, $[\theta_1, \theta_2, \theta_3]$ its position, ..."). Additionally, the priors such models have learned during pre-training on gigantic datasets should be able to guide the optimization towards the correct parameters. It is further conceivable that such a model would have access to a rendering API, e.g., through Blender's Python interface, and therefore be able to self-improve by proposing a new set of parameters, rendering the image that corresponds to this new set of parameters, and then basing subsequent proposals on the quality or error of this current proposition. Finally, the use of VLMs as oracles would come with the additional benefit of the aforementioned smooth optimization space.

This idea could combine advances in NLP and autoregressive techniques, such as chain-of-thought prompting, with the graphics- and optimization community. It opens up several underlying research questions, such as which model would perform best on such a task, how the optimal prompt should be designed, whether one could fine-tune such a model for differentiable rendering, and what conditioning information would be most helpful.

Finally, both proposed ideas, RL and oracle rendering, could also be used in a hybrid scheme, where we either first use one of the techniques in a coarse pre-optimization to find a suitable initialization followed by precise, gradient-based methods, or alternate between the techniques and gradient-based methods in turns to progressively refine the initial solution.

Chapter 7

Conclusions

Fast training of deep graphics assets and successful optimization of inverse rendering scenarios are integral parts of modern computer graphics and the graphics research community. This thesis has presented three works that address commonly arising problems in these domains. The following short summaries will recapitulate on the respective works and outline their importance for real-world applications.

Metappearance

In Metappearance, we presented a novel training paradigm for visual appearance networks by using meta-learning to train fast, efficient and accurate neural representations. In a careful study of different training paradigms, we found our trained networks to perform on-par with "traditionally" trained, over-fit networks that take orders of magnitude more training iterations, while at the same time retaining the generality and inference speed of general networks. Metappearance further implicitly encourages data scarcity, allowing our models to perform at indistinguishable visual quality while using 99% less data, which could enable distributed architectures or streaming scenarios and subsequently enable the use of deep graphics assets in applications that involve user interaction and feedback. Moreover, the introduction of meta-learning as a form of "advanced optimization" into the graphics community could be applied to not only the (neural) model itself, but to arbitrary parts of the pipeline, as our follow-up work on meta-sampling [174] has shown.

Plateau-Reduced Differentiable Path Tracing

The subsequent chapter in his thesis addressed the problem of plateaus in inverse rendering. We show that our developed variational formulation can achieve convergence on problems with intricate, global light transport that previous methods did not converge upon. We find that constructing these examples is surprisingly straightforward, and they might very well occur in real-world inverse rendering scenarios, e.g., due to occlusion in autonomous driving and robotic vision, or discontinuous contacts in path planning and navigation scenarios. Our developed formulation is easily applicable to these scenarios, agnostic to the platform or underlying rendering algorithm, and requires only minimal modifications in form of a few lines of code. In follow-up work, this approach has been applied to other sub-fields in light transport research [47] and extended to second-order information [71].

ZeroGrads: Learned Local Surrogate Losses

In the third work presented in this thesis, we introduced the concept of a learned, local *neural surrogate* of the cost landscape, whose gradients can be used to drive an optimization scenario. Importantly, our algorithm does not make any assumptions about the underlying forward model or graphics pipeline; it instead supervises the surrogate learning via *point-samples* of the cost landscape, to which we fit a local neural network. This network is optimized on-the-fly, alongside the optimization parameter, and can easily be differentiated using readily available automatic differentiation libraries. Due to the network's smoothness prior, ZeroGrads exhibits significantly less noise than prior gradient estimation approaches and, as such, can be scaled to high-dimensional problems with well over 35,000 interlinked optimization variables. Since the current way of manually creating differentiable algorithms (i.e., re-writing entire programs, manually handling edge-cases, ...) is simply not scalable to arbitrary forward models, we believe our neural surrogates to be a valuable asset to the graphics community.

Conclusion

This thesis charts a trajectory starting with meta-learning neural networks to encode visual appearance in Chapter 3, with the introduction of Metappearance, which has served as inspiration for meta-learning a style-space for neural avatar heads [213]. Subsequently, shifting the focus from network optimization towards more general optimization techniques for inverse rendering, Chapter 4 present Plateau-Reduced Differentiable Path Tracing and introduced techniques for robust, variational optimization in inverse rendering. Finally, Chapter 5 presents ZeroGrads, a model capable of optimizing arbitrary, non-differentiable and potentially black-box forward models, which subsequent research has used for the discrete optimization of the biomimetic design space of winged seeds [160].

In summary, the overarching theme emerging from these chapters and their individual contributions is the growing synergy between machine learning techniques and physically-grounded graphics and their optimization. As these respective fields advance, research on how to further bridge the gap between data-driven and physically-grounded approaches, and on the limits of meta-learning and surrogate-based optimization in handling the complexity of real-world scenes will become ever more important. Additionally, research on how to interact with these processes – either through editability of the trained visual appearance networks or through user-control during optimization – will be an important future trend if we want to enable users to customize assets to their individual needs. Current trends – such as the integration of differentiable rendering and generative models, e.g., for generative 3D content creation, or advances in self-supervised learning for graphics and the push toward real-time optimization – underscore the relevance of the methods presented in this thesis. As generated neural assets and visual appearance networks continuously move towards production-readiness, the contributions of this thesis provide a foundation for future research at the intersection of learning, optimization, and graphics.

Appendix A

Appendix A: Metappearance

A.1 Meta-Learning

As slightly different variants of meta-learning are used for our different applications, we here detail their differences. We use different variants since, in order to compute the meta-gradients, one must backpropagate through backpropagation itself, which is a very compute-intensive process, as higher-order gradients (more specifically, the Hessian-vector product) must be calculated throughout the computation graph. To alleviate the computational burden this imposes, several MAML-variants that use first-order gradient approximations have been proposed. One of those algorithms that finds use in this work is first-order MAML (FOMAML) [63], which approximates higher-order gradients by replacing the Hessian with the identity-matrix and hence updates the meta-objective with the most recent inner-loop gradient, with significant savings on GPU memory and compute time. In practice, this means that, while MAML directly optimizes over the single gradient steps that are taken to reach a solution, FOMAML approximates this high-dimensional gradient trajectory with the local gradient of its last vertex. FOMAML has been shown to produce results close to MAML on certain applications [63; 214], which is commonly attributed to the fact that ReLU networks behave almost linear in high-dimensional spaces [87], which in turn implies that their derivatives do not carry much second-order gradient information and can be omitted without severe performance penalties. We also experimented with Reptile [214], but observed no performance improvements.

For the exact algorithm setup, please confer the following application subsections.

Note that using MAML is only compute- and memory-intensive during the meta-training phase: for inference, we run a mere gradient descent on the model parameters, and no additional overhead is incurred. We observed performance increase across all tasks and applications when also meta-learning the inner-loop learning rate, as proposed by Li et al. [172]. We implement our meta-networks and competitors in PyTorch [232] and Torchmeta [45].

Fig. A.1 shows the data splits used during each method's training. In `General`, test and train are split into disjoint sets, and the network generalizes over entire instances of the data. For `Overfit` and `Finetune`, the samples in a problem instance are split into disjoint sets, e.g., train- and test-angles of a BRDF. For `Meta`, we distinguish between a meta-train and a meta-test set (in the meta-learning literature, these are also called context- and target-set). During completion, the inner-loop samples from the meta-train-set, while its final performance *after* completion — on which the meta-gradients are calculated — is evaluated on data from the meta-test-set. However, this is *not* the data used to evaluate the final model at inference time. Instead, the same withheld test set as in `General` is used. We would like to emphasize that neither training nor meta-training sees test-data, ever, and we report all our experimental results exclusively on test data.

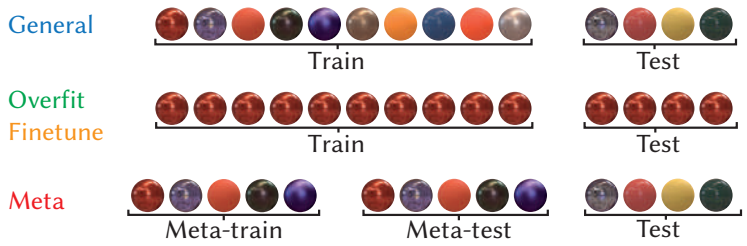


Figure A.1: Splitting of test and train for all four training types

A.2 Networks and Implementation Details

We summarize the exact hyperparameters and algorithm setups in Tab. A.1. The following sections shortly elaborate on the choice of models and approaches we use for the respective applications.

Table A.1: Algorithm setup and meta hyper-parameters for our experiments and the approaches we compare against.

	TEXTURES	BRDF	SVBRDFSTAT	SVBRDFNONSTAT	ILLUMINATION	TRANSPORT
Publication	Henzler et al. [108]	Sztrajman et al. [293]	Henzler et al. [108]	Deschaintre et al. [50]	Georgoulis et al. [83]	Zheng and Zwicker [335]
Type	CNN	MLP	CNN	CNN	CNN	MLP
Rendering	—	Mitsuba Jakob [129]	Cook-Torrance	Cook-Torrance	Blender	PBRT style
Input	RGB Image	MERL	Flash Image	RGB Image	RGB Image	PSS samples
Data	500 textures	Matusik [189]	Henzler et al. [108]	Deschaintre et al. [50]	Gardner et al. [79]	500 Cornell box scenes
Meta Algorithm	FOMAML	MAML	FOMAML	FOMAML	MAML	MAML
Gradient Order	First	Higher	First	First	Higher	Higher
Cosine Annealing	Yes	No	Yes	No	No	No
Meta-SGD	Yes	Yes	Yes	Yes	Yes	Yes
Meta-SGD Init.	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}
Meta-Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Meta-Optim. LR	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-5}	1×10^{-5}	1×10^{-4}
Weight Decay	—	1×10^{-6}	—	1×10^{-6}	1×10^{-6}	1×10^{-6}
Inner-Loop Steps	15	10	20	15	15	8
Meta-Batchsize	5	1	3	1	1	1
Meta Train Time	$\sim$ 2 days	$\sim$ 3 days	$\sim$ 4 days	$\sim$ 4 days	$\sim$ 2 days	$\sim$ 4 days
Meta Train Epo.	100,000	7.6×10^6	80,000	200,000	220,000	180,000
Time Meta-forw.	0.022	0.00117	0.0309	0.0158	0.01233	0.0296
Time Meta Step	0.06185	0.00309	0.0742s	0.0316	0.02566	0.0592
Time Meta-Inf.	0.6185	0.0309	1.484s	0.474	0.384	0.486
Iterations Overfit	1,000	83,000	5,000	2,000	5,000	8,600
Itera. Finetune	100	1,000	1,000	500	1,000	1,000
Batchsize	4	512	4	8	8	2,000
Latent Space Dim.	64	10	64	512	512	10
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Learning Rate	1×10^{-4}	5×10^{-4}	1×10^{-4}	1×10^{-5}	5×10^{-5}	1×10^{-4}
LR $\times$ Finetune	10	1	10	10	20	10
Weight Decay	1×10^{-5}	—	1×10^{-5}	—	—	—
Training Loss	L1 VGG	Log. MAE	Henzler et al. [108]	Deschaintre et al. [50]	MSE	FW KL Divergence

A.2.1 Textures

We use a U-Net [264] CNN with residual skip connections and AdaIN blocks [125]. As in [82], we optimize for the mean absolute error of VGG matrices between the reference exemplar and the network's output. The architecture is similar to Henzler et al. [108] and we refer to their publication for the exact network details. We use this architecture as our method `Overfit` and empirically determined that 1000 training iterations are sufficient to replicate most textures faithfully.

Method `General` preprends a ResNet-based encoder (ResNet-50, [104]) to the aforementioned U-Net in order to project the input image into a latent space, from where the previously mentioned U-Net decoder, conditioned on the latent code $z \in \mathbb{R}^{64}$, reconstructs the exemplar's features.

Finetuning has been applied in recent publications to steer the output of a

Table A.2: Different learning-rate and optimizer comparisons for the baseline methods we compare against. We choose the best-performing optimizer, respectively.

	0.1	0.01	0.001	1×10^{-4}	1×10^{-5}
`Finetune`, Adam	—	0.246	**0.205**	0.209	0.309
`Overfit`, Adam	—	—	0.188	**0.183**	0.598
`Finetune`, SGD	0.643	0.327	0.289	0.416	—
`Overfit`, SGD	0.287	0.259	0.515	—	—

general model towards more accurate representations [108; 52; 95]. Note that, for fine-tuning, we increase the learning rate by a factor of 10, as in Henzler et al. [108], to accelerate convergence, which makes `Finetune` a strong baseline.

For our `Meta`-method, we found a higher number of inner-loop steps to outperform the gains from second-order gradients, and hence use FOMAML with $k = 15$ inner-loop steps.

To show that we compare our meta-method against the best-configured competitors, we ran several experiments to empirically determine the best-suited optimizer and learning-rate setting. The results of these experiments are depicted in Tab. 2, while we show results on unseen test-data in Fig. A.2.

A.2.2 BRDFs

Much work was devoted to create, efficiently compress, interpolate and re-sample (spaces of) BRDFs; for a general survey we refer to Guarnera et al. [91]. Many applications revolve around representing a full BRDF in high quality from a small set of measurements. When these are taken in a suitable pattern [217], a linear basis found through principal component analysis (PCA) can be used as an encoding. Recently, Rainer et al. [249] showed that NN-based encoding of radiance data is a viable alternative to traditional, PCA-based methods. Moreover, Hu et al. [119] and Rainer et al. [250] encode multiple BRDFs and bidirectional texture functions (BTFs) in a single network, respectively. Finally, Sztrajman et al. [293] show that a compact two-layer MLP can learn the mapping between angular measurements and RGB reflectance and is able to faithfully reproduce BRDFs.

To tackle the task of BRDF reconstruction, we use a simple two-layer MLP

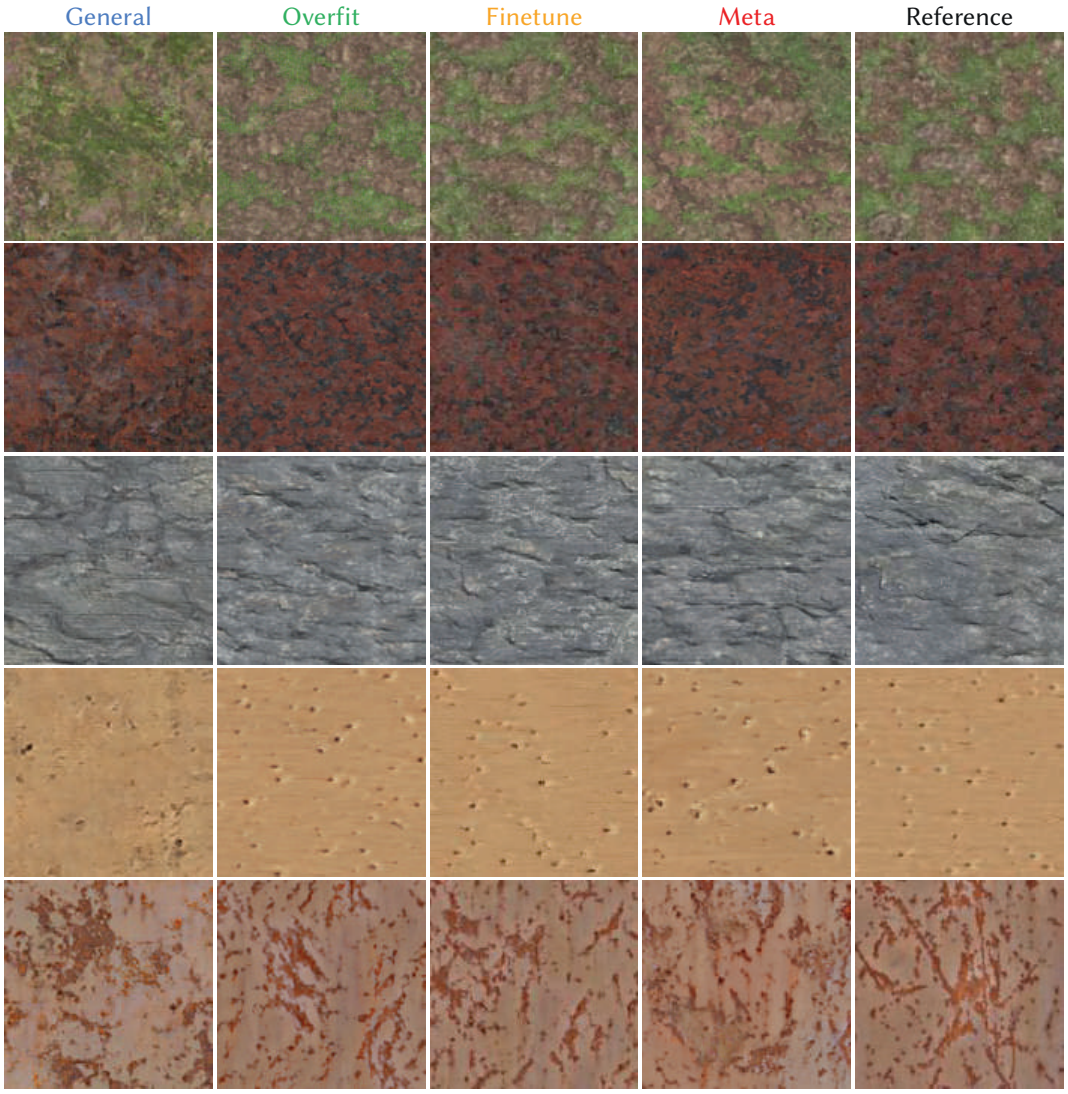


Figure A.2: Inference results for unseen test textures from the classes Grass, Marble, Rock, Wood, Rust (top to bottom).

with 21 neurons per hidden layer, as described by Sztrajman et al. [293], as method `Overfit`. The network receives randomly sample batches of light- and view-direction in Rusinkiewicz [266] parametrization and then outputs the logarithmic RGB reflectance for these query directions. We train our competitor `Overfit` with the official codebase from [293] with one network per MERL-material.

For `General`, we take inspiration from [119] and create a latent space of BRDFs by training a encoder CNN (for the exact details of the architecture we refer to their publication) that consumes the entire BRDF measurement of $180 \times 90 \times 90$ RGB triplets at once. For the encoder, we use the official code from Hu et al. [119],

which was kindly provided by the authors. To enable a fair comparison between all methods, we use the two-layer MLP of Sztrajman et al. [293] as a decoder. For `Finetune`, we fine-tune the output of `General` for 1000 iterations, which amounts to seeing each measurement in the data twice. We found that increasing the learning rate had no visible benefits and hence did not modify it.

Our `Meta`-method uses the same base architecture as `Overfit` and [293], a two-layer MLP with 21 neurons per hidden layer. We observed that incorporating second-order gradients into the optimization leads to more consistent results across all tasks and therefore use MAML with $k = 10$ inner-loop steps, which is made possible by the lightweight architecture (675 parameters in total). While using a meta-batchsize improved generalization for the previous task, we found that averaging the meta-gradients over batches here leads to a decrease in result quality and hence use a meta-batchsize of one. We use the classic 80%-20% train-test split for training, both for the MERL materials and the angular measurements within a MERL material, and show results on unseen test BRDFs in Fig. A.3.

A.2.3 Stationary svBRDFs

Classically, svBRDF were acquired by optimization [164] involving appropriate priors [56; 181; 207], optimization in a neural representation [3; 176] and finally methods that solve the task using a feed-forward network [108]. Optimization can be performed in the pixel- [3], or NN basis [108]. We again use an encoder-decoder approach to solve this problem, similar to the one motivated by Henzler et al. [108]. Their method uses an encoder-decoder architecture, where the encoder first projects a flash-illuminated image of a material into a latent space. The latent code is then used to condition the decoder, which is additionally provided with noise (for details, we refer to the publication) to then infer the svBRDF parameters. More specifically, the decoder outputs parameter maps of the Cook-Torrance [1982] reflectance model, i.e., diffuse and specular albedo, roughness and height, which is differentiated to a normal map. The generated svBRDF maps are then rendered by a differentiable renderer, assuming collocated camera and light, flat geometry and stationarity, as elaborated on by Aittala et al. [3].

Figure A.3: Inference results for different test BRDFs from the MERL database.

We implement method `General` with the public code provided by Henzler et al. [108], without fine-tuning, and train until convergence. For `Finetune`, we follow the method proposed in [108] and fine-tune the learned general decoder for 1000 epochs, with the learning rate increased by a factor of 10. For `Overfit`, we train the decoder of `General` only, as there is no need to span a latent space when overfitting only a single exemplar. We additionally let the model learn the rotation parameters of the flash highlight, which, surprisingly, does not always put the flash in the center. We attribute this to the fact that minimizing the error between gram matrices, which is one of the main parts of the training loss, matches exemplar statistics globally and discards information about relative spatial layout. Note that

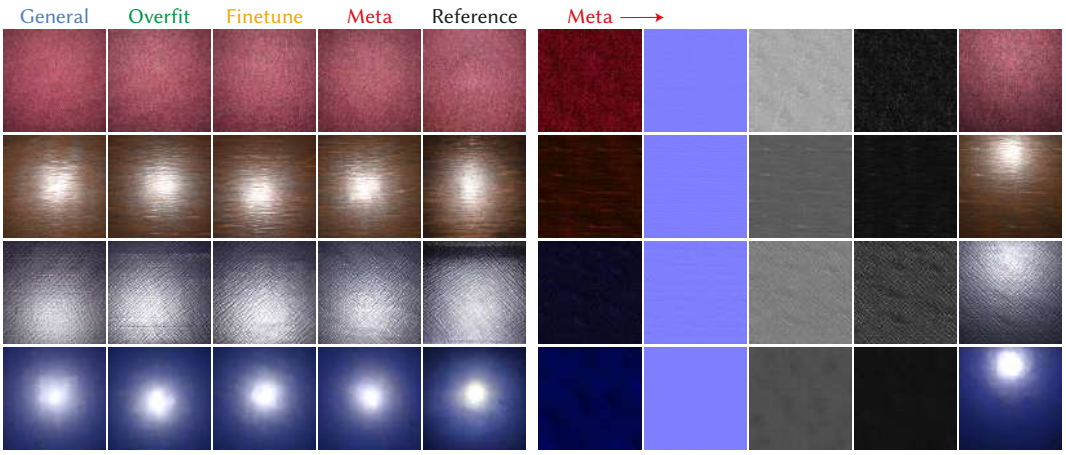


Figure A.4: Results for unseen test svBRDFs. On the right, we show the parameter maps produced by `Meta`: diffuse albedo, normals, roughness and specular albedo. We provide a top-light illuminated rendering with those maps in the rightmost column. We do not show the parameter maps of the other approaches for brevity and refer to [108]. Note that all maps must be free of baked shading, as they are stationary by construction.

the ratio of fine-tuning to over-fitting is 20%, which is remarkable.

Due to GPU memory constraints, our `Meta`-method for this application uses FOMAML and limits training to the decoder, with $k = 20$ gradient steps and a meta-batchsize of 3. We observed slightly improved performance when using latent codes generated by a pre-trained encoder over constant latents, which can be interpreted as a data pre-processing step. We also experimented with lower amounts of inner-loop steps, but the results were not convincing. We attribute this to the ambiguity and under-constrained nature of the problem of estimating svBRDFs from a single image: While it may be relatively simple for a model to quickly learn RGB colors or statistics alone, inversely solving the rendering operation for the svBRDF-maps that created said colors or statistics is a much harder task. We note that it is not uncommon to use a large number of inner-loop steps when meta-learning on ambiguous problems: Tancik et al. [296] use 64 steps on a Reptile-model to meta-learn an MLP that approximates neural radiance fields.

We show inference results for unseen test svBRDFs in Fig. A.4. We further show the parameter maps produced by our method and confirm that they are free of baked shading with a re-lighting. This can also be proven by construction, as only non-stationary parameter maps can bake non-stationary shading into albedo.

Stationary maps (like the ones used here) can, by construction, not bake-in non-stationary shading, as explained by Aittala et al. [3] and Henzler et al. [108]. Proving this, we have re-lit our results following the protocol in [108] and achieved relit-errors (VGG Gram L1, lower is better) of 0.25 / 0.10 / 0.12 / 0.16 for `General`, `Overfit`, `Finetune`, `Meta`, respectively.

A.2.4 Non-Stationary svBRDFs

To estimate non-stationary shading parameter maps, we use the approaches presented by Deschaintre et al. [50]. For the results produced by method `General`, we run their publicly available model implementation. For all other methods, we use a PyTorch-port of their original Tensorflow implementation. All methods use their proposed rendering-loss, which is crucial for accurate reconstruction. For `Overfit`, we run the network for 2000 iterations on randomly created scene configurations (as in the original publication, we use 3 diffuse and 6 specular scenes and aim to re-create all other settings as closely as possible). Method `Finetune` starts from `General`'s parameter maps and refines these, in equal fashion, for another 500 iterations. The ratio for fine-tuning to over-fitting thus is 25%, which is remarkable. We have experimented with different learning rates and, similar to our previous experiments (e.g., TEXTURE) found a learning rate multiplied by factor 10 to achieve fastest convergence. Even higher learning rates, e.g., 1×10^{-3}, lead the network to diverge and produce uniformly colored shading maps only. Lower learning rates, e.g., 5×10^{-5} require lots of training iterations (roughly 50% of `Overfit`) to achieve satisfactory performance, which somewhat defies the purpose of fine-tuning. `Meta` uses FOMAML as meta-learning algorithm, as the sheer size of the network (`General` has over 80 million trainable parameters) makes computing higher-order gradients through several inner-loop steps computationally intractable. We use $k = 15$ steps for the reported results. We alternatively experimented with running a lower-resolution version of the training process (128×128px) and a higher number of MAML inner-loop steps, but found this to de-stabilize training. For all methods, we train and evaluate on the publicly available data from Deschaintre et al. [50].

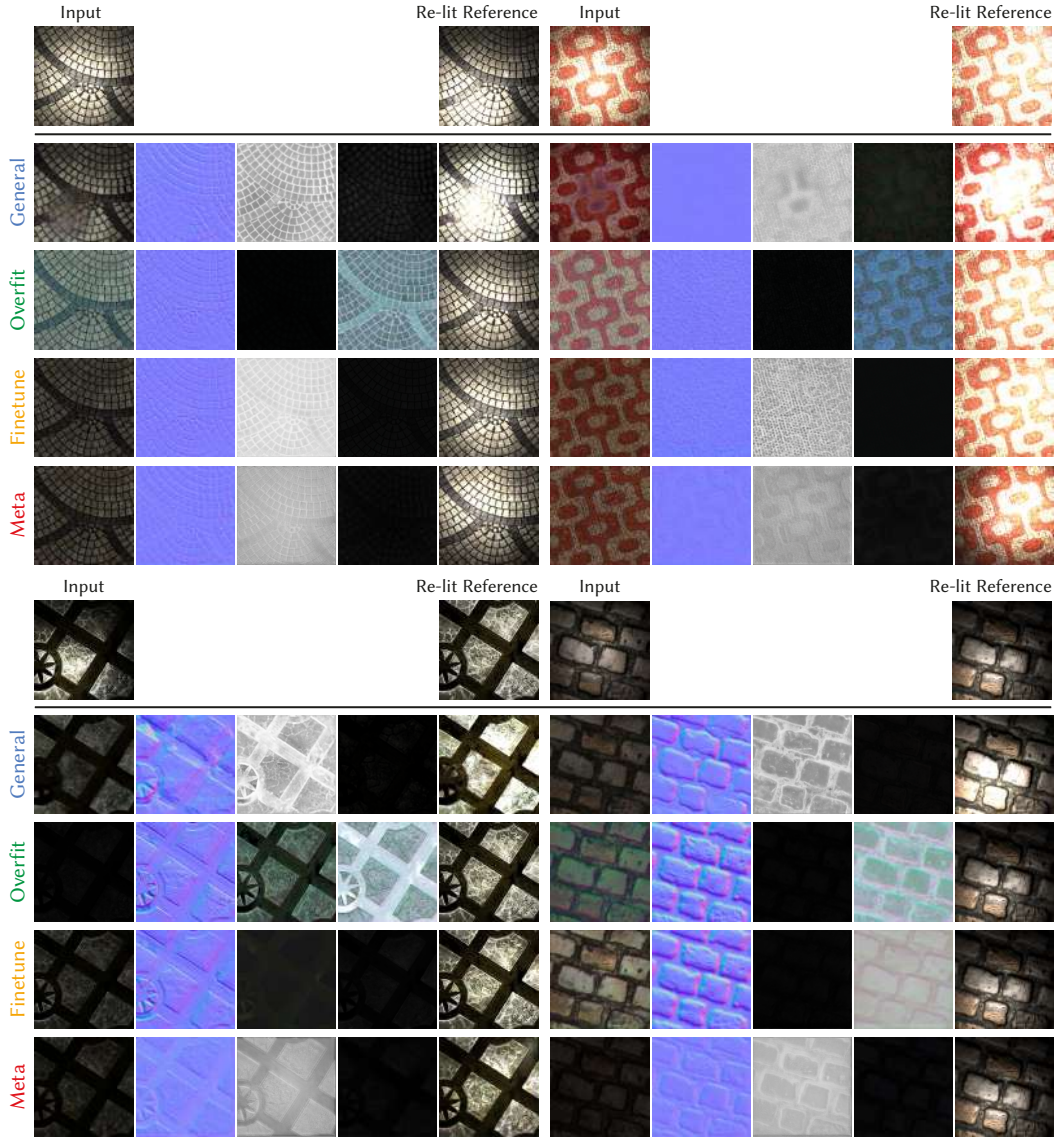


Figure A.5: Inference results for svBRDFs from [50]. We show the shading parameter maps diffuse albedo, normals, roughness and specular albedo and a re-lighting. Note how `Overfit` outputs a high specular albedo and a reduced diffuse albedo, something that (mostly) doesn't occur in other methods. This is because `Overfit` cannot benefit from priors over the reflectance data and hence only adjusts the maps to fit the rendering.

A.2.5 Illumination

Prior work has predicted parametric or general illumination using optimization [171; 227; 316], particularly indoors [282; 81; 80; 315] but also outdoors [114], and even as a volume [313; 284].

In order to meta-learn this task, we train on the Laval HDR Dataset [79], which provides a wide variety of illumination conditions. As the high spatial resolution

of the provided envmaps quickly makes computation intractable, we use a down-sampled version of the dataset at 32×64 pixels. Our architecture for encoding illumination is inspired by Rematas et al. [256] and similarly uses a U-Net-like encoder-decoder architecture with skip connections. Our encoder takes as input a 128×128 RGBN rendering of a sphere illuminated with the respective environment map. The down-branch extracts the image information through a cascade of 3×3 convolutions, all followed by ReLU activation and batch normalization (BN). As in Gao et al. [78] and Rematas et al. [256], we restrict the use of BN to the encoder part and find this to achieve higfher output fidelity than applying BN on the full architecture. To avoid checkerboard artefacts, we use bilinear upsampling in the decoder branch, followed by a zero-padded 3×3 ReLU-convolution, until we finally arrive at the original spatial resolution of 128×128, which we spatial-pool to the desired envmap resolution of 32×64. Additionally, we found it beneficial to let the network operate in log-space and also append a positional-encoded (6 encoding functions) coordinate grid to each input. We use this architecture for all methods.

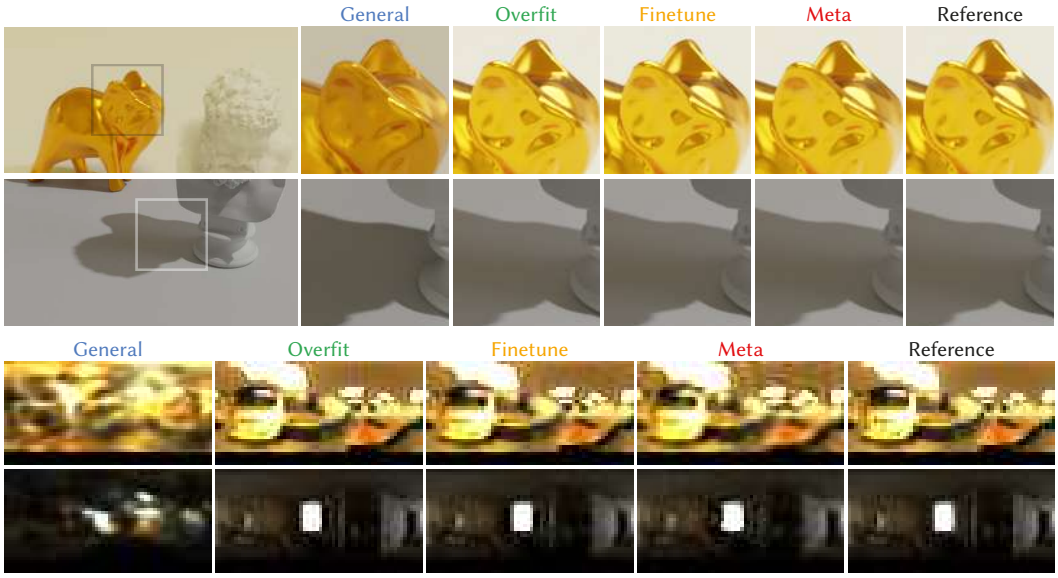


Figure A.6: Relighting results with unseen environment maps from the test-set. The top row shows re-renderings illuminated by the different method's outputs, which we display in the bottom row. Note how all methods achieve crisp, hard shadows, indicating that the high dynamic range of the illumination is matched well, but only optimization-based methods can regress fine nuances, such as the shading gradient in the bottom inset.

Similarly to the previous applications, we train `Overfit` on one problem in-

stance only, which it overfits in roughly 5,000 iterations. While over-fitting for more iterations is entirely possible and will result in a slight performance improvement, we found the performance-gain per increased time to diminish significantly after 5,000 iterations and empirically chose to stop the training then. Similarly, `Finetune` needs around 1,000 iterations to fully nudge the output of `General` to convergence. We experimented with several learning rate configurations for the fine-tuning experiment and chose to use the best-performing optimizer (Adam, learning rate 1×10^{-3}). Note that, as in previous experiments, we again increase the fine-tuning learning rate, which makes `Finetune` a strong baseline (please also cf. Tab. A.1). For `Meta`, we run MAML with $k = 15$ inner-loop steps. Interestingly, this application required us to change the outer-loop learning rate for `Meta` from 1×10^{-4}, as in previous experiments, to 1×10^{-5}, to stabilize training. We attribute this to the high dynamic range of the envmaps and the consequently high values of the gradients. The Meta-SGD init of 0.001 did not require changing.

A.2.6 Transport

To (meta-) learn the light transport in a scene, we use the method presented by Zheng and Zwicker [335], where a normalizing flow is used to warp the renderer's PSS in order to produce PSS samples that reduce rendering variance. In this scenario, the normalizing flow can be interpreted as an importance model that learns to produce samples proportional to the scene-dependent radiance, and can then, once training is finished, be sampled from.

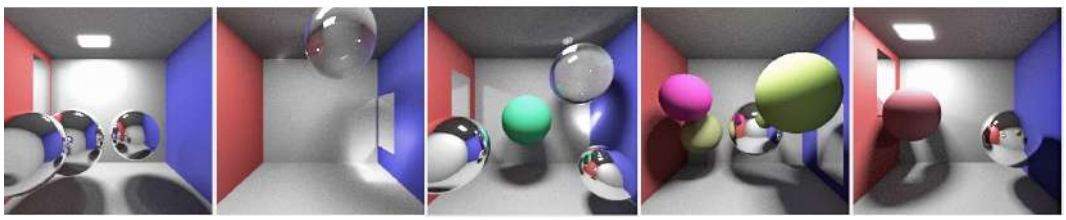


Figure A.7: Random samples from the Cornell box distribution.

We re-implement the normalizing flow architecture described in Zheng and Zwicker [335], which is a variant of RealNVP [54]. We use 8 coupling layers, each of which consists of two residual-MLPs with 40 neurons per layer and batch

Table A.3: Timing measurements for the TRANSPORT application in seconds.

	Regular	General	Overfit	Finetune	Meta
Prepare Rays	0.0s	4.1s	4.1s	4.1s	4.1s
Model Inference	0.0s	0.04s	362.2s	42.1s	0.5s
Trace Image	164.0s	164.0s	164.0s	164.0s	164.0s
Total	164.0s	168.1s	530.3s	210.2s	168.6s

normalization. As in Zheng and Zwicker [335], we pre-train the model to achieve the identity warp and use the resulting weights as initialization for all further experiments to speed up convergence (note that this pre-training cost is excluded from all timings we report). Moreover, we found it beneficial to include an additional ActNorm layer [150] and use it for all methods. The network is trained with batchsize 2000. To keep computation tractable, we limit PSS warping to $m = 2$, i.e., our model learns importance distributions for the first two bounces. For all subsequent bounces, we continue with randomly sampled rays. This is an established technique (cf. [335; 200]), as later bounces contribute less to the final render and hence are not as amenable to importance sampling.

For our experiments, we use a PBRT-style renderer written in C++ and integrated into Python via pybind11. At inference time, the renderer *consumes* PSS coordinates created by the trained normalizing flow, whereas, for the data creation process, it is also able to *return* the PSS coordinates that have been used to render the scene. To generate the data for the corresponding PSS warps, we create a set of Cornell-box-like scenes (Fig. A.7), with a random number of spheres (between 1 and 5) with random materials (diffuse color, reflective metal, refractive glass) and a random top-light configuration. We trace all these scenes at resolution 120×120 pixels (cf. Tab. A.3 for timing) to ease computation, but ask the reader to note that path-tracing with the final flow model can be carried out at any resolution. We store all paths traversed during rendering and subsequently re-sample those that carry high throughput to achieve approx. 20 *epp* (examples-per-pixel, a SPP metric that originates from omitting the pixel-filter and is effectively the average spp, cf. [335]), which leads to a 6-dimensional dataset per scene (recall that PSS dimensionality is

$2(m+1)$) that consists of $120 \times 120 \times 20$ samples.

For `Overfit`, we adhere to the training guidelines published in Zheng and Zwicker [335] and train each network for 60 epochs (this corresponds to approx. 8,600 gradient steps). As usual, over-fitting produces one network instantiation per scene. For `General`, we want to be able to train a network that generalizes across scenes. To allow this, we prepend the previously discussed flow model with a PointNet (PN) [245] -like encoder that consumes the *already resampled* dataset. Using the resampled PSS coordinates as input effectively allows the encoder to focus on encoding and out-sources the task of deciding which samples are important to a pre-processing stage that is equivalent for all methods. Our PN encoder uses three linear layers with 64, 128 and 512 neurons, respectively, batch normalization and pReLU activation units, and outputs a latent code on which we then condition the flow by concatenation. As in our previous experiments, `Finetune` again starts from the output of `General` and re-fines the estimated density for a total of 1,000 gradient steps. `Meta` is trained with eight MAML inner-loop steps and consumes batches of size 10,000. We found the higher batchsize necessary to stabilize meta-training with a higher number of inner-loop steps. Note that even after all inner-loop steps have been completed, `Meta` still has seen much fewer data samples (`Meta`: $8 \times 10{,}000 = 80{,}000$) than its competitors, that are presented with the entire dataset of approx. 288,000 samples.

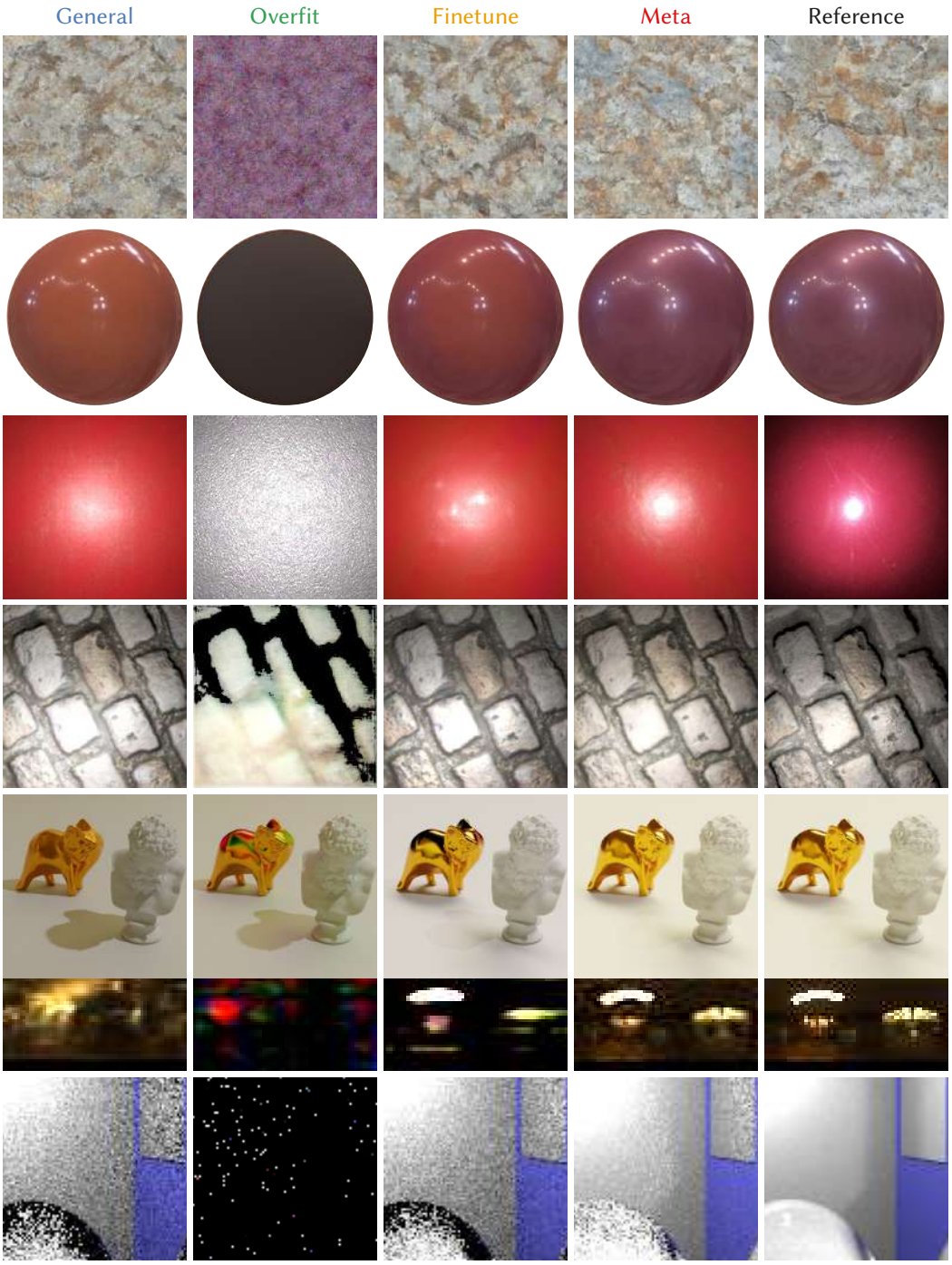


Figure A.8: Equal-time comparisons on unseen data from the test set for each application (top to bottom row: TEXTURE, BRDF, SVBRDF, SVBRDFNONSTAT, ILLUMINATION including the regressed envmap, TRANSPORT). Methods `Overfit` and `Finetune` are ran with the same amount of gradient steps `Meta` uses, i.e., the same wall-clock time. As `Overfit` starts the training from scratch, it cannot move from the (random) model initialization in such a low number of gradient steps, which is why its output is unsatisfactory. `General` does not change during inference. `Finetune` moves the output of `General` towards the reference, but cannot achieve good quality in such few optimization steps. `Meta` encodes the reference best across all applications.

Appendix B

Appendix: Plateau-reduced Differentiable Path Tracing

This supplemental contains the hyperparameters we used for our experiments (Sec. B.1), including an additional analysis of our two main parameters N and σ (Sec. B.2), experiments on compatibility with plateau-free problems and other renderers (Sec. B.3) and the derivations of the equations presented in the main text (Sec. B.4).

B.1 Hyperparameters

Hyperparameters: Tab. B.1 shows all the hyperparameters we use for our main experiments for all tasks. The first two columns are hyperparameters of our approach: N is the number of samples we use during an optimization iteration (for an analysis, cf. Fig. B.2 left), and σ_0 is the kernel spread with which we start the optimization (for an analysis, cf. Fig. B.3). spp is the rendering setting we use for rendering with Mitsuba, which we generally did not tune and hence do not regard as a hyperparameter of our method, but set such that the noise is less than the signal we want to optimize. We use the same spp across all path-tracing

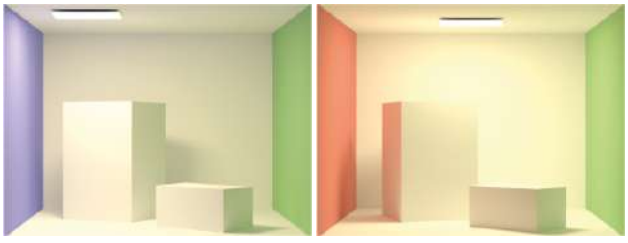


Figure B.1: The full view of the GI task.

methods. LR is the optimizer's learning rate (we use Adam with default parameters) and the last column shows the number of optimization iterations we run. We warm-start our σ annealing schedule after approx. 50% of the optimization and use $\sigma_m = 0.01$ for all experiments as the lowest value we decrease σ to during the annealing schedule, in order to avoid numerical instabilities.

Table B.1: Experiment parameters (columns) for all tasks (rows).

	N	σ_0	spp	LR	Iter.
CUP	2	0.250	16	0.01	400
SHAD.	2	0.500	32	0.02	400
OCCL.	2	0.800	32	0.02	600
GI	4	0.125	16	0.05	500
SORT	16	0.500	32	0.01	4000
CAUST.	4	0.125	32	0.01	500

Average spread: We additionally re-ran all experiments where $\sigma_0 \neq 0.5$ with the average kernel spread of $\sigma_0 = 0.5$ and show the optimization outcome in Tab. B.2. Our method still performs very well and achieves results that are comparable with our findings from the main text.

Additional Information: Fig. B.1 shows the full view of the GLOBAL ILL. task. We include this here as, in the main text, we only show the inset that the optimization sees. Note how the left wall changes color, the light changes position, and the large box changes rotation around its horizontal axis.

B.2 Parameter Analysis

Timing: We further investigate the influence of the number of samples N on the convergence and runtime of our method. Recall that N is the number of *perturbations*, and not the number of samples per pixel (cf. the main text and Alg. 1 for details). Fig. B.2 (right) shows that our method's runtime scales linearly with the number of samples we use. This is as the bulk of our method's time is spent in evaluating f, i.e., within the rendering operation. Using more samples means evaluating f more often, which leads to an increased runtime. The overhead of the sampling operation and the gradient computation is small in comparison and, given the linear increase in

Table B.2: Image- and parameter-space MSE (rows) for $\sigma_0 = 0.5$ on different tasks (columns). Our method still performs well and finds the correct parameters.

	CUP	OCCL.	GI	CAUST.
Img.	1.8×10^{-7}	2.2×10^{-3}	1.0×10^{-4}	2.4×10^{-3}
Param.	3.0×10^{-7}	7.2×10^{-3}	6.6×10^{-2}	2.2×10^{-4}

Fig. B.2, can be neglected. We also show Mitsuba's runtime as the blue, dotted line.

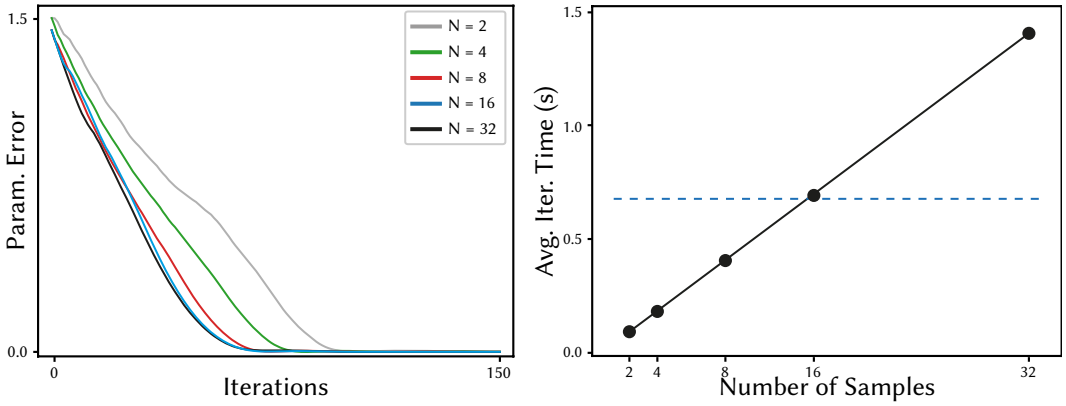


Figure B.2: Convergence comparison on the SHADOW task (left) and runtime analysis (right) of our method for different number of samples N (colored lines left, horizontal axis right).

It is constant, as Mitsuba only renders a single sample, but does so with complicated methods like re-parametrization, gradient tracking or adjoint scattering. We can thus render approx. 16 samples before reaching Mitsuba's runtime (cf. also Tab. 3, main text). Therefore, our runtime does not significantly change with the number of problem dimensions (e.g., 1D vs. nD), but with the time it takes to evaluate f.

Convergence: How does rendering with a higher number of samples N affect the performance of our method? To answer this question, Fig. B.2 (left) shows our method's convergence for different values of N. A low number of $N = 2$ (i.e., a single sample and its antithesis) achieves the slowest convergence rate, while converge improves with more samples and stagnates at around $N = 10$. The final error decreases slightly with higher N (param.-MSE 9.5×10^{-5} for $N = 2$ vs. 4.7×10^{-6} for $N = 16$), but this improvement translates to no visible rendering improvement due to the small scale (10^{-5}) of the values. Using more than $N = 2$ hence yields no improvement here, as the faster convergence is offset by the longer runtime (cf.

Fig. B.2). This relation might, however, change for different tasks.

Choosing σ: Moreover, we investigate how the choice of the initial kernel spread σ_0 affects the optimization outcome. With otherwise equal hyperparameters, we run the SHADOW task with σ_0 varying in $[0,1]$ and show the results in Fig. B.3. For very small σ_0, i.e., $\sigma_0 < 0.2$, the optimization does not converge and produces a similar failure case to the differentiable path tracer by moving the sphere out of the image. This is as for $\sigma_0 \to 0$, our method approaches the rigid optimization by Mitsuba. The loss landscape is not smoothed and the optimization stagnates or fails. For $\sigma_0 \to 1$, we encounter a different failure case: the sampled values are so far spaced out that some of them lie outside the view frustum. As we use only $N = 2$ samples, it is thus very unlikely that we sample the proximity of the true position, leading to very noisy gradients that let our method diverge. This issue can easily be alleviated by enlarging the camera's field of view (FOV), upon which our method converges again (orange dots in Fig. B.3, camera FOV changed from 40° to 60°), as the samples are then back inside the view frustum. In general, we normalize all parameter spaces to $[0,1]$ where possible, e.g., the rotation in the CUP task.

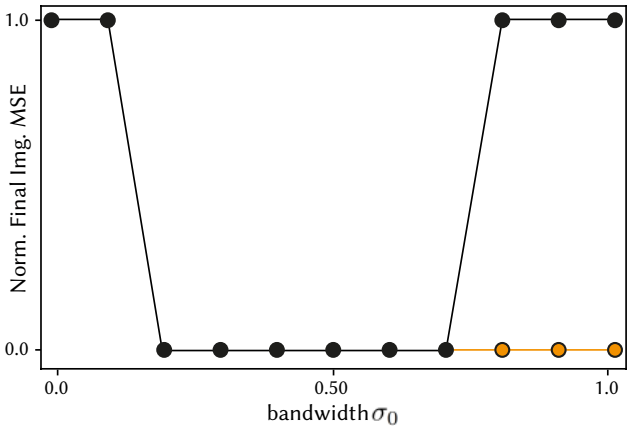


Figure B.3: The effect of σ_0 (horizontal) on the optimization outcome (vertical). We show the outcome with an enlarged camera FOV in orange.

B.3 Compatibility

Plateau-free problems: In this section, we show that our method is compatible with optimization problems that are already plateau-free by design. To this end, we optimize an image texture that is rendered onto a plane under environment illumination. The texture has dimension 128×128 in RGB space, making this a 49,152-dimensional problem.

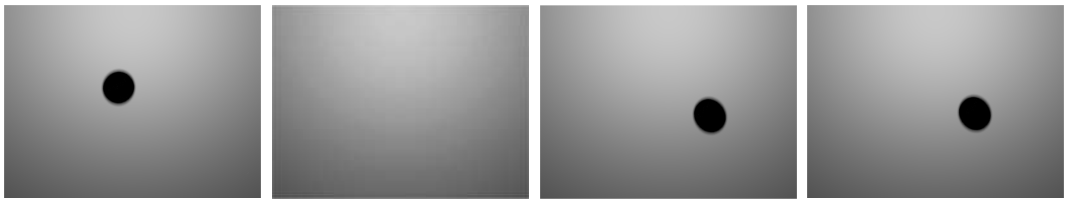


Figure B.5: The SHADOW task re-run with `Redner` as renderer.

Fig. B.4 shows the reference texture and our method's final results, alongside the convergence curves for the image (orange) and parameter (black) error.

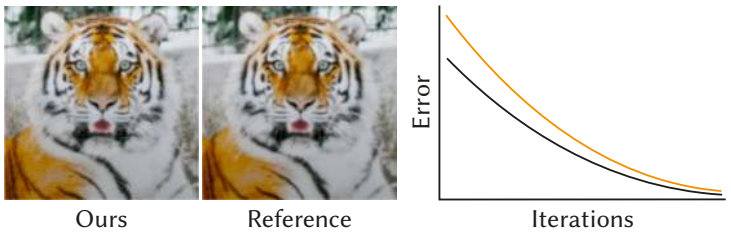


Figure B.4: Texture optimization using our approach. Image- and parameter error in orange and black, respectively.

Path Tracer: Subsequently, we will use a different path tracing engine as backbone for our method and show that our methods also works with a different rendering backbone. For this experiment, we use `Redner`, which uses edge-sampling to derive gradient expressions during path tracing. As we can see from Fig. B.5, this method fails similarly to `Mitsuba`, whereas our method again successfully delivers a complete optimization and finds the correct parameters.

B.4 Additional Derivations

We derive here show how we differentiate our kernel and arrive at the equations presented in the main text.

We define our kernel as a Normal distribution in parameter space with mean 0, i.e.,

$$\kappa(\tau) = \mathcal{N}(0,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\tau^2}{2\sigma^2}\right)$$

which we will then use to offset our current parameters $\theta' = \theta - \kappa(\tau)$. Performing this translation with the original kernel is equivalent to convolving directly with the translated kernel (cf. Fig. B.6), which naturally also holds for the derivative kernel.

This allows us to rewrite the translated kernel as

$$\kappa'(\tau) = \mathcal{N}(\theta, \sigma) = -\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\tau-\theta)^2}{2\sigma^2}\right).$$

Differentiating the above equation yields

$$\frac{\partial \kappa'}{\partial \theta}(\tau) = -\frac{\tau-\theta}{\sigma^3\sqrt{2\pi}} \exp\left(\frac{-(\tau-\theta)^2}{2\sigma^2}\right),$$

which evidently is a translated version of Eq. 11 in the main text. To avoid clutter in the notation, we hence write

$$\frac{\partial \kappa}{\partial \theta}(\tau) = \frac{-\tau}{\sigma^3\sqrt{2\pi}} \exp\left(\frac{-\tau^2}{2\sigma^2}\right).$$

In order to find the CDF of this function, we must integrate its PDF. The PDF must integrate to 1 over the entire domain. As explained in the main text, we treat each halfspace separately and hence normalize the PDF on the positive halfspace to integrate to 0.5, yielding

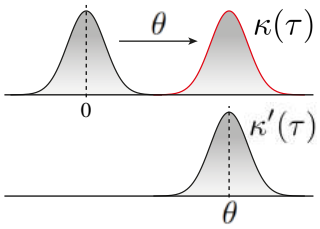


Figure B.6

$$\frac{\tau}{2\sigma^2} \exp\left(\frac{-\tau^2}{2\sigma^2}\right),$$

which, upon integration, results in the CDF

$$-0.5 \exp\left(\frac{-\tau^2}{2\sigma^2}\right) + C^+,$$

where C^+ is the integration constant on the positive halfspace. Handling the negative halfspace analogously results in the same equation, but with a flipped sign and C^- as integration constant. The fact that the CDF must be continuous, monotonically increasing and defined in $(0,1)$ tells us that $C^+ = 1$ and $C^- = 0$. To enable importance sampling with the CDF, we must invert it into the inverse cumulative distribution

function (ICDF), which yields

$$F^{-1}(\xi) = \begin{cases} \sqrt{2\sigma^2 \log(2(1-\xi))} & \tau > 0 \\ \sqrt{2\sigma^2 \log(2\xi)} & \tau < 0\,. \end{cases} \tag{B.1}$$

Solving the domain constraints of the square-root and the logarithm, we find that the ICDF for positive halfspace is defined for $\xi \in [0.5, 1)$, whereas its negative counterpart is defined in $\xi \in (0, 0.5]$, which, given $\xi \in (0, 1)$, can be simplified to yield the final equation presented in the main text

$$F^{-1}(\xi) = \sqrt{-2\sigma^2 \log(\xi)}\,.$$

Appendix C

Appendix: ZeroGrads - Learning Local Surrogate Losses For Non-Differentiable Graphics

This supplementary contains additional information on our surrogate implementation and hyperparameters (Sec. C.1), rendering setups and detailed descriptions of the tasks we solve (Sec. C.2.1).

C.1 Implementation Details

We implement all our experiments in PyTorch [231]. The proxy powering our surrogate is implemented as a MLP and activated by a leaky ReLU. We randomly initialize our Neural Proxy for each optimization run (via the standard PyTorch initialization, for the quadratic proxy, we choose the identity matrix) and optimize its weights alongside the parameter with a separate Adam optimizer. We perform three update steps on the surrogate parameters ϕ per optimization iteration in order to improve the surrogate's fit to the sampled data. This is simple autodiff-driven GD and hence very fast. Note that no new data is sampled between these update steps, they merely serve to improve the surrogate fit and do not increase the required computational budget. For all gradient updates, we use the Adam optimizer with standard parameters and learning rates as specified in Tab. C.1. We additionally experimented with different sampling patterns and found both both low-discrepancy

(Sobol) and antithetic samples and found both to improve performance, and adapt antithetic samples for simplicity. We normalize the network's inputs to [0,1]. For the lower-dimensional tasks ($n_{\text{dim}} < 50$), it suffices to use 3 hidden layers with 64 neurons each, whereas for the higher-dimensional tasks (below the horizontal line in Tab. C.1), we found that we needed to increase the surrogate's capacity to 8 layers à 128 neurons and additionally use positional encoding to increase the frequencies that the network can encode.

C.1.1 Hyperparameters

Our method comes with two hyperparameters: the number of samples N we use to estimate our surrogate's gradient with (cf. Alg.2 in the main text), and the spread of the locality kernel λ, which will influence how far these samples are spaced out around the current parameter θ.

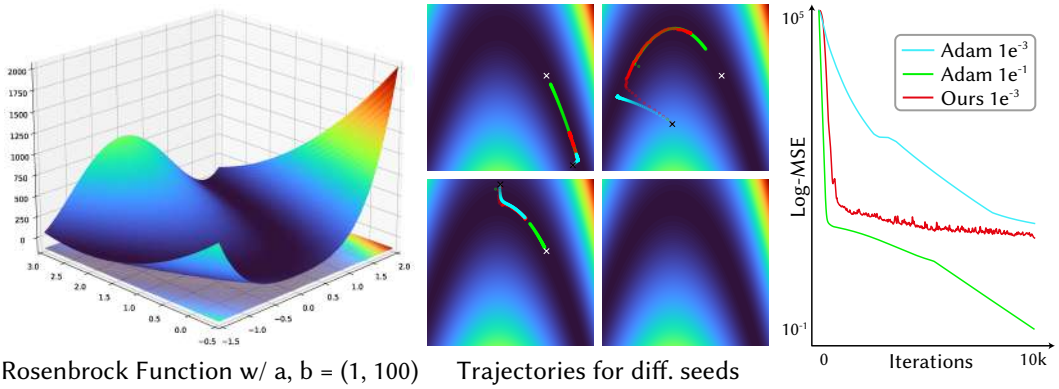


Figure C.1: We evaluate our method on the Rosenbrock function against gradient descent with analytical gradients and Adam with equal learning rate, sample count and iterations. Similar to Adam, our method struggles to make progress in the valleys of low slope, a common limitation of gradient-based techniques. Adam, with a higher learning rate, converges faster than our method. The convergence plots in the right subfigure are median values over an ensemble of 10 independent runs and seeds.

We specify the number of samples N we use for estimating the surrogate's gradients in Tab. C.1. For the lower-dimensional tasks, it suffices to use $N = 2$, whereas for the higher-dimensional tasks, the noise and higher variance from this rough gradient estimate impede convergence and thus require higher sample counts. We would like to emphasize that those are still far lower than what competing methods use, e.g., $2n_{\text{dim}}$ for FD or $m \times n_{\text{dim}}, m \gg 2$, for directional Gaussian smoothing (DGS) [331].

Table C.1: Our hyperparameters σ_o and N, as well as the experiment settings for the different tasks, sorted by dimensionality in ascending order. MPL is short for matplotlib.

	σ_o	N	n_{dim}	LR θ	LR ϕ	Renderer
WICKER	0.33	2	3	1×10^{-3}	1×10^{-3}	Blender
BRDF	0.33	2	4	1×10^{-3}	1×10^{-3}	Mitsuba
CBOX	0.10	2	4	5×10^{-4}	1×10^{-3}	Mitsuba
GRAVITY	0.20	2	5	1×10^{-3}	1×10^{-3}	Blender
ROCKET	0.33	2	10	1×10^{-3}	5×10^{-4}	MPL
NODEGR.	0.20	2	24	1×10^{-3}	1×10^{-3}	Blender
LED	0.33	2	336	1×10^{-3}	1×10^{-3}	Blender
MOSAIC	0.025	16	320	5×10^{-4}	1×10^{-3}	Blender
CAUSTIC	0.013	20	1,024	2×10^{-4}	1×10^{-4}	PyTorch
MESH	0.025	20	7,686	2×10^{-3}	1×10^{-4}	NVDiff.
SPLINE GEN.	0.025	20	8,764	1×10^{-5}	1×10^{-4}	MPL
MLP	0.025	20	35,152	1×10^{-4}	1×10^{-4}	MPL
TEXTURE	0.025	20	196,608	1×10^{-5}	1×10^{-4}	MPL

Our method also benefits from more samples in the lower-dimensional regime, but these come at the cost of increased compute, which is why we tried to achieve a minimal number to keep the overhead low.

We show a comparison of different sample counts on the MESH and MLP tasks in Fig. C.3 and detail the remaining hyperparameters and experiment settings in Tab. C.1, where σ_o denotes the spread of the locality kernel λ. As a general rule of thumb, we recommend setting the initial σ_o to 0.33 on normalized domains and finetune from there, if necessary. For higher-dimensional, interlinked problems, we have found a more fine-granular sampling to be necessary and use $\sigma_o = 0.025$. We use 15% of the locality spread as the spread of the smoothing kernel κ.

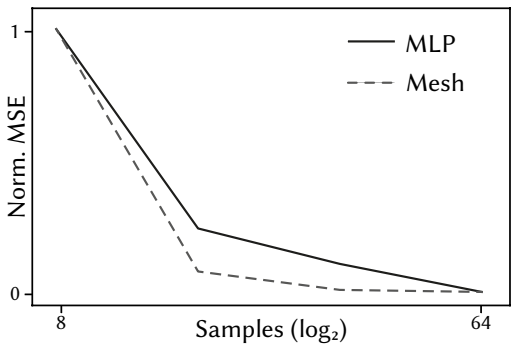


Figure C.3: Final error vs. samplecount N.

C.2 Tasks

This section provides information on the task setup, problems, goals and rendering architectures used.

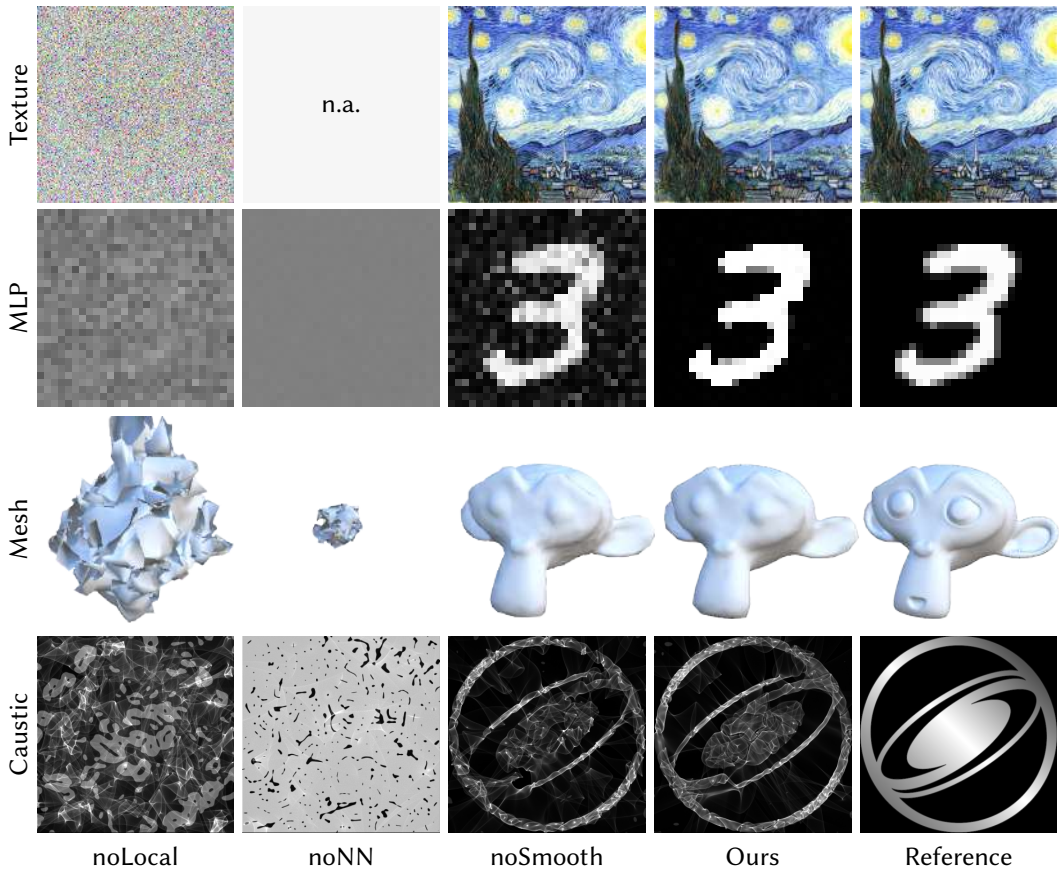


Figure C.2: We show the results of our ablated methods from the main manuscript (Sec. 4.1) on the higher-dimensional tasks. Similar to CMA, the result for the quadratic proxy (noNN) could not be run due to the quadratic memory complexity.

C.2.1 Rendering settings and task descriptions

To render the images for the tasks Mosaic, Wicker, LED, NodeGraph and Gravity, we interface our method with Blender via an efficient socket-based local TCP network, which enables us to make use of Blender's rendering engines and the embedded physics solver. All images were set to render noise-free under either Eevee or Cycles, with 16 to 128 samples and denoising activated. For the tasks BRDF and Cornell-Box, and the comparisons with Mitsuba, we use Mitsuba 3 [130] with the path-replay backpropagation integrator at 16spp. For the Mesh task, we use NVDiffRast [157] with standard hyperparameters. For the remaining tasks Rocket, Spline Generation and Texture, we use a custom matplotlib-based renderer [126]. Note that none of this interfacing is necessary for our method to work, but pure convenience for rapid prototyping and reducing I/O times from and to disk. Most importantly, we do not

propagate any gradient information through the rendering process, even if this were possible, e.g., when using a differentiable renderer. One could alternatively render an image, save it to disk and manually load it and perform a gradient update step, which would yield the same results, but be arguably less convenient.

While some of our higher-dimensional example tasks could in theory also be solved via established, specialized methods (e.g., [130; 215; 115]), they show that our method scales well to higher dimensional problems and reinforce our argument of general applicability. All comparisons to the following optimization algorithms are performed under the same budget of function evaluations.

For the comparisons with GAs, we use the publicly available Python package pygad [77]. For SA [321], we use the `scipy` library [308]. For SPSA [283], we use the publicly available `spsa` package [212]. Note that, while we use standard hyperparameters for the other packages, we here adapted the SPSA perturbation radius to the sampling radius used by our method in order to enable a fair comparison (the default value of 2.0 is too large for many of our problems, e.g., for the delicate task of network training).

TEXTURE For the TEXTURE task, we use our method to optimize the 256 pixels of an image texture, leading to a $256 \times 256 \times 3 = 196,608$ optimization problem. We randomly initialize the texels from $\mathcal{N}(0.5, 0.05)$, i.e., they are drawn from a Normal distribution with mean 0.5, corresponding to a grey value. As is common, we additionally employ a whitening transform during optimization [218].

MLP This task is an extension of the texture task to address the concern that optimization variables are not sufficiently interlinked with each other. To this end, we train a MLP to replicate randomly sampled digits from the MNIST [162] dataset. The MLP has two ReLU-activated hidden layers of 32 neurons and a final layer with 784 neurons that is activated by a Sigmoid, leading to a total of $35,152$ network weights and hence to a $35,152$-dimensional optimization problem. The weights are initialized via the standard formula $\mathcal{U}(-k,k)$, where k is the reciprocal of the layer's input features [231].

CAUSTIC For this task, we take inspiration from Wyman and Davis [320] and

write a fast, rasterization-based caustic renderer.

The idea is that a parallel bundle of rays from a faraway directional light source hits a parameterized refractive surface (our heightfield, usually modeled as a glass slab [218; 225; 272]), and gets refracted according to Snell's law (we use an index of refraction of 1.33). The refracted rays then hit a receiver plane, where we record, for each pixel, the number of received rays, resulting in an approximate caustic. We use an equal ray- and receiver resolution of 512p. The relation between the optimization variables (the heightfield, in our case parameterized as a cubic B-Spline of resolution 32^2, randomly initialized) and the final output in this task is highly non-linear, as a change in the heightfield has the potential to affect various pixels across the entire receiver plane. Moreover, the task is not trivially differentiable, as the conversion of the (continuous) hitpoint on the receiver plane to discrete pixel coordinates in the image grid is a discontinuous operation.

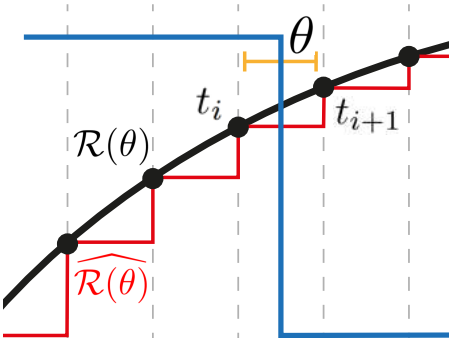


Figure C.4: An illustration why differentiating an ODE solver $\mathcal{R}(\theta)$ w.r.t. time is not trivially differentiable: moving the event-time θ of the blue signal within the yellow interval will not affect the observed outcome, as the solver operates on the discretized version $\widehat{\mathcal{R}}$ only and will continue to observe "on" and "off" at timesteps i and $i+1$, respectively.

MESH For the MESH task, we optimize the vertices of a triangle mesh such that the renderings of the mesh match those of a reference shape. Our source mesh has $2,562$ vertices whose 3D positions we optimize, leading to a highly interlinked $7,686$-dimensional problem. We follow the approach in [215] and use their smooth formulation, the AdamUniform optimizer and the Laplacian regularization, thereby nicely showing that our surrogate successfully learns to replicate the regularized loss landscape. For fairness, all competitors operate in this parametrization. Following [215], the source shape is initialized as a tessellated sphere and rendered from 13 different viewpoints under environment illumination using NVDiffRast [157] –

however, without backpropagating their gradient information; all gradients employed in the optimization are produced by our surrogate.

SPLINE GENERATION For the SPLINE GENERATION task, we train a generative model, a VAE[148], to replicate digits from the MNIST dataset in a spline representation. Our VAE consists of an encoder-MLP with roughly 40k neurons, and a decoder-MLP with $8,764$ neurons. To stabilize training, we use a pre-trained encoder that serves as feature extractor and projects the MNIST images into the latent space, from where we learn a generative decoder that predicts the horizontal and vertical translation of 10 spline support points (initialized diagonally across the image plane). Subsequently, we fit a spline through these predicted support points with a (matplotlib-based) non-differentiable renderer and learn our surrogate on the reconstructed splines' image-space MSE, regularized by the VAE's KLD (weighting factor 0.1). Descending along the surrogate gradients then produces the weights for a generative decoder that can be sampled to generate new MNIST digits. Again, we initialize all stateful components with the standard formula $\mathcal{U}(-k,k)$, where k is the reciprocal of a layer's input features [231].

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015.

[2] Jonas Adler and Ozan Öktem. Learned primal-dual reconstruction. *IEEE transactions on medical imaging*, 37(6), 2018.

[3] Miika Aittala, Timo Aila, and Jaakko Lehtinen. Reflectance modeling by neural texture synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 35(4), 2016.

[4] Maruan Al-Shedivat, Liam Li, Eric Xing, and Ameet Talwalkar. On data efficiency of meta-learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1369–1377. PMLR, 2021.

[5] Shir Amir, Yossi Gandelsman, Shai Bagon, and Tali Dekel. Deep vit features as dense visual descriptors. *arXiv preprint arXiv:2112.05814*, 2(3):4, 2021.

[6] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

[7] Navid Ansari, Hans-Peter Seidel, Nima Vahidi Ferdowsi, and Vahid Babaei. Autoinverse: Uncertainty aware inversion of neural networks. *Advances in Neural Information Processing Systems*, 35:8675–8686, 2022.

[8] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv:1810.09502*, 2018.

[9] Steve Bako, Mark Meyer, Tony DeRose, and Pradeep Sen. Offline deep importance sampling for monte carlo path tracing. In *Comp. Graph. Forum (Proc. EGSR)*, volume 38, 2019.

[10] Sai Bangaru, Lifan Wu, Tzu-Mao Li, Jacob Munkberg, Gilbert Bernstein, Jonathan Ragan-Kelley, Fredo Durand, Aaron Lefohn, and Yong He. Slang.d: Fast, modular and differentiable shader programming. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 42(6):1–28, December 2023. doi: 10.1145/3618353.

[11] Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. Unbiased warped-area sampling for differentiable rendering. *ACM Trans. Graph.*, 39(6), 2020.

[12] Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. Systematically differentiating parametric discontinuities. *ACM Trans Graph*, 40(4), 2021.

[13] Sai Praveen Bangaru, Michaël Gharbi, Tzu-Mao Li, Fujun Luan, Kalyan Sunkavalli, Miloš Hašan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Frédo Durand. Differentiable rendering of neural sdfs through reparameterization. *arXiv preprint arXiv:2206.05344*, 2022.

[14] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5855–5864, 2021.

[15] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5470–5479, 2022.

[16] Russell R Barton. Metamodeling: a state of the art review. In *Proceedings of Winter Simulation Conference*, pages 237–244. IEEE, 1994.

[17] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

[18] Petr Beckmann and Andre Spizzichino. The scattering of electromagnetic waves from rough surfaces. *Norwood*, 1987.

[19] Gabriel Belouze. Optimization without backpropagation. *arXiv preprint arXiv:2209.06302*, 2022.

[20] Mojtaba Bemana, Karol Myszkowski, Jeppe Revall Frisvad, Hans-Peter Seidel, and Tobias Ritschel. Eikonal fields for refractive novel-view synthesis. In *ACM SIGGRAPH*, 2022.

[21] Alexander William Bergman, Petr Kellnhofer, and Gordon Wetzstein. Fast training of neural lumigraph representations using meta learning. In *NeuriIPS*, 2021.

[22] Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. *NeurIPS*, 33, 2020.

[23] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

[24] Sai Bi, Stephen Lombardi, Shunsuke Saito, Tomas Simon, Shih-En Wei, Kevyn Mcphail, Ravi Ramamoorthi, Yaser Sheikh, and Jason Saragih. Deep relightable appearance models for animatable faces. *ACM Trans. Graph.*, 40 (4), 2021.

[25] Paul E Black. Dictionary of algorithms and data structures. 1998.

[26] James F Blinn. Models of light reflection for computer synthesized pictures. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 192–198, 1977.

[27] Daniel Bolya, Chaitanya Ryali, Judy Hoffman, and Christoph Feichtenhofer. Window attention is bugged: How not to interpolate position embeddings. *arXiv preprint arXiv:2311.05613*, 2023.

[28] George EP Box and Norman R Draper. *Empirical model-building and response surfaces*. John Wiley & Sons, 1987.

[29] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Practical shadow mapping. *Journal of Graphics Tools*, 7(4):9–18, 2002.

[30] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[31] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: Autograd and xla. *Astrophysics Source Code Library*, pages ascl–2111, 2021.

[32] Kartik Chandra. An unexpected challenge in using forward-mode automatic differentiation for low-memory deep learning. *Undergrad Theses*, 2021.

[33] Kartik Chandra, Tzu-Mao Li, Joshua Tenenbaum, and Jonathan Ragan-Kelley. Designing perceptual puzzles by differentiating probabilistic programs. In *Proc. SIGRAPH*, 2022.

[34] Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. Gradient descent: The ultimate optimizer. *Advances in Neural Information Processing Systems*, 35:8214–8225, 2022.

[35] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[36] Swarat Chaudhuri and Armando Solar-Lezama. Smooth interpretation. *ACM Sigplan Notices*, 45(6):279–291, 2010.

[37] Swarat Chaudhuri and Armando Solar-Lezama. Smoothing a program soundly and robustly. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 277–292. Springer, 2011.

[38] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *NeuRIPS*, 32, 2019.

[39] Per Christensen, Julian Fong, Jonathan Shade, Wayne Wooten, Brenden Schubert, Andrew Kensler, Stephen Friedman, Charlie Kilpatrick, Cliff Ramshaw, Marc Bannister, et al. Renderman: An advanced path-tracing architecture for movie rendering. *ACM Trans. Graph.*, 37(3):1–21, 2018.

[40] Forrester Cole, Kyle Genova, Avneesh Sud, Daniel Vlasic, and Zhoutong Zhang. Differentiable surface rendering via non-differentiable sampling. In *Proc. ICCV*, 2021.

[41] Robert L Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Trans. Graph.*, 1(1), 1982.

[42] Veronica Czitrom. One-factor-at-a-time versus designed experiments. *The American Statistician*, 53(2):126–131, 1999.

[43] Kristin J Dana and Jing Wang. Device for convenient measurement of spatially varying bidirectional reflectance. *JOSA A*, 21(1), 2004.

[44] Cuthbert Daniel. One-at-a-time plans. *Journal of the American statistical association*, 68(342):353–360, 1973.

[45] Tristan Deleu, Tobias Würfl, Mandana Samiei, Joseph Paul Cohen, and Yoshua Bengio. Torchmeta: A Meta-Learning library for PyTorch, 2019. URL `arXiv:1909.06576`.

[46] Tristan Deleu, David Kanaa, Leo Feng, Giancarlo Kerg, Yoshua Bengio, Guillaume Lajoie, and Pierre-Luc Bacon. Continuous-time meta-learning with forward mode differentiation. *arXiv preprint arXiv:2203.01443*, 2022.

[47] Thomas Deliot, Eric Heitz, and Laurent Belcour. Transforming a non-differentiable rasterizer into a differentiable one with stochastic gradient estimation. *arXiv preprint arXiv:2404.09758*, 2024.

[48] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[49] Xi Deng, Fujun Luan, Bruce Walter, Kavita Bala, and Steve Marschner. Reconstructing translucent objects using differentiable rendering. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–10, 2022.

[50] Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 37(4), 2018.

[51] Valentin Deschaintre, Miika Aittala, Frédo Durand, George Drettakis, and Adrien Bousseau. Flexible svbrdf capture with a multi-image deep network. In *Computer graphics forum*, volume 38, pages 1–13. Wiley Online Library, 2019.

[52] Valentin Deschaintre, George Drettakis, and Adrien Bousseau. Guided fine-tuning for large-scale material transfer. In *Comp. Graph. Forum*, volume 39, 2020.

[53] Luc Devroye. Sample-based non-uniform random variate generation. In

Proceedings of the 18th conference on Winter simulation, pages 260–265, 1986.

[54] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[55] Julie Dorsey, Holly Rushmeier, and François Sillion. *Digital modeling of material appearance*. 2010.

[56] Ron O Dror, Edward H Adelson, and Alan S Willsky. Recognition of surface reflectance properties from a single image under unknown real-world illumination. *CVPR*, 2001.

[57] John C Duchi, Peter L Bartlett, and Martin J Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2): 674–701, 2012.

[58] DS Ebert. *Texturing & Modeling, A procedural Approach*. Morgan Kaufman, 2002.

[59] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 571–576. 2023.

[60] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999.

[61] L eon Bottou. Online learning and stochastic approximations. *Online learning in neural networks*, 17(9):142, 1998.

[62] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460. IEEE, 2012.

[63] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[64] Michael Fischer and Tobias Ritschel. Metappearance: Meta-learning for visual appearance reproduction. *ACM Trans. Graph.*, 41(6):1–13, 2022.

[65] Michael Fischer and Tobias Ritschel. Plateau-free differentiable path tracing. *arXiv preprint arXiv:2211.17263*, 2022.

[66] Michael Fischer and Tobias Ritschel. Plateau-reduced differentiable path tracing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4285–4294, 2023.

[67] Michael Fischer and Tobias Ritschel. Zerograds: Learning local surrogates for non-differentiable graphics. *ACM Transactions on Graphics (TOG)*, 43 (4):1–15, 2024.

[68] Michael Fischer, Konstantin Kobs, and Andreas Hotho. Nicer: Aesthetic image enhancement with humans in the loop. *arXiv preprint arXiv:2012.01778*, 2020.

[69] Michael Fischer, Iliyan Georgiev, Thibault Groueix, Vladimir G Kim, Tobias Ritschel, and Valentin Deschaintre. Sama: Material-aware 3d selection and segmentation. *arXiv preprint arXiv:2411.19322*, 2024.

[70] Michael Fischer, Zhengqin Li, Thu Nguyen-Phuoc, Aljaz Bozic, Zhao Dong, Carl Marshall, and Tobias Ritschel. Nerf analogies: Example-based visual attribute transfer for nerfs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4640–4650, 2024.

[71] Michael Fischer, Tobias Ritschel, et al. Higher-order differentiable rendering. *arXiv preprint arXiv:2412.03489*, 2024.

[72] Harley Flanders. Differentiation under the integral sign. *The American Mathematical Monthly*, 80(6):615–627, 1973.

[73] Sebastian Flennerhag, Andrei A Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2019.

[74] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *CVPR*, 2019.

[75] Alexander IJ Forrester and Andy J Keane. Recent advances in surrogate-based optimization. *Progress in aerospace sciences*, 45(1-3):50–79, 2009.

[76] Anna Frühstück, Ibraheem Alhashim, and Peter Wonka. Tilegan: synthesis of large-scale non-homogeneous textures. *ACM Transactions on graphics (TOG)*, 38(4):1–11, 2019.

[77] Ahmed Fawzy Gad. Pygad: An intuitive genetic algorithm python library, 2021.

[78] Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. Deep inverse rendering for high-resolution SVBRDF estimation from an arbitrary number of images. *ACM Trans. Graph.*, 38(4), 2019.

[79] Marc-André Gardner, Kalyan Sunkavalli, Ersin Yumer, Xiaohui Shen, Emiliano Gambaretto, Christian Gagné, and Jean-François Lalonde. Learning to predict indoor illumination from a single image. *arXiv preprint arXiv:1704.00090*, 2017.

[80] Marc-André Gardner, Yannick Hold-Geoffroy, Kalyan Sunkavalli, Christian Gagné, and Jean-François Lalonde. Deep parametric indoor lighting estimation. In *PICCV*, 2019.

[81] Mathieu Garon, Kalyan Sunkavalli, Sunil Hadap, Nathan Carr, and Jean-François Lalonde. Fast spatially-varying indoor lighting estimation. In *CVPR*, 2019.

[82] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv:1508.06576*, 2015.

[83] Stamatios Georgoulis, Konstantinos Rematas, Tobias Ritschel, Mario Fritz, Tinne Tuytelaars, and Luc Van Gool. What is around the camera? In *ICCV*, 2017.

[84] Stamatios Georgoulis, Konstantinos Rematas, Tobias Ritschel, Efstratios Gavves, Mario Fritz, Luc Van Gool, and Tinne Tuytelaars. Reflectance and natural illumination from single-material specular objects using deep learning. *IEEE PAMI*, 40(8), 2017.

[85] Lily Goli, Cody Reading, Silvia Sellán, Alec Jacobson, and Andrea Tagliasacchi. Bayes' rays: Uncertainty quantification for neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20061–20070, 2024.

[86] Ian Goodfellow. Deep learning, 2016.

[87] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[88] Henri Gouraud. *Computer display of curved surfaces*. The University of Utah, 1971.

[89] Josif Grabocka, Randolf Scholz, and Lars Schmidt-Thieme. Learning surrogate losses. *arXiv preprint arXiv:1905.10108*, 2019.

[90] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proc. SIGGRAPH*, pages 9–20, 1998.

[91] Darya Guarnera, Giuseppe Claudio Guarnera, Abhijeet Ghosh, Cornelia Denk, and Mashhuda Glencross. Brdf representation and acquisition. 35(2), 2016.

[92] Paul Guerrero, Miloš Hašan, Kalyan Sunkavalli, Radomír Měch, Tamy Boubekeur, and Niloy J Mitra. Matformer: A generative model for procedural materials. *arXiv preprint arXiv:2207.01044*, 2022.

[93] Julia Guerrero-Viu, Michael Fischer, Iliyan Georgiev, Elena Garces, Diego Gutierrez, Belen Masia, and Valentin Deschaintre. Fine-grained spatially varying material selection in images. *arXiv preprint arXiv:2506.09023*, 2025.

[94] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

[95] Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. Materialgan: reflectance capture using a generative svBRDF model. *arXiv:2010.00114*, 2020.

[96] H-M Gutmann. A radial basis function method for global optimization. *J global optimization*, 19(3):201–227, 2001.

[97] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv:1609.09106*, 2016.

[98] John K Haas. A history of the unity game engine. *Diss. WORCESTER POLYTECHNIC INSTITUTE*, 483:484, 2014.

[99] Marc Habermann, Lingjie Liu, Weipeng Xu, Michael Zollhoefer, Gerard Pons-Moll, and Christian Theobalt. Real-time deep dynamic characters. *ACM Trans. Graph.*, 40(4):1–16, 2021.

[100] John Michael Hammersley and JG Mauldon. General principles of antithetic variates. In *Mathematical proceedings of the Cambridge philosophical society*, volume 52, 1956.

[101] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[102] Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International conference on machine learning*, pages 1225–1234. PMLR, 2016.

[103] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[104] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[105] Eric Heitz, Laurent Belcour, and Thomas Chambon. Iterative α-(de) blending: A minimalist deterministic diffusion model. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–8, 2023.

[106] Philipp Henzler, Niloy Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3d shape from unstructured 2d image collections. In *Proc. ICCV*, 2019.

[107] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Learning a neural 3d texture space from 2d exemplars. In *CVPR*, 2020.

[108] Philipp Henzler, Valentin Deschaintre, Niloy J Mitra, and Tobias Ritschel. Generative modelling of brdf textures from flash images. *ACM Trans Graph (Proc SIGGRAPH Asia)*, 40(5), 2021.

[109] Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. Product importance sampling for light transport path guiding. 35 (4), 2016.

[110] Lukas Hermanns and Tobias Alexander Franke. Screen space cone tracing for glossy reflections. In *ACM SIGGRAPH 2014 Posters*, pages 1–1. 2014.

[111] Pedro Hermosilla, Sebastian Maisch, Tobias Ritschel, and Timo Ropinski. Deep-learning the latent space of light transport. *Comp. Graph. Forum*, 38(4), 2019.

[112] Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. Image analogies. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 557–570. 2023.

[113] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[114] Yannick Hold-Geoffroy, Akshaya Athawale, and Jean-François Lalonde. Deep sky modeling for single image outdoor lighting estimation. In *CVPR*, 2019.

[115] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.

[116] John H Holland. Genetic algorithms and the optimal allocation of trials. *SIAM journal on computing*, 2(2):88–105, 1973.

[117] Holger H Hoos and Thomas Stutzle. Evaluating las vegas algorithms-pitfalls and remedies. *arXiv preprint arXiv:1301.7383*, 2013.

[118] Anita Hu, Nishkrit Desai, Hassan Abu Alhaija, Seung Wook Kim, and Maria Shugrina. Diffusion texture painting. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–12, 2024.

[119] Bingyang Hu, Jie Guo, Yanjun Chen, Mengtian Li, and Yanwen Guo. DeepBRDF: A deep representation for manipulating measured brdf. 39(2), 2020.

[120] Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Tieniu Tan, and Jian Sun. Meta-sr: A magnification-arbitrary network for super-resolution. In *CVPR*, 2019.

[121] Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. Node graph optimization using differentiable proxies. In *ACM SIGGRAPH*, 2022.

[122] Yiwei Hu, Chengan He, Valentin Deschaintre, Julie Dorsey, and Holly Rushmeier. An inverse procedural modeling pipeline for svbrdf maps. *ACM Trans. Graph.*, 41(2):1–17, 2022.

[123] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[124] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024.

[125] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

[126] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[127] Yuchi Huo, Rui Wang, Ruzahng Zheng, Hualin Xu, Hujun Bao, and Sung-Eui Yoon. Adaptive incident radiance field sampling and reconstruction using deep reinforcement learning. *ACM Trans. Graph.*, 39(1):1–17, 2020.

[128] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. *NeuRIPS*, 31, 2018.

[129] Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[130] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022, 2022.

[131] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. Dr.jit: A just-in-time compiler for differentiable rendering. *ACM Trans Graph. (Proc. SIGGRAPH)*, 41(4), 2022.

[132] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. Dr. jit: A just-in-time compiler for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–19, 2022.

[133] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org.

[134] Kevin G Jamieson, Robert Nowak, and Ben Recht. Query complexity of derivative-free optimization. *Advances in Neural Information Processing Systems*, 25, 2012.

[135] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proc. CVPR*, 2020.

[136] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M Kakade, and Michael I Jordan. How to escape saddle points efficiently. In *International Conference on Machine Learning*, pages 1724–1732. PMLR, 2017.

[137] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *J Global optimization*, 13(4): 455–92, 1998.

[138] Bela Julesz. Experiments in the visual perception of texture. *Scientific American*, 232(4), 1975.

[139] Herman Kahn. Random sampling (monte carlo) techniques in neutron attenuation problems. i. *Nucleonics (US) Ceased publication*, 6(See also NSA 3-990), 1950.

[140] James T Kajiya. The rendering equation. In *Proc. SIGGRAPH*, 1986.

[141] Kaizhang Kang, Zimin Chen, Jiaping Wang, Kun Zhou, and Hongzhi Wu. Efficient reflectance capture using an autoencoder. *ACM Trans. Graph.*, 37 (4), 2018.

[142] Kaizhang Kang, Cihui Xie, Chengan He, Mingqi Yi, Minyi Gu, Zimin Chen, Kun Zhou, and Hongzhi Wu. Learning efficient illumination multiplexing for joint capture of reflectance and shape. *ACM Trans. Graph.*, 38(6):165–1, 2019.

[143] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D mesh renderer. In *Proc. CVPR*, pages 3907–16, 2018.

[144] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.

[145] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A simple and robust mutation strategy for the metropolis light transport algorithm. In *Computer Graphics Forum*, volume 21, pages 531–540. Wiley Online Library, 2002.

[146] Alexander Keller, Pascal Grittmann, Jiří Vorba, Iliyan Georgiev, Martin Šik, Eugene d'Eon, Pascal Gautron, Petr Vévoda, and Ivo Kondapaneni. Advances in monte carlo rendering: The legacy of jaroslav křivánek. In *SIGGRAPH Courses*, 2020.

[147] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

[148] Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[149] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[150] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

[151] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

[152] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[153] Thomas Kiser, Michael Eigensatz, Minh Man Nguyen, Philippe Bompas, and Mark Pauly. Architectural caustics—controlling light with geometry. In *Advances in architectural geometry 2012*, pages 91–106. Springer, 2013.

[154] Alexandr Kuznetsov, Milos Hasan, Zexiang Xu, Ling-Qi Yan, Bruce Walter, Nima Khademi Kalantari, Steve Marschner, and Ravi Ramamoorthi. Learning generative models for rendering specular microgeometry. *ACM Trans. Graph.*, 38(6), 2019.

[155] Alexandr Kuznetsov, Krishna Mullia, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. Neumip: multi-resolution neural materials. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 40(4), 2021.

[156] Eric P Lafortune and Yves D Willems. A 5d tree to reduce the variance of monte carlo ray tracing. In *EGSR*, pages 11–20, 1995.

[157] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Trans Graph*, 39(6), 2020.

[158] Robert Lange, Tom Schaul, Yutian Chen, Tom Zahavy, Valentin Dalibard, Chris Lu, Satinder Singh, and Sebastian Flennerhag. Discovering evolution strategies via meta-black-box optimization. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 29–30, 2023.

[159] Averill M Law, W David Kelton, and W David Kelton. *Simulation modeling and analysis*, volume 3. Mcgraw-hill New York, 2007.

[160] Qiqin Le, Jiamu Bu, Yanke Qu, Bo Zhu, and Tao Du. Computational biomimetics of winged seeds. *ACM Transactions on Graphics (TOG)*, 43(6):1–13, 2024.

[161] Quentin Le Lidec, Ivan Laptev, Cordelia Schmid, and Justin Carpentier. Differentiable rendering with perturbed optimizers. *NeurIPS*, 34, 2021.

[162] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[163] Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. Reparameterization gradient for non-differentiable models. *Proc. NeurIPS*, 31, 2018.

[164] Hendrik PA Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Image-based reconstruction of spatial appearance and geometric detail. *ACM Trans. Graph.*, 22(2), 2003.

[165] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[166] Beichen Li, Liang Shi, and Wojciech Matusik. End-to-end procedural material capture with proxy-free mixed-integer optimization. *ACM Transactions on Graphics (TOG)*, 42(4):1–15, 2023.

[167] Beichen Li, Yiwei Hu, Paul Guerrero, Milos Hasan, Liang Shi, Valentin Deschaintre, and Wojciech Matusik. Procedural material generation with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 43(6):1–14, 2024.

[168] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[169] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable Monte Carlo ray tracing through edge sampling. *ACM Trans Graph.*, 37(6), 2018.

[170] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph.*, 39(6):1–15, 2020.

[171] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image. In *Proc. CVPR*, 2020.

[172] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv:1707.09835*, 2017.

[173] Selena Zihan Ling, Nicholas Sharp, and Alec Jacobson. Vectoradam for rotation equivariant geometry optimization. *Advances in Neural Information Processing Systems*, 35:4111–4122, 2022.

[174] Chen Liu, Michael Fischer, and Tobias Ritschel. Learning to learn and sample brdfs. In *Computer Graphics Forum*, volume 42, pages 201–211. Wiley Online Library, 2023.

[175] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[176] Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. Material editing using a physically based rendering network. In *CVPR*, 2017.

[177] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[178] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *Proc. CVPR*, 2020.

[179] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3D reasoning. In *Proc. ICCV*, pages 7708–17, 2019.

[180] Tom Lokovic and Eric Veach. Deep shadow maps. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 311–318. 2023.

[181] Stephen Lombardi and Ko Nishino. Reflectance and natural illumination from a single image. In *ECCV*. Springer, 2012.

[182] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *Proc. ECCV*, pages 154–169. Springer, 2014.

[183] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Trans. Graph.*, 38 (6):1–14, 2019.

[184] Linjie Lyu, Marc Habermann, Lingjie Liu, Ayush Tewari, Christian Theobalt, et al. Efficient and differentiable shadow computation for inverse problems. In *Proc. ICCV*, 2021.

[185] Joe Marks, Brad Andalman, Paul A Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, et al. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proc. SIGGRAPH*, pages 389–400, 1997.

[186] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11 (2):431–441, 1963.

[187] Stephen R Marschner and Donald P Greenberg. Inverse lighting for photography. In *Color and Imaging Conference*, volume 5, pages 262–265. Society of Imaging Science and Technology, 1997.

[188] Stephen Robert Marschner. *Inverse rendering for computer graphics*. Cornell University, 1998.

[189] Wojciech Matusik. *A data-driven reflectance model*. PhD thesis, Massachusetts Institute of Technology, 2003.

[190] Maxim Maximov, Laura Leal-Taixé, Mario Fritz, and Tobias Ritschel. Deep appearance maps. In *ICCV*, 2019.

[191] Morgan McGuire and Michael Mara. Efficient gpu screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3(4):73–85, 2014.

[192] Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

[193] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[194] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1), 2021.

[195] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21(132): 1–62, 2020.

[196] Louis Montaut, Quentin Le Lidec, Antoine Bambade, Vladimir Petrik, Josef Sivic, and Justin Carpentier. Differentiable collision detection: a randomized smoothing approach. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3240–3246. IEEE, 2023.

[197] Joep Moritz, Stuart James, Tom SF Haines, Tobias Ritschel, and Tim Weyrich. Texture stationarization: Turning photos into tileable textures. In *Computer graphics forum*, volume 36, pages 177–188. Wiley Online Library, 2017.

[198] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. *Proc. NeurIPS*, 31, 2018.

[199] Thomas Müller, Markus Gross, and Jan Novák. Practical path guiding for efficient light-transport simulation. In *Comp. Graph. Forum*, volume 36, 2017.

[200] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph. (Proc, SIGGRAPH)*, 38(5), 2019.

[201] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *ACM Trans. Graph. (proc SIGGRAPH)*, 40(4), 2021.

[202] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *arXiv preprint arXiv:2106.12372*, 2021.

[203] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.

[204] Andreas Munk, Adam Ścibior, Atılım Güneş Baydin, Andrew Stewart, Goran Fernlund, Anoush Poursartip, and Frank Wood. Deep probabilistic surrogate networks for universal simulator approximation. *arXiv preprint arXiv:1910.11950*, 2019.

[205] Peter Naglič, Franjo Pernuš, Boštjan Likar, and Miran Bürmen. Lookup table-based sampling of the phase function for monte carlo simulations of light propagation in turbid media. *Biomedical Optics Express*, 8(3):1895–1910, 2017.

[206] Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, H-P Seidel, and

Tobias Ritschel. Deep shading: convolutional neural networks for screen space shading. In *Comp. Graph. Forum*, volume 36, pages 65–78, 2017.

[207] Giljoo Nam, Joo Ho Lee, Diego Gutierrez, and Min H Kim. Practical svbrdf acquisition of 3d objects with unstructured flash photography. *ACM Trans. Graph.*, 37(6), 2018.

[208] Jiří Navrátil, Alan King, Jesus Rios, Georgios Kollias, Ruben Torrado, and Andrés Codas. Accelerating physics-based simulations using end-to-end neural network proxies: An application in oil reservoir modeling. *Frontiers in big Data*, 2:33, 2019.

[209] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17:527–566, 2017.

[210] Addy Ngan, Frédo Durand, and Wojciech Matusik. Experimental analysis of brdf models. *Rendering Techniques*, 2005.

[211] Chuong H Nguyen, Tobias Ritschel, Karol Myszkowski, Elmar Eisemann, and Hans-Peter Seidel. 3d material style transfer. In *Computer Graphics Forum*, volume 31, pages 431–438. Wiley Online Library, 2012.

[212] Jack Nguyen. spsa. `https://github.com/SimpleArt/spsa`, 2022.

[213] Thu Nguyen-Phuoc, Gabriel Schwartz, Yuting Ye, Stephen Lombardi, and Lei Xiao. Alteredavatar: Stylizing dynamic 3d avatars with fast style adaptation. *arXiv preprint arXiv:2305.19245*, 2023.

[214] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv:1803.02999*, 2018.

[215] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 40(6), December 2021. doi: 10.1145/3478513.3480501.

[216] Baptiste Nicolet, Fabrice Rousselle, Jan Novak, Alexander Keller, Wenzel Jakob, and Thomas Müller. Recursive control variates for inverse rendering. *ACM Transactions on Graphics (TOG)*, 42(4):1–13, 2023.

[217] Jannik Boll Nielsen, Henrik Wann Jensen, and Ravi Ramamoorthi. On optimal, minimal brdf sampling for reflectance acquisition. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 34(6), 2015.

[218] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), December 2019. doi: 10.1145/3355089.3356498.

[219] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans. Graph.*, 38(6), 2019.

[220] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Trans Graph*, 39(4), 2020.

[221] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[222] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics (TOG)*, 23(3):488–495, 2004.

[223] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[224] Art Owen and Yi Zhou. Safe and effective importance sampling. *J of the American Statistical Association*, 95(449), 2000.

[225] Marios Papas, Wojciech Jarosz, Wenzel Jakob, Szymon Rusinkiewicz, Wojciech Matusik, and Tim Weyrich. Goal-based caustics. In *Computer Graphics Forum*, volume 30, pages 503–511. Wiley Online Library, 2011.

[226] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019.

[227] Jeong Joon Park, Aleksander Holynski, and Steven M Seitz. Seeing the world in a bag of chips. In *CVPR*, 2020.

[228] Keunhong Park, Philipp Henzler, Ben Mildenhall, Jonathan T Barron, and Ricardo Martin-Brualla. Camp: Camera preconditioning for neural radiance fields. *ACM Transactions on Graphics (TOG)*, 42(6):1–11, 2023.

[229] Sejun Park, Chulhee Yun, Jaeho Lee, and Jinwoo Shin. Minimum width for universal approximation. *arXiv preprint arXiv:2006.08859*, 2020.

[230] Konstantinos E Parsopoulos and Michael N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural computing*, 1(2):235–306, 2002.

[231] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[232] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 2019.

[233] Yash Patel, Tomáš Hodaň, and Jiří Matas. Learning surrogates via deep embedding. In *Proc. ECCV*, pages 205–221. Springer, 2020.

[234] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3): 287–296, 1985.

[235] Ken Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, 2002.

[236] Felix Petersen, Amit H. Bermano, Oliver Deussen, and Daniel Cohen-Or. Pix2vex: Image-to-geometry reconstruction using a smooth differentiable renderer, 2019.

[237] Felix Petersen, Bastian Goldluecke, Christian Borgelt, and Oliver Deussen. Gendr: A generalized differentiable renderer. In *Proc. CVPR*, pages 4002–4011, 2022.

[238] André Susano Pinto, Alexander Kolesnikov, Yuge Shi, Lucas Beyer, and Xiaohua Zhai. Tuning computer vision models with task rewards. In *International Conference on Machine Learning*, pages 33229–33239. PMLR, 2023.

[239] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5): 1–17, 1964.

[240] Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int J Computer Vision*, 40(1), 2000.

[241] Michael JD Powell. Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical programming*, 14:224–248, 1978.

[242] Michael JD Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*, pages 51–67. Springer, 1994.

[243] Arcot J Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 91–100, 1999.

[244] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[245] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[246] Nestor V Queipo, Raphael T Haftka, Wei Shyy, Tushar Goel, Rajkumar Vaidyanathan, and P Kevin Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.

[247] Lara Raad, Axel Davy, Agnès Desolneux, and Jean-Michel Morel. A survey of exemplar-based texture synthesis. *Annals of Mathematical Sciences and Applications*, 3(1), 2018.

[248] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

[249] Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. Neural BTF compression and interpolation. In *Comp. Graph. Forum*, volume 38, 2019.

[250] Gilles Rainer, Abhijeet Ghosh, Wenzel Jakob, and Tim Weyrich. Unified neural encoding of btfs. 39(2), 2020.

[251] Tom Rainforth, Rob Cornish, Hongseok Yang, Andrew Warrington, and Frank Wood. On nesting Monte Carlo estimators. In *Proc. ICML*, pages 4267–4276. PMLR, 2018.

[252] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 117–128, 2001.

[253] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[254] Pradyumna Reddy, Paul Guerrero, Matt Fisher, Wilmot Li, and Niloy J Mitra. Discovering pattern structure using differentiable compositing. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[255] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *Proc. CVPR*, 2021.

[256] Konstantinos Rematas, Tobias Ritschel, Mario Fritz, Efstratios Gavves, and Tinne Tuytelaars. Deep reflectance maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4508–4516, 2016.

[257] Alex Renda, Yishen Chen, Charith Mendis, and Michael Carbin. Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 442–455. IEEE, 2020.

[258] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, 2015.

[259] Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. A versatile scene model with differentiable visibility applied to generative pose estimation. In *Proc. ICCV*, 2015.

[260] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J Global Optimization*, 56(3):1247–1293, 2013.

[261] Tobias Ritschel, Thorsten Grosch, Min H Kim, H-P Seidel, Carsten Dachsbacher, and Jan Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM transactions on graphics (tog)*, 27(5):1–8, 2008.

[262] Carlos Rodriguez-Pardo and Elena Garces. Seamlessgan: Self-supervised synthesis of tileable texture maps. *IEEE Transactions on Visualization and Computer Graphics*, 29(6):2914–2925, 2022.

[263] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[264] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[265] Scott D Roth. Ray casting for modeling solids. *Computer graphics and image processing*, 18(2):109–144, 1982.

[266] Szymon M Rusinkiewicz. A new change of variables for efficient brdf representation. In *EGWR*, 1998.

[267] Chaitanya Ryali, Yuan-Ting Hu, Daniel Bolya, Chen Wei, Haoqi Fan, Po-Yao Huang, Vaibhav Aggarwal, Arkabandhu Chowdhury, Omid Poursaeed, Judy Hoffman, et al. Hiera: A hierarchical vision transformer without the bells-and-whistles. In *International Conference on Machine Learning*, pages 29441–29454. PMLR, 2023.

[268] Paul Sanzenbacher, Lars Mescheder, and Andreas Geiger. Learning neural light transport. *arXiv:2006.03427*, 2020.

[269] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, pages 317–335. PMLR, 2018.

[270] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in neural information processing systems*, 28, 2015.

[271] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[272] Yuliy Schwartzburg, Romain Testuz, Andrea Tagliasacchi, and Mark Pauly. High-contrast computational caustic design. *ACM Transactions on Graphics (TOG)*, 33(4):1–11, 2014.

[273] Prafull Sharma, Julien Philip, Michaël Gharbi, Bill Freeman, Fredo Durand, and Valentin Deschaintre. Materialistic: Selecting similar materials in images. *ACM Transactions on Graphics*, 42(4), 2023.

[274] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Optimal approximation rate of relu networks in terms of width and depth. *Journal de Mathématiques Pures et Appliquées*, 157:101–135, 2022.

[275] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. Match: differentiable material graphs for procedural material capture. *ACM Trans. Graph.*, 39(6):1–15, 2020.

[276] Sergey Shirobokov, Vladislav Belavin, Michael Kagan, Andrei Ustyuzhanin, and Atilim Gunes Baydin. Black-box optimization with local generative surrogates. *Proc. NeurIPS*, 33:14650–14662, 2020.

[277] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[278] Ping Tat Sin, Hiu Fung Ng, and Hong Va Leong. Neural proxy: Empowering neural volume rendering for animation. In *Proc. Pacific Graphics*, pages 1–6, 2021.

[279] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *arXiv:1906.01618*, 2019.

[280] Vincent Sitzmann, Eric R Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. MetaSDF: Meta-learning signed distance functions. *arXiv:2006.09662*, 2020.

[281] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *NeurIPS*, 34, 2021.

[282] Shuran Song and Thomas Funkhouser. Neural illumination: Lighting prediction for indoor environments. In *CVPR*, 2019.

[283] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[284] Pratul P Srinivasan, Ben Mildenhall, Matthew Tancik, Jonathan T Barron, Richard Tucker, and Noah Snavely. Lighthouse: Predicting lighting volumes for spatially-coherent illumination. In *CVPR*, 2020.

[285] Joe Staines and David Barber. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

[286] Joe Staines and David Barber. Optimization by variational bounding. In *ESANN*, 2013.

[287] Jos Stam. Computing light transport gradients using the adjoint method. *arXiv preprint arXiv:2006.15059*, 2020.

[288] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 557–562, 2002.

[289] Adéla Šubrtová, Michal Lukáč, Jan Čech, David Futschik, Eli Shechtman, and Daniel Sỳkora. Diffusion image analogies. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–10, 2023.

[290] Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

[291] Hyung Ju Terry Suh, Tao Pang, and Russ Tedrake. Bundled gradients through contact via randomized smoothing. *IEEE Robotics and Automation Letters*, 7 (2):4000–4007, 2022.

[292] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[293] Alejandro Sztrajman, Gilles Rainer, Tobias Ritschel, and Tim Weyrich. Neural brdf representation and importance sampling. In *Comp. Graph. Forum*, volume 40, 2021.

[294] Shufeng Tan and Michael L Mayrovouniotis. Reducing data dimensionality through optimizing neural network inputs. *AIChE Journal*, 41(6), 1995.

[295] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

[296] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*, 2021.

[297] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih,

Matthias Nießner, et al. State of the art on neural rendering. In *Comp. Graph. Forum*, volume 39, 2020.

[298] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Trans. Graph.*, 38(4): 1–12, 2019.

[299] Ethan Tseng, Felix Yu, Yuting Yang, Fahim Mannan, Karl ST Arnaud, Derek Nowrouzezahrai, Jean-François Lalonde, and Felix Heide. Hyperparameter optimization in black-box image processing using differentiable proxies. *ACM Trans. Graph.*, 38(4):27–1, 2019.

[300] Ethan Tseng, Yuxuan Zhang, Lars Jebe, Xuaner Zhang, Zhihao Xia, Yifei Fan, Felix Heide, and Jiawen Chen. Neural photo-finishing. *ACM Trans. Graph.*, 41(6):1–15, 2022.

[301] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proc. CVPR*, 2017.

[302] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, 2016.

[303] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[304] Eric Veach. *Robust Monte Carlo methods for light transport simulation*. Stanford University, 1998.

[305] Eric Veach and Leonidas J Guibas. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76, 1997.

[306] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Path replay backpropagation: differentiating light paths using constant memory and linear time. *ACM Trans Graph*, 40(4), 2021.

[307] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG)*, 41(4): 1–18, 2022.

[308] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[309] Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. On-line learning of parametric mixture models for light transport simulation. *ACM Trans. Graph.*, 33(4), 2014.

[310] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007.

[311] Xiaotao Wan, Joseph F Pekny, and Gintaras V Reklaitis. Simulation-based optimization with surrogate models—application to supply chain management. *Computers & chemical engineering*, 29(6):1317–1328, 2005.

[312] Shaofei Wang, Marko Mihajlovic, Qianli Ma, Andreas Geiger, and Siyu Tang. MetaAvatar: Learning animatable clothed human models from few depth images. *arXiv:2106.11944*, 2021.

[313] Zian Wang, Jonah Philion, Sanja Fidler, and Jan Kautz. Learning indoor inverse rendering with 3d spatially-varying lighting. In *CVPR*, 2021.

[314] Gregory J Ward. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 265–272, 1992.

[315] Henrique Weber, Donald Prévost, and Jean-François Lalonde. Learning to estimate indoor lighting from 3d objects. In *3DV*, 2018.

[316] Xin Wei, Guojun Chen, Yue Dong, Stephen Lin, and Xin Tong. Object-based illumination estimation with rendering-aware neural networks. In *ECCV*, 2020.

[317] Michael Weinmann, Juergen Gall, and Reinhard Klein. Material classification based on training data synthesized using a btf database. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part III 13*, pages 156–171. Springer, 2014.

[318] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[319] Chenghao Wu, Zahra Montazeri, and Tobias Ritschel. Learning to rasterize differentiable. *arXiv preprint arXiv:2211.13333*, 2022.

[320] Chris Wyman and Scott Davis. Interactive image-space techniques for approximating caustics. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 153–160, 2006.

[321] Yang Xiang, Sylvain Gubian, Brian Suomela, and Julia Hoeng. Generalized simulated annealing for global optimization: the gensa package. *R J.*, 5(1):13, 2013.

[322] Jiankai Xing, Fujun Luan, Ling-Qi Yan, Xuejun Hu, Houde Qian, and Kun Xu. Differentiable rendering using RGBXY derivatives and optimal transport. *ACM Trans. Graph.*, 41(6):1–13, 2022.

[323] Kai Yan, Fujun Luan, Miloš Hašan, Thibault Groueix, Valentin Deschaintre, and Shuang Zhao. Psdr-room: Single photo to scene using differentiable rendering. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–11, 2023.

[324] Yuting Yang, Connelly Barnes, Andrew Adams, and Adam Finkelstein. A δ: autodiff for discontinuous programs-applied to shaders. *ACM Transactions on Graphics (TOG)*, 41(4):1–24, 2022.

[325] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Trans Graph*, 38(6), 2019.

[326] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. Inverse global illumination: Recovering reflectance models of real scenes from photographs. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 215–224, 1999.

[327] Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. Monte carlo estimators for differential light transport. *ACM Trans Graph*, 40(4), 2021.

[328] Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. A differential theory of radiative transfer. *ACM Trans. Graph.*, 38(6), 2019.

[329] Cheng Zhang, Bailey Miller, Kan Yan, Ioannis Gkioulekas, and Shuang Zhao. Path-space differentiable rendering. *ACM Trans. Graph.*, 39(4), 2020.

[330] Cheng Zhang, Zhao Dong, Michael Doggett, and Shuang Zhao. Antithetic sampling for monte carlo differentiable rendering. *ACM Trans. Graph.*, 40(4), 2021.

[331] Jiaxin Zhang, Hoang Tran, Dan Lu, and Guannan Zhang. A scalable evolution strategy with directional gaussian smoothing for blackbox optimization. *arXiv preprint*, 2020.

[332] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Ic-light github page, 2024.

[333] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[334] Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *arXiv:2106.01970*, 2021.

[335] Quan Zheng and Matthias Zwicker. Learning to importance sample in primary sample space. 38(2), 2019.

[336] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11396–11404, 2020.

[337] Shilin Zhu, Zexiang Xu, Henrik Wann Jensen, Hao Su, and Ravi Ramamoorthi. Deep kernel density estimation for photon mapping. In *Comp. Graph. Forum. (proc. EGSR)*, volume 39, 2020.

[338] Shilin Zhu, Zexiang Xu, Tiancheng Sun, Alexandr Kuznetsov, Mark Meyer, Henrik Wann Jensen, Hao Su, and Ravi Ramamoorthi. Photon-driven neural reconstruction for path guiding. *ACM Trans. Graph.*, 41(1), 2021.

[339] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019.