Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering

MERLIN NIMIER-DAVID, École Polytechnique Fédérale de Lausanne (EPFL)
SÉBASTIEN SPEIERER, École Polytechnique Fédérale de Lausanne (EPFL)
BENOÎT RUIZ, École Polytechnique Fédérale de Lausanne (EPFL)
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL)

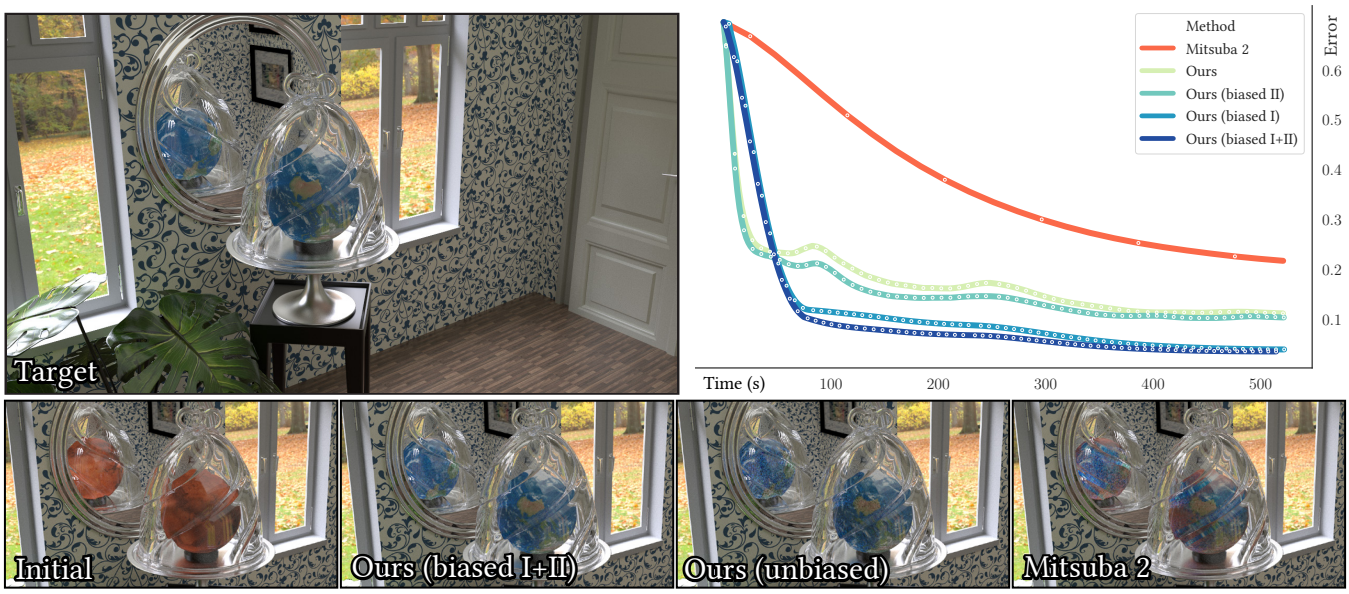


Fig. 1. Globe: Our method is able to reconstruct the texture of a globe seen through a bell jar in this interior scene with complex materials and interreflection. Starting from a different initialization (Mars), it attempts to match a reference rendering by differentiating scene parameters with respect to L_2 image distance. The plot on the right shows convergence over time for prior work [Nimier-David et al. 2019] and multiple variants of radiative backpropagation. Our method removes the severe overheads of differentiation compared to ordinary rendering, and we demonstrate speedups of up to ~1000× compared to prior work.

Physically based differentiable rendering has recently evolved into a powerful tool for solving inverse problems involving light. Methods in this area perform a differentiable simulation of the physical process of light transport and scattering to estimate partial derivatives relating scene parameters to pixels in the rendered image. Together with gradient-based optimization, such algorithms have interesting applications in diverse disciplines, e.g., to improve the reconstruction of 3D scenes, while accounting for interreflection and transparency, or to design meta-materials with specified optical properties.

Authors' addresses: Merlin Nimier-David, École Polytechnique Fédérale de Lausanne (EPFL), merlin.nimier-david@epfl.ch; Sébastien Speierer, École Polytechnique Fédérale de Lausanne (EPFL), sebastien.speierer@epfl.ch; Benoît Ruiz, École Polytechnique Fédérale de Lausanne (EPFL), benoit.ruiz@epfl.ch; Wenzel Jakob, École Polytechnique Fédérale de Lausanne (EPFL), wenzel.jakob@epfl.ch.

0730-0301/2020/7-ART146 $15.00
https://doi.org/10.1145/3386569.3392406

The most versatile differentiable rendering algorithms rely on reverse-mode differentiation to compute all requested derivatives at once, enabling optimization of scene descriptions with millions of free parameters. However, a severe limitation of the reverse-mode approach is that it requires a detailed transcript of the computation that is subsequently replayed to back-propagate derivatives to the scene parameters. The transcript of typical renderings is extremely large, exceeding the available system memory by many orders of magnitude, hence current methods are limited to simple scenes rendered at low resolutions and sample counts.

We introduce *radiative backpropagation*, a fundamentally different approach to differentiable rendering that does not require a transcript, greatly improving its scalability and efficiency. Our main insight is that reverse-mode propagation through a rendering algorithm can be interpreted as the solution of a continuous transport problem involving the partial derivative of radiance with respect to the optimization objective. This quantity is "emitted" by sensors, "scattered" by the scene, and eventually "received" by objects with differentiable parameters. Differentiable rendering then decomposes into two separate primal and adjoint simulation steps that scale to complex scenes rendered at high resolutions. We also investigated biased variants of this algorithm and find that they considerably improve both runtime and convergence speed. We showcase an efficient GPU implementation of radiative backpropagation and compare its performance and the quality of its gradients to prior work.

CCS Concepts: • **Computing methodologies → Rendering**.

Additional Key Words and Phrases: Ray Tracing, Global Illumination, Differentiable Rendering

ACM Reference Format:
Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. 2020. Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering. *ACM Trans. Graph.* 39, 4, Article 146 (July 2020), 15 pages. https://doi.org/10.1145/3386569.3392406

1 INTRODUCTION

Consider the task of determining the material or shape of an object from a set of images. This is a surprisingly difficult problem, since the pixel values encode a complex superposition of indirect effects: for instance, a part of the object could appear brighter because a neighboring object reflects light towards it, or it could appear darker because another object casts a shadow. To solve this task, it is therefore not enough to focus on the object of interest alone–we need to understand its relationship to the surrounding environment as well.

In the context of rendering, this type of radiative coupling between objects has been a crucial ingredient in the quest for photorealism, and a great variety of physically-based methods that simulate the underlying principles of light transport and scattering have been proposed in the past decades [Pharr et al. 2016]. These methods solve an integral equation describing the energy balance of light and tend to be fairly expensive–hours of rendering time for a single image are not uncommon.

At a high level, a rendering algorithm can be interpreted as a function $f : \mathcal{X} \to \mathcal{Y}$, whose input encodes the shape and materials of objects. Evaluating $\mathbf{y} = f(\mathbf{x})$ yields a rendered image that accounts for effects such as shadowing, specular interreflection, or subsurface scattering. Such interactions obscure the individual properties of objects, hence an inverse $f^{-1}(\mathbf{y})$ to recover scene parameters $\mathbf{x}$ from an image $\mathbf{y}$ is normally not available.

This paper is concerned with differentiable rendering techniques, which are increasingly relevant in systems that perform such an inversion. The main distinction of differentiable rendering compared to "ordinary" rendering methods is the availability of derivatives $\partial\mathbf{y}/\partial\mathbf{x}$ relating inputs and outputs of the simulation. Combined with a first-order optimization algorithm, a differentiable renderer is then able to navigate the parameter space to find solutions to different types of inverse problems. This is not a panacea: optimization objectives can be highly non-convex, requiring significant attention to initial guesses and regularization to obtain satisfactory results. In this article, we focus primarily on the mechanics of differentiation in the context of rendering rather than general optimization-related challenges. Our implementation of Radiative Backpropagation currently targets shading and emission-related parameters and does not support gradients that arise due to visibility-related discontinuities. That said, we believe that our technique could be extended to such cases and include a discussion in Section 3.6.

Note that the spaces $\mathcal{X}$ and $\mathcal{Y}$ are generally high-dimensional (10^5–10^8D), hence direct evaluation of the Jacobian $\mathbf{J}_f := \partial f(\mathbf{x})/\partial\mathbf{x}$ is infeasible[1]. Instead, all previously proposed rendering techniques compute simpler matrix-vector products that fall into one of two categories: *forward-mode* methods evaluate $\boldsymbol{\delta}_\mathbf{y} = \mathbf{J}_f\, \boldsymbol{\delta}_\mathbf{x}$ to determine how an infinitesimal perturbation $\boldsymbol{\delta}_\mathbf{x}$ of the scene parameters changes the output image. In contrast, *reverse-mode* approaches use a transpose product $\boldsymbol{\delta}_\mathbf{x} = \mathbf{J}_f^T\, \boldsymbol{\delta}_\mathbf{y}$ to determine how the scene parameters should be updated (to first order) to realize a desired change $\boldsymbol{\delta}_\mathbf{y}$ to the output image. While most differentiable rendering techniques proposed so far are in the former category, the latter approach is preferable for solving inverse problems, especially when the parameter $\mathcal{X}$ space has more than a few dimensions[2].

However, one major downside of the reverse-mode approach is that it requires a detailed transcript of intermediate computation steps to back-propagate derivatives to the scene parameters. For an hour-long rendering, this data structure would be staggeringly large, exceeding the available system memory by many orders of magnitude. Simple scenes rendered at low resolutions and sample counts can fit, though transcript-related memory traffic remains a severe performance bottleneck even in such cases.

This article introduces *radiative backpropagation*, a new technique for differentiable rendering that addresses these limitations. Despite operating in reverse mode, our method does not require a transcript of intermediate steps, allowing it to scale to complex and long-running simulations. Our key observation is that back-propagating derivatives through a rendering algorithm can be re-cast as the solution of a modified light transport problem involving the partial derivative of radiance with respect to the optimization objective. Importantly, this new problem can be solved separately from the original rendering problem, i.e., without retaining intermediate state.

Following a theoretical discussion, we propose a practical and efficient way of implementing our algorithm. Building on top of Mitsuba 2, we use its tracing just-in-time compiler and automatic differentiation backend to produce OptiX "derivative shaders" of individual system components (e.g. BSDFs, light sources, volumes, etc.) that can be attached to scene objects to enable efficient GPU-accelerated radiative backpropagation. We conclude with a demonstration of the superior performance and scalability of our method compared to prior work on differentiable rendering. An open source implementation of our method will be released at https://rgl.epfl.ch/publications/NimierDavid2020Radiative.

2 BACKGROUND AND RELATED WORK

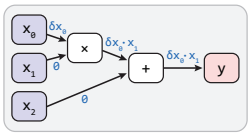
We now review relevant background material and related work.

Differentiation of numerical algorithms. The forward and reverse modes mentioned in the introduction are standard techniques used to transform *primal* computations into programs that evaluate corresponding derivatives. This can be done manually (i.e. with hand-written derivative code), or using software-assisted code transformations that are collectively known as *automatic differentiation* (AD). We refer to the excellent book of Griewank and Walther [2008] for a thorough review of AD. In the following, we briefly contrast forward- and reverse-mode differentiation of a simple example function $f(x_0, x_1, x_2) := x_0 \cdot x_1 + x_2$.

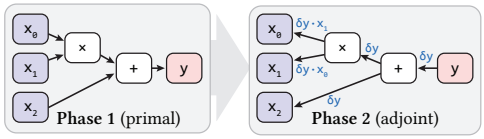
[1] The Jacobian also lacks sparsity: due to interreflection, scene parameters tend to have a nonzero effect on many pixels.

[2] The runtime of forward-mode methods grows linearly with the dimension of $\mathcal{X}$.

- *Forward mode* propagates derivatives from inputs to outputs by repeated application of the chain rule. Below, it propagates a perturbation δx_0 along edges, resulting in the partial derivative $\partial f/\partial x_0 = x_1$. More generally, it can evaluate arbitrary directional derivatives $\mathbf{J}_f\,\boldsymbol{\delta}_\mathbf{x}$ using a single pass, but separate derivatives with respect to individual inputs require multiple passes and are therefore costly.



- *Reverse mode* (or *backpropagation* in the context of neural networks) evaluates the chain rule in reverse, from outputs to inputs, and is ideal for optimization of functions with one output and many inputs. Below, it computes $\operatorname{grad} f(\mathbf{x}) = (x_1, x_0, 1)$.



  When f is multi-valued, reverse mode can evaluate arbitrary directional derivatives $\mathbf{J}_f^T\,\boldsymbol{\delta}_\mathbf{y}$ with respect to perturbations of the outputs $\boldsymbol{\delta}_\mathbf{y}$ in a single pass.
  The program execution order must be reversed during the propagation of derivatives, which is generally accomplished by recording a transcript of operations (also known *Wengert tape* [1964] or *computation graph*) during the primal phase that is replayed in a subsequent adjoint phase. As part of this process, each primal variable is associated with a corresponding *adjoint variable* (e.g. $\delta x_0 = \delta y \cdot x_1$) that tracks its sensitivity with regard to perturbations of the output.

Maintenance and traversal of the transcript adds considerable additional runtime cost, which can be amortized by differentiating multiple coherent evaluations of f at the same time—Griewank and Walther [2008] refer to this as *vector mode*.

Similar to the work of Li et al. [2018] and Nimier-David et al. [2019] our method also operates in reverse mode. However, whereas these prior works differentiated the entire rendering algorithm using vector mode AD, our proposed implementation applies a scalar form of AD to the individual scene elements (e.g. BSDFs, emitters, etc.). This process furthermore occurs *symbolically* during a pre-process step, transforming primal shaders into *derivative shaders* that are used as part of radiative backpropagation. Importantly, derivative shaders store primal and adjoint variables in processor registers, removing the need for a costly transcript data structure.

Reducing AD memory overheads. For long-running computations, the memory requirements of reverse-mode differentiation are prohibitive. Checkpointing strategies [Volin and Ostrovskii 1985] reduce storage overheads by discarding information that can be recovered later on by repeating parts of the computation, but this introduces considerable additional complexity. Furthermore, the size of the checkpoints themselves can be problematic in vector-mode differentiable renderers, which normally propagate large wavefronts consisting of many millions of rays.

It is interesting to note that an explicit reversal of the execution order may be possible in certain problems, entirely removing the need for a transcript. The *adjoint sensitivity method* [Pontryagin 1962; Andersson 2013] is a well-known example of this idea from the area of optimal control. Here, an ordinary differential equation is integrated up to a certain time in the primal phase. The adjoint phase then solves the same equation once more *backwards in time* to propagate sensitivities with respect to an optimization objective to parameters of the model. This idea has been applied in computer graphics to control the behavior of fluid simulations [McNamara et al. 2004], to retarget motion to elastic robots while minimizing vibration [Hoshyari et al. 2019], and to reduce the cost of training neural residual networks [Chen et al. 2018] in the area of machine learning. At a high level, our approach resembles the adjoint method in that it also transforms derivative propagation from a discrete problem into a continuous formulation that can be sampled in reverse. The specifics of the two approaches are very different, since steady-state light transport lacks a natural time variable.

Differentiable rendering. Beginning with the work on *OpenDR* by Loper et al. [2014], a number of approximate differentiable rendering techniques have been proposed in the field of computer vision. Relying on smooth rasterization of meshes or volumes [Rhodin et al. 2015; Kato et al. 2017; Liu et al. 2019; Petersen et al. 2019], these methods focus on primary visibility without accounting for indirect effects (shadows, interreflection, etc.).

Early uses of differentiation in rendering algorithms include the estimation of spatial and directional gradients for adaptive sampling and interpolation [Ramamoorthi et al. 2007]. Later work has focused on parametric derivatives of physically based simulations to optimize the material properties of volumes [Che et al. 2018; Velinov et al. 2018; Gkioulekas et al. 2016, 2013; Khungurn et al. 2015; Zhao et al. 2016] or surfaces [Li et al. 2018; Azinović et al. 2019]. Geometric derivatives are challenging in this context because surface boundaries introduce discontinuities that can lead to incorrect gradients if precautions are not taken. Li et al. [2018] introduced the first method to correctly account for this effect in the context of physically-based rendering using a novel silhouette edge sampling strategy. Recently, Loubet et al. [2019] proposed an alternative way of dealing with visibility via re-parameterization of light transport integrals, and Zhang et al. [2019] introduced a unified framework for differentiating surfaces and volumes in the presence of discontinuities.

Only two physically based rendering systems proposed thus far perform reverse-mode differentiation: Redner [Li et al. 2018], which is based on hand-written derivative code, and Mitsuba 2 [Nimier-David et al. 2019], which uses AD. Our implementation builds on the latter and significantly improves the runtime and memory cost of derivative propagation.

3 METHOD

Before delving into the specifics of our method, we briefly review the relevant radiative transfer equations and their differential formulations. The former roughly follows the notation of Veach [1997], and the latter is a subset of the framework proposed by Zhang et

al. [2019], which we include here for completeness. To keep the discussion simple, we initially focus on a simplified problem without volumetric effects, and we furthermore ignore derivatives that arise due to parameter-dependent silhouette boundaries. The ideas underlying radiative backpropagation are orthogonal to these aspects, and we refer the reader to Zhang et al. [2019] for a full definition of the differential quantities with all terms. The volumetric analog of radiative backpropagation is derived in the appendix.

Rendering algorithms estimate sets of measurements $I_1, \ldots, I_n$ corresponding to pixels in an image. These measurements are defined as inner products on ray space $\mathcal{A} \times S^2$ involving the incident radiance L_i and the importance function W_k of pixels $k = 1, \ldots, n$:

$$\begin{aligned} I_k &= \langle W_k, L_i \rangle \qquad \textbf{(Measurement)} \\ &= \int_{\mathcal{A}} \int_{S^2} W_k(\mathbf{p}, \boldsymbol{\omega})\, L_i(\mathbf{p}, \boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\omega}^{\perp}\, \mathrm{d}\mathbf{p}. \end{aligned}$$

where $\mathrm{d}\boldsymbol{\omega}^{\perp}$ indicates integration with respect to projected solid angle at $\mathbf{p}$. Radiance is invariant along unoccluded rays, relating L_i to the outgoing radiance at the nearest surface $\mathbf{r}(\mathbf{p}, \boldsymbol{\omega})$ visible along the ray $(\mathbf{p}, \boldsymbol{\omega})$:

$$L_i(\mathbf{p}, \boldsymbol{\omega}) = L_o(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}), -\boldsymbol{\omega}). \qquad \textbf{(Transport)}$$

At surfaces, radiance satisfies an energy balance condition relating sources and sinks, specifically emission $L_e(\mathbf{p}, \boldsymbol{\omega})$, and absorption or reflection of incident illumination modeled via $f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')$, the *Bidirectional Scattering Distribution Function* (BSDF) of the surface at $\mathbf{p}$:

$$L_o(\mathbf{p}, \boldsymbol{\omega}) = L_e(\mathbf{p}, \boldsymbol{\omega}) + \int_{S^2} L_i(\mathbf{p}, \boldsymbol{\omega}')\, f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')\, \mathrm{d}\boldsymbol{\omega}'^{\perp}. \qquad \textbf{(Scattering)}$$

Monte Carlo rendering techniques such as path tracing [Kajiya 1986] recursively sample the above equations using sophisticated importance sampling strategies to obtain unbiased estimates of the measurements $I_1, \ldots, I_n$.

3.1 Differential radiative transfer

We now turn to *differential radiative transfer* by differentiating the left and right hand sides of the preceding three equations with respect to all scene parameters $\mathbf{x} = (x_1, \ldots, x_m)$, resulting in vectorial equations. For notational convenience, we define $\partial_{\mathbf{x}} := \partial / \partial \mathbf{x}$ and assume component-wise multiplication of vector quantities. The resulting equations will relate *differential incident, outgoing, and emitted radiance* $\partial_{\mathbf{x}} L_i$, $\partial_{\mathbf{x}} L_o$, and $\partial_{\mathbf{x}} L_e$. Differential radiance in many ways resembles "ordinary" radiance, and we will simply think of it as another type of radiation that can be transported and scattered.

To start, a differential measurement of pixel k involves a ray-space inner product involving differential incident radiance $\partial_{\mathbf{x}} L_i$ and importance W_k:

$$\partial_{\mathbf{x}} I_k = \int_{\mathcal{A}} \int_{S^2} W_k(\mathbf{p}, \boldsymbol{\omega}) \partial_{\mathbf{x}} L_i(\mathbf{p}, \boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\omega}^{\perp}\, \mathrm{d}\mathbf{p}. \qquad \textbf{(Diff. Measurement)}$$

Here, we have assumed a static sensor[3], hence $\partial_{\mathbf{x}} W_k = 0$. Differentiating the transport equation yields no surprises: differential radiance is transported in the same manner as normal radiance:

$$\partial_{\mathbf{x}} L_i(\mathbf{p}, \boldsymbol{\omega}) = \partial_{\mathbf{x}} L_o(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}), -\boldsymbol{\omega}). \qquad \textbf{(Diff. Transport)}$$

The derivative of the scattering equation is more interesting:

$$\begin{aligned} \partial_{\mathbf{x}} L_o(\mathbf{p}, \boldsymbol{\omega}) = {} & \underbrace{\partial_{\mathbf{x}} L_e(\mathbf{p}, \boldsymbol{\omega})}_{\textbf{Term 1}} \qquad \textbf{(Diff. Scattering)} \\ & + \int_{S^2} \Big[\underbrace{\partial_{\mathbf{x}} L_i(\mathbf{p}, \boldsymbol{\omega}')\, f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')}_{\textbf{Term 2}} \\ & \quad + \underbrace{L_i(\mathbf{p}, \boldsymbol{\omega}')\, \partial_{\mathbf{x}} f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')}_{\textbf{Term 3}} \Big]\, \mathrm{d}\boldsymbol{\omega}'^{\perp}. \end{aligned}$$

The above can be interpreted as another kind of energy balance equation. In particular,

- **Term 1.** Differential radiance is "emitted" by light sources whose brightness changes when perturbing the scene parameters $\mathbf{x}$.
- **Term 2.** Differential radiance "scatters" in the same way as normal radiance, i.e., according to the BSDF of the underlying surface.
- **Term 3.** Differential radiance is additionally "emitted" when the material at $\mathbf{p}$ changes as a function of the scene parameters.

In the following, we will develop tools to sample these equations in reverse mode.

3.2 Optimization using differential transport

Applications of differentiable rendering to optimization problems (e.g. inverse rendering) require an objective function $g : \mathcal{Y} \to \mathbb{R}$ that measures the quality of tentative solutions. This objective could be a simple pixel-wise L_1 or L_2 error or a function with a more complex dependence, such as a Wasserstein metric or a convolutional neural network. Given g, we seek to minimize the concatenation $g(f(\mathbf{x}))$ using an iterative gradient-based optimization technique, bearing in mind that evaluations are necessarily noisy due to the underlying Monte Carlo integration.

It is interesting to note that gradients are a major asset in this context: Jamieson et al. [2012] showed that *derivative free optimization* (DFO) methods can never achieve an optimization error better than $\Omega(1/\sqrt{l})$ when optimizing noisy objectives, where l is the iteration count, and this even applies to methods that estimate derivatives using finite differences. In contrast, the error of gradient-based methods always decreases proportional to $\Theta(1/l)$ when the function is strongly convex[4].

It is tempting to perform differentiable rendering by simply evaluating the composition $g(f(\mathbf{x}))$ and recording a transcript, followed by reverse-mode propagation. The first issue with this approach was noted by Azinović et al. [2019] and can be observed after rewriting the gradient as a product of the Jacobians via the chain rule:

$$\partial_{\mathbf{x}}\, g(f(x)) = \mathbf{J}_{g \circ f}(\mathbf{x}) = \mathbf{J}_g(f(\mathbf{x}))\, \mathbf{J}_f(\mathbf{x}).$$

In practice, both factors above are estimated using Monte Carlo integration. Since this occurs simultaneously using the same samples, the resulting random variables are correlated, meaning that the identity $\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]$ no longer holds, and the resulting gradients are thus biased. The second problem is that the transcript of $g \circ f$ is *even longer* than the already problematic rendering step, especially when the objective function is nontrivial. In contrast, our approach

[3] Compatibility of our approach with existing methods which compute visibility-related gradients is discussed in Section 3.6.

[4] A standard assumption made to analyze the asymptotic behavior of such methods.

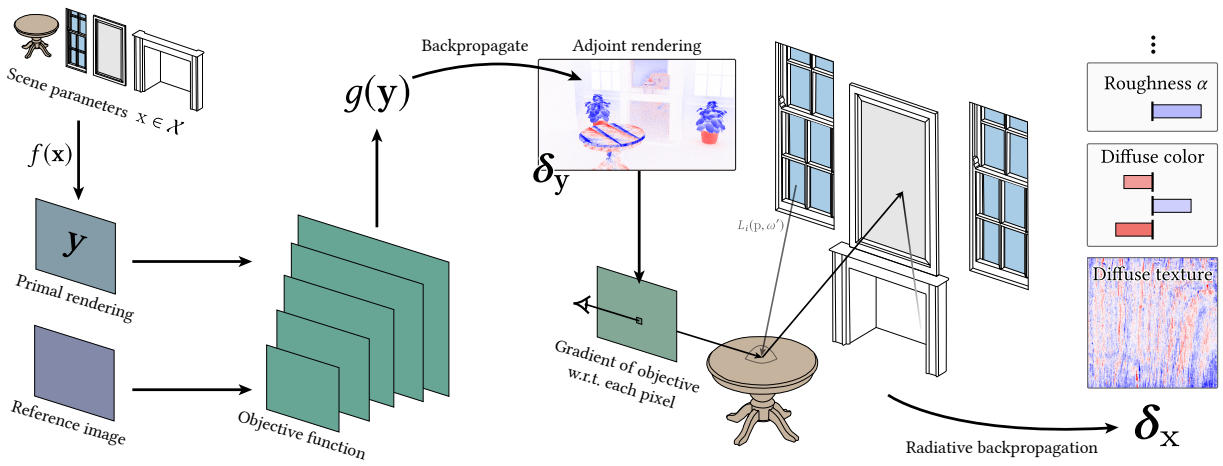


Fig. 2. Our method efficiently computes gradients of an arbitrary objective function $g(f(\mathbf{x}))$ with respect to scene parameters $\mathbf{x}$. At each iteration, it performs a fast primal (i.e. non-differentiable) rendering step producing an image $\mathbf{y} = f(\mathbf{x})$. The second step differentiates the objective function to compute an "adjoint rendering" $\boldsymbol{\delta}_\mathbf{y}$ that expresses the sensitivity of individual image pixels with respect to the optimization task. Radiative backpropagation is the final step and main contribution of this article. It consists of a physical simulation, in which adjoint radiance $\boldsymbol{\delta}_\mathbf{y}$ is emitted by the sensor, scattered by the scene, and eventually received by objects with differentiable parameters. Its output are parameter gradients $\boldsymbol{\delta}_\mathbf{x}$.

splits differentiation into three steps: rendering, differentiation of the objective, and radiative backpropagation. In pseudocode:

```
def grad(x):
    # 1. Ordinary rendering (no AD)
    y = f(x)
    # 2. Differentiate objective at y (manually or w/ AD)
    δ_y = J_g^T(y)
    # Estimate δ_x = J_f^T δ_y using radiative backpropagation
    return radiative_backprop(x, δ_y)
```

We refer to $\boldsymbol{\delta}_\mathbf{y} \in \mathbb{R}^n$ as the *adjoint rendering*[5]. It encodes the sensitivity of pixels with respect to the objective, i.e., how the rendered image should change to optimally improve the objective locally. The algorithm's main steps are illustrated in Figure 2.

3.3 Adjoint radiance

We now turn to the stochastic evaluation of $\mathbf{J}_f^T \boldsymbol{\delta}_\mathbf{y}$. Recall that the rows of the Jacobian $\mathbf{J}_f$ are the parametric derivatives of pixel measurements, i.e.

$$\mathbf{J}_f^T = \left[\partial_\mathbf{x} I_0, \ldots, \partial_\mathbf{x} I_n\right].$$

Substituting the differential measurement equation yields

$$\begin{aligned}\mathbf{J}_f^T \boldsymbol{\delta}_\mathbf{y} &= \sum_{k=1}^{n} \boldsymbol{\delta}_{\mathbf{y},k}\, \partial_\mathbf{x} I_k \\ &= \int_\mathcal{A} \int_{S^2} \underbrace{\left[\sum_{k=1}^{n} \boldsymbol{\delta}_{\mathbf{y},k} W_k(\mathbf{p}, \boldsymbol{\omega})\right]}_{=: A_e(\mathbf{p}, \boldsymbol{\omega})} \partial_\mathbf{x} L_i(\mathbf{p}, \boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\omega}^\perp\, \mathrm{d}\mathbf{p},\end{aligned}$$

where we have defined the *emitted adjoint radiance* $A_e(\mathbf{p}, \boldsymbol{\omega})$ and $\boldsymbol{\delta}_{\mathbf{y},k}$ refers to pixel k of $\boldsymbol{\delta}_\mathbf{y}$. With this substitution, the desired gradient $\mathbf{J}_f^T \boldsymbol{\delta}_\mathbf{y}$ turns into an inner product on ray space:

$$= \langle A_e, \partial_\mathbf{x} L_i \rangle.$$

We think of A_e as an emitted quantity—for instance, in the case of a pinhole camera, it can be interpreted as a textured "spot light" that projects the adjoint rendering into the scene. Whereas a primal renderer might, e.g., estimate an inner product of emitted importance and incident radiance, radiative backpropagation replaces these with differential quantities: emitted adjoint radiance, and the incident differential radiance.

3.4 Operator formulation

The previous expression is reminiscent of the starting point of the operator formulation developed by Arvo [1995] and Veach [1997]. We wish to follow a similar approach here and begin by defining an effective emission term $\mathbf{Q}(\mathbf{p}, \boldsymbol{\omega})$ that includes terms 1 and 3 of the differential scattering equation from the previous page. Recall that the latter is nonzero when the scattered radiance at $\mathbf{p}$ changes as a function of the material parameters.

$$\mathbf{Q}(\mathbf{p}, \boldsymbol{\omega}) := \partial_\mathbf{x} L_e(\mathbf{p}, \boldsymbol{\omega}) + \int_{S^2} L_i(\mathbf{p}, \boldsymbol{\omega}')\, \partial_\mathbf{x} f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')\, \mathrm{d}\boldsymbol{\omega}'^\perp,$$

We then define a *scattering operator* $\mathcal{K}$, and a *propagation operator* $\mathcal{G}$:

$$\begin{aligned}(\mathcal{K}h)(\mathbf{p}, \boldsymbol{\omega}) &:= \int_{S^2} h(\mathbf{p}, \boldsymbol{\omega}')\, f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')\, \mathrm{d}\boldsymbol{\omega}'^\perp, \\ (\mathcal{G}h)(\mathbf{p}, \boldsymbol{\omega}) &:= h(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}), -\boldsymbol{\omega}),\end{aligned}$$

reducing the differential transport and scattering equations to

$$\partial_\mathbf{x} L_i = \mathcal{G} \partial_\mathbf{x} L_o, \quad \text{and} \quad \partial_\mathbf{x} L_o = \mathbf{Q} + \mathcal{K} \partial_\mathbf{x} L_i.$$

Differential radiance scatters and propagates like "ordinary" radiance, hence $\mathcal{K}$ and $\mathcal{G}$ are *identical* to Veach's operators, allowing us

[5] Unless noted otherwise, we use the qualifier "adjoint" in the AD sense throughout this article, i.e., indicating sensitivity with regards to an optimization objective. The terminological overlap with bidirectional light transport techniques is unfortunate.

to immediately state the solution of outgoing differential radiance:

$$\begin{aligned}\partial_{\mathbf{x}} L_o &= \mathbf{Q} + \mathcal{K}\mathcal{G}\,\partial_{\mathbf{x}} L_o \\ &= \underbrace{(I - \mathcal{K}\mathcal{G})^{-1}}_{=:\mathbf{S}}\mathbf{Q} = \sum_{i=0}^{\infty}(\mathcal{K}\mathcal{G})^i\mathbf{Q}\end{aligned}$$

via the solution operator $\mathcal{S}$ given in terms of a Neumann series expansion of $\mathcal{K}$ and $\mathcal{G}$. Veach also showed that $\mathcal{G}$, $\mathcal{K}$, and $\mathcal{GS}$ are *self-adjoint*[6] linear operators when the scene satisfies elementary physical constraints—in particular, energy-conserving and reciprocal BSDFs. Self-adjoint operators O have the property that $\langle O\,v_1, v_2\rangle = \langle v_1, O\,v_2\rangle$, meaning that

$$\mathbf{J}^T\boldsymbol{\delta}_{\mathbf{y}} = \langle A_e, \partial_{\mathbf{x}} L_i\rangle = \langle A_e, \mathcal{GS}\mathbf{Q}\rangle = \boxed{\langle \mathcal{GS}A_e, \mathbf{Q}\rangle.}$$

This equation encapsulates the key idea of radiative backpropagation. It states that instead of scattering and propagating differential radiance we can also start "from the other side" and scatter and propagate adjoint radiance instead. This is a *vastly easier* problem: A_e is scalar-valued, while $\mathbf{Q}$ is a vectorial function whose dimension matches that of the parameter space $\mathcal{X}$ (i.e. potentially millions).

3.5 Sampling strategies for differential rendering

To complete the symmetry, we can finally define analogous incident and outgoing variants of adjoint radiance satisfying

$$A_i = \mathcal{G}A_o, \quad \text{and} \quad A_o = A_e + \mathcal{K}A_i$$

In the next section, we present a simple path tracing-style integrator that samples adjoint incident radiance A_i at surface locations to compute its inner product $\langle A_i, \mathbf{Q}\rangle$ with the effective differential emission. At specific surface positions, the $\mathbf{Q}$ term is extremely sparse[7], admitting a highly efficient integration procedure.

A great variety of alternative sampling schemes are conceivable: for instance, certain scene parameters may have a significant effect on the scene's local radiance distribution (corresponding to very large component values in the effective emission term $\mathbf{Q}$). Scene locations affected by this could be sampled in a more targeted fashion to create an additional connection strategy akin to *next event estimation* in the context of path tracing.

The benefits of such a connection strategy will be marginal if most of the generated samples cannot be connected to the adjoint subpath sampled from the camera end (e.g. due to occlusion). Analogous to *bidirectional path tracing* [Veach and Guibas 1995], it may be helpful to scatter and transport these samples multiple times through the scene to increase the probability of a successful connection, creating a family of bidirectional connection strategies.

One of the main contributions of our work is that it casts the problem of reverse-mode differentiable rendering into familiar light transport terms, enabling the application of a large toolbox of algorithms and sampling strategies developed in the last decades.

[6] In the sense of functional analysis.

[7] On a surface with a textured diffuse BSDF, it will e.g. be zero except for components corresponding to texels that are interpolated during BSDF evaluations at $\mathbf{p}$.

3.6 Loose ends

Dependence on primal radiance. One potential stumbling block is that the effective emission term $\mathbf{Q}$ contains the primal incident radiance L_i. In our implementation, sampling $\mathbf{Q}$ therefore involves a recursive invocation of a classical path tracer. We perform this recursive estimate at every interaction with differentiable parameters, which can be costly (quadratic complexity) when sampling long light paths with many interactions. The same issue was also reported by Zhang et al. [2019] in the context of forward-mode differentiation. Interactions with objects whose properties are not being differentiated do not trigger any additional computation compared to a standard path tracer.

When the scene involves very long light paths and many differentiable objects, it might therefore be preferable to precompute a data structure that enables cheap approximate queries of L_i. For instance, a path guiding technique [Müller et al. 2017] could be used to accelerate rendering in the primal phase. The resulting spatio-directional tree storing an interpolant of L_i could subsequently be used to perform radiance queries in the adjoint phase.

Volumetric scattering. Radiative backpropagation also supports participating media. The theoretical derivation is somewhat technical and can be found in Appendix A.1. Section 5 showcases several results involving optimization of heterogeneous media.

Visibility. Although our prototype implementation of radiative backpropagation does not currently compute visibility-related gradients, we believe that they can be integrated into our framework.

Two very different high-level approaches for differentiating discontinuous light transport integrals have been studied in prior work: Li et al. [2018] propose a new sampling strategy for silhouette edges that accounts for their effect on derivatives. This strategy could in principle also be used during radiative backpropagation, but this would require modifications to the previous theoretical discussion (e.g. extra boundary terms).

Loubet et al. [2019] observed that the problem of incorrect gradients can also be addressed by performing a change of variables in all light transport integrals. In particular, the new coordinates must be carefully chosen so that the discontinuities remain static when perturbing the scene parameters $\mathbf{x}$—naturally, the re-parameterization must itself depend on $\mathbf{x}$ to achieve this. Loubet et al. also propose a simple ray-tracing based query to estimate suitable parameterizations with controllable bias. Such a change of variables could be performed during radiative backpropagation, which requires no modifications of our previous derivations.

3.7 Radiative backpropagation path tracing

Listings 1 and 2 provide the pseudocode of a simple path tracing-style variant of radiative backpropagation. Note that all scene elements (BSDFs, emitters) in these listing are assumed to have an implicit dependence on the scene parameters $\mathbf{x}$. For simplicity, we omit a number of optimizations that are standard in path tracers, and which are similarly straightforward to implement in our method—in particular: russian roulette, direct illumination sampling strategies for emitters, and multiple importance sampling to combine BSDF and emitter sampling strategies—our prototype implementation

```
def radiative_backprop(x, δ_y):
    # Initialize parameter gradient(s) to zero
    δ_x = 0
    for _ in range(num_samples):
        # Importance sample a ray from the sensor
        p, ω, weight = sensor.sample_ray()
        # Evaluate the adjoint emitted radiance
        weight *= A_e(δ_y, p, ω) / num_samples
        # Propagate adjoint radiance into the scene
        δ_x += radiative_backprop_sample(x, p, ω, weight)
    # Finished, return gradients
    return δ_x
```

Listing 1. Radiative backpropagation takes scene parameters $\mathbf{x}$ and an adjoint rendering $\boldsymbol{\delta}_\mathrm{y}$ as input. It samples a large set of camera rays and propagates the associated adjoint radiance A_e (depending on $\boldsymbol{\delta}_y$) into the scene.

```
def radiative_backprop_sample(x, p, ω, weight):
    # Find an intersection with the scene geometry
    p' = r(p, ω)
    # Backpropagate to parameters of emitter, if any
    δ_x = adjoint([[ L_e(p', −ω) ]], weight)
    # Sample a ray from the BSDF
    ω', bsdf_value, bsdf_pdf = sample f_s(p', −ω, ·)
    # Backpropagate to parameters of BSDF, if any
    δ_x += adjoint([[ f_s(p', −ω, ω') ]],
                   weight * L_i(p, ω') / bsdf_pdf)
    # Recurse
    return δ_x + radiative_backprop_sample(
        x, p', ω', weight * bsdf_value / bsdf_pdf)
```

Listing 2. Simplified pseudocode of the propagation operation for surfaces. The two `adjoint()` operations correspond to the two terms of the effective differential emission $\mathbf{Q}$ and propagate adjoint radiance to parameters of the emitters and material models (see Section 3.7 for details on their semantics).

does use these optimizations, however. One interesting optimization opportunity that we currently do not exploit is that many pixels may be associated with a relatively small amount of adjoint radiance (i.e. $A_e(\mathbf{p}, \boldsymbol{\omega}) \approx 0$), in which case rays $(\mathbf{p}, \boldsymbol{\omega})$ should be sampled directly from A_e instead of the sensor's importance function.

Two lines in Listing 2 of the form

```
grad += adjoint([[ q(z) ]], δ)
```

deserve further explanation. This syntax indicates an evaluation of the adjoint of q: we wish to propagate the gradient $\boldsymbol{\delta}$ from the outputs of the function q evaluated at $(\mathbf{x}, \mathbf{z})$ to its differentiable parameters $\mathbf{x} \in \mathcal{X}$ by evaluating $\mathbf{J}_q^T(\mathbf{x}, \mathbf{z})\, \boldsymbol{\delta}$. The function `adjoint` then returns the derivative with respect to the scene parameters i.e. a vector of size $\dim \mathcal{X}$. Note that adjoints of typical components of rendering systems yield *extremely sparse* gradient vectors: for instance, evaluations of a textured BSDF will only depend on a few parameters that correspond to nearby texels. Adding million-dimensional vectors with only a few nonzero entries at every interaction would be very wasteful, hence it is crucial that `adjoint()` exploits the underlying sparsity pattern. The next section discusses our implementation of this important operation.

3.8 Worse is better: acceleration using biased gradients

The previous subsection introduced an unbiased algorithm for estimating parametric derivatives using an optical analog of reverse-mode propagation. We now turn to a counter-intuitive aspect of our method—that a naïve formulation with biased gradients could be preferable! We propose two approximations: the first replaces the effective emission

$$\mathbf{Q}(\mathbf{p}, \boldsymbol{\omega}) = \partial_\mathbf{x} L_e(\mathbf{p}, \boldsymbol{\omega}) + \int_{S^2} L_i(\mathbf{p}, \boldsymbol{\omega}')\, \partial_\mathbf{x} f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')\, \mathrm{d}\boldsymbol{\omega}'^{\perp},$$

by a simplified expression

$$\mathbf{Q}_\mathrm{approx}(\mathbf{p}, \boldsymbol{\omega}) \coloneqq \partial_\mathbf{x} L_e(\mathbf{p}, \boldsymbol{\omega}) + \int_{S^2} \partial_\mathbf{x} f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')\, \mathrm{d}\boldsymbol{\omega}'^{\perp},$$

where we have substituted the primal radiance term L_i with the value 1. In this case, gradients of individual material parameters still record the correct sign (namely, whether the parameter should increase or decrease), but their magnitude will generally be incorrect. This change was originally motivated from an efficiency perspective: using $\mathbf{Q}_\mathrm{approx}$, radiative backpropagation no longer requires recursive invocations of the primal integrator, and the quadratic time complexity thus becomes linear. However, in our experiments, we found that biased gradients are not only faster to compute, but that they paradoxically lead to improved convergence per iteration even when accounting for their different overall scale.

To try to understand potential reasons for this effect, we refer to a recent study of sign-based gradient descent techniques by Balles and Henning [2018]. Comparing SGD to stochastic sign descent, which *only uses the sign* of computed gradients, the authors report: "On the well-conditioned problem, gradient descent vastly outperforms the sign-based method in the noise-free case, but the difference is evened out when noise is added." In our case, $\mathbf{Q}_\mathrm{approx}$ removes a significant source of variance in the radiative backpropagation procedure, which we believe to be responsible for this counter-intuitive improvement. In the remainder of this article, results using this simplification are labeled *Biased (I)*.

Our second approximation builds on the observation that the adjoint phase computes many quantities found in normal path tracers: samples from the BSDF and direct illumination strategies, MIS weights, etc. At negligible additional cost, we can therefore render an image of the scene *while propagating gradients*. This suggests the following iterative scheme with a joint primal and adjoint phase:

```
1 y = f(x)
2 while not converged:
3     δ_y = J_g y
4     y, δ_x = radiative_backprop(x, δy)
5     # .. gradient step ..
```

The above iteration now contains an intentional "off-by-1 error": iteration i propagates the adjoint rendering from iteration $i - 1$

through the Jacobian of iteration i.

$$\boldsymbol{\delta}_{\mathbf{x}}^{(i)} = \mathbf{J}_f^T\left(\mathbf{x}^{(i)}\right)\, \boldsymbol{\delta}_{\mathbf{y}}^{(i-1)}$$

which can be a good approximation if the Jacobian changes slowly from iteration to iteration. In the context of neural networks, this optimization is known as *pipelining* [Petrowski et al. 1993]. In the remainder of this article, results using pipelining are labeled *Biased (II)*, and results that also use $\mathbf{Q}_{\text{approx}}$ are labeled *Biased (I + II)*.

4 IMPLEMENTATION

We have implemented radiative backpropagation on top of Mitsuba 2 [Nimier-David et al. 2019], a retargetable rendering system with several different computational backends and spectral modes. We build on its "`gpu_autodiff_rgb`" target, which performs differentiable simulation on the GPU via CUDA, using OptiX [Parker et al. 2010] for ray tracing along with an RGB-based color representation.

The original codebase propagates wavefronts (typically a few million rays) through the scene one bounce at a time, while keeping a transcript of the underlying calculation in large GPU-allocated arrays to enable reverse-mode automatic differentiation. A key issue with this approach is that rendering simple scenes can easily exhaust the memory capacity of currently available GPUs (we use a NVIDIA RTX 2080 Ti card with 12 GiB of RAM) depending on the resolution and number of samples per pixel (e.g. 512×512 rendered at more than 16 spp for the Globe scene shown in Figure 1). However, scenes with complex radiative transport may require many more samples to obtain reasonably converged gradients. Even when everything fits, memory latency and bandwidth become critical bottlenecks due to the large amount of data transfer that is needed to write and later read the transcript during differentiation. To achieve reasonable convergence without exhausting memory, it is necessary to average the gradients of many short rendering passes. Alternatively, the optimizer can be adjusted to work with very noisy gradients obtained at low sample counts by taking a larger number of tiny gradient descent steps. In either case, performance suffers due to the limits in memory capacity and bandwidth.

We address these problems by discarding the wavefront approach altogether and instead implement radiative backpropagation as a megakernel, which requires no extra memory apart from a single array that is used to store gradients. This megakernel evaluates adjoints of shaders (BSDFs, light sources, etc.) that are automatically extracted from the their primal counterparts.

4.1 Wavefronts versus Megakernels

Seen from a high level, wavefront-based rendering systems propagate a set of rays in lockstep, using large memory regions to record their current state during this process. Separate steps of the algorithm (ray tracing, scattering, direct illumination sampling, etc.) are implemented as small independent kernels that parallelize over the wavefront and mutate its state in global memory.

As the name suggests, a megakernel instead consists of a single large kernel that includes all components of the original rendering algorithm. This kernel processes one light path at a time, generally using processor registers rather than global memory to record its state. Many instances of this megakernel execute concurrently and asynchronously. Wavefront and megakernel-based rendering each have distinct advantages and disadvantages—we refer to Laine et al. [2013] for a thorough discussion. The recent emergence of GPUs with hardware-accelerated ray tracing functionality has tipped the scales towards the latter approach, since they are highly optimized for megakernels.

Note that Mitsuba 2's previous approach to differentiable rendering would not have been practical to implement as a megakernel due to the need to maintain and traverse a transcript, which involves a graph and numerous auxiliary data structures including red-black-trees and multiple hash tables. The original system used wavefronts, i.e. vectorial AD precisely to amortize the runtime cost of maintaining these data structures, which is impossible in a megakernel because each element is processed individually.

4.2 Transitioning to a megakernel

Turning again to radiative backpropagation in the context of these constraints, we observe that the method mostly behaves like a path tracer, where all remaining challenges related to differentiable rendering have been pushed into the effective emission term $\mathbf{Q}$. If this function could somehow be turned into a "shader" that requires no access to dynamic AD-related data structures, then standard tools for megakernel-based rendering (e.g. OptiX) would be applicable.

Motivated by this idea, we decided to convert Mitsuba 2 from a wavefront architecture into a megakernel, while at the same time adding adjoints of all relevant system components referenced in $\mathbf{Q}$ (emitters, BSDFs, participating media). If carried out manually, both are complex changes that would require a substantial redesign of the system[8]. We instead build on the just-in-time (JIT) compilation approach used by Mitsuba 2 and the underlying Enoki library [Jakob 2019] to automatically perform this transformation. A brief review of this JIT compiler is provided in Appendix B.

In its normal mode of operation, the JIT compiler records a symbolic representation of each instruction until a synchronization point triggers evaluation, at which point queued computations are compiled and executed on the GPU. Our approach is based on a simple modification of this scheme: we simply interrupt the process following code generation and return a string representation the generated program (using NVIDIA's PTX intermediate representation) rather than executing it. These functions can then be attached to scene objects and will trigger computation following ray intersections (in OptiX terminology, this is called a "closest hit program"). In the end, the only manual CUDA implementation remaining is the radiative backpropagation code itself which pulls these fragments together, representing less than a thousand lines of code for the surface case. Listing 3 shows how the primal evaluation routine of a BSDF can be extracted using this simple recording mechanism.

Listing 4 shows how adjoints can be extracted using the same mechanism, which entails four additional steps:

(1) Declaration of an additional input: the *gradient of the output* of the primal function.

[8] We note that source transformation-based AD tools such as ADIFOR [Bischof et al. 1992] or Tapenade [Hascoet and Pascual 2013] could in principle perform the latter step, but their restriction to FORTRAN / plain C fragments makes them challenging to reconcile with the highly object-oriented nature of modern renderers.

```
// Declare inputs that act as placeholders
SurfaceInteraction3f si = enoki::zero<SurfaceInteraction3f>();
Vector3f             wo = enoki::zero<Vector3f>();

enoki::start_recording("bsdf_eval");

// Execute BSDF sampling routine *symbolically*
Spectrum result = bsdf.eval(si, wo);

// Declare inputs and outputs. The PTX string will then expose
// the recorded computation as a function with these arguments
enoki::set_inputs(si, wo);
enoki::set_outputs(result);
enoki::stop_recording();

// ... record other functions ...

const char *ptx_string = enoki::ptx_module();
```

Listing 3. We automatically transform Mitsuba 2 from a wavefront renderer into a megakernel-based architecture. To do so, we make a small modification to Enoki's lazy just-in-time compiler that allows it to briefly pause execution while recording all operations involving GPU arrays into a PTX instruction sequence. This sequence can subsequently be extracted and attached to scene objects. The above example extracts the evaluation routine of a BSDF.

(2) Declaration of differentiable parameters, causing subsequent arithmetic in `bsdf.eval()` to be recorded onto the AD transcript.
(3) Reverse-mode traversal via `enoki::backward()`. This step is also symbolic and appends further instructions to the recording.
(4) Recording of an atomic instruction (`enoki::scatter_add()`) that will accumulate the resulting gradient(s) into the vector $\boldsymbol{\delta}_{\mathbf{x}}$.

The resulting code fragment provides an efficient sparse implementation of the $\boldsymbol{\delta}_x$ `+= adjoint([[ q(z) ]], gradient)` operation discussed in Section 3. Importantly, the machinery of reverse-mode AD is used during code generation, but is no longer needed at evaluation time.

Limitations. There are several limitations to our approach: when components of the renderer execute loops, we can only extract a correct instruction trace if the length of this loop is known or can be bounded. Fortunately, this is the case for standard loop constructs found in renderers: for instance, the number of steps needed for hierarchical sample warping or bisection of a discrete CDF is related to the resolution of the underlying distribution and thus exactly known, and root-finding techniques such as Newton iterations can be bounded with a conservative iteration count.

Since we effectively unroll all computations, our approach is suboptimal for long-running loops. In fact, the heterogeneous volume sampling code of Mitsuba 2, which relies on a ray marching loop, generated an unrolled kernel of more than 10MiB of PTX source. When added to the OptiX megakernel, compilation exhausted the system memory and eventually led to a crash. As a workaround, we selectively re-implemented the relevant loop in CUDA while still relying on automatic extraction of the remaining heterogeneous medium-related functionality (trilinearly interpolation of medium parameters like σ_t).

```
// Declare inputs that act as placeholders
SurfaceInteraction3f si = enoki::zero<SurfaceInteraction3f>();
Vector3f             wo = enoki::zero<Vector3f>();

// The gradient of the function's output
// is an input of the adjoint
Spectrum grad_output = enoki::zero<Spectrum>();

// Reference to a BSDF parameter (e.g. roughness)
Float &param = /* .. */;

enoki::start_recording("bsdf_eval_d");

// We want to keep track of derivatives wrt 'param'
enoki::set_requires_gradient(param);

// Evaluate the BSDF symbolically & record a transcript
Spectrum result = bsdf.eval(si, wo);

// Backpropagate 'grad_output' to 'param'
enoki::set_gradient(result, grad_output);
enoki::backward<Float>();

// The derivative shader only has inputs and accumulates
// its outputs into a global array storing parameter gradients
enoki::set_inputs(si, wo, grad_output)
enoki::scatter_add(/* |δx| entry */, enoki::gradient(param));

enoki::stop_recording();
```

Listing 4. We furthermore use Enoki to differentiate components of the renderer via reverse-mode AD, recording the resulting instruction sequence symbolically. The resulting "adjoint shaders" operate without access to AD-related data structures and atomically accumulate gradients with respect to inputs into a global array.

Like normal rendering, radiative backpropagation is "embarrassingly parallel", and can thus be parallelized over many cores of CPUs/GPUs or even multiple machines. One potential issue in this context is that adjoints of shaders atomically accumulate gradients into a set of global variables, which can lead to memory contention. Allocating dedicated memory regions for each core that are merged at the end of the adjoint phase could potentially alleviate such resource conflicts.

5 RESULTS

We now analyze the performance and correctness of radiative backpropagation in a number of different settings.

Validation. To test the correctness of the computed gradients, we selected several simple cases (diffuse RGB coefficients, heterogeneous medium density, diffuse texture), and compare the gradients of these parameters with respect to the loss generated by our method, Mitsuba 2 (which uses automatic differentiation), as well as finite differences in Figure 3. We generally find good agreement with a small amount of residual noise.

Texture optimization. The following experiments focus on the solution of inverse problems using gradient-based optimization. We

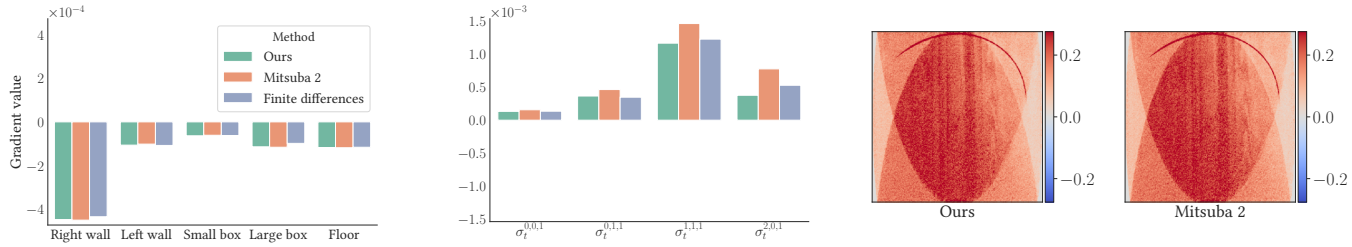


Fig. 3. Validation of gradients comparing radiative backpropagation to Mitsuba 2's existing implementation of automatic differentiation, and finite differences when practical. We plot the partial derivative of the loss w.r.t. each parameter. **(a)**: Gradients of the RGB colors of objects in the Cornell box. **(b)**: Derivatives with respect to extinction in a 3×3 heterogeneous volume lit by an environment map. **(c)**: Gradients of a diffuse texture of a wooden table lit by an area light. In all cases, we find close agreement.

present results for unbiased radiative backpropagation and the two approximations presented in Section 3.8: biased gradients ("biased I"), pipelining ("biased II"), and both ("biased I + II"). As before, the main focus of our work is on efficient differentiation rather than solving specific optimization problems. Additional techniques such as multi-resolution, multi-view and hyper-parameter adjustments are orthogonal to our method and could be used to further improve convergence.

Figure 1 showcases the reconstruction of the texture of a globe encased in a curved sheet glass modeled using two interfaces. This scene involves fine detail, and the underlying simulation accounts for paths with up to 24 scattering events including specular-diffuse-specular interreflection. In such a setting where high resolution and sample count are both required, the AD-based approach of Mitsuba 2 rapidly exhausts the available GPU memory. As a consequence, rendering and differentiation must be split in multiple smaller passes, which negatively impacts the overall running time.

We use a simple L_2 objective function, adding a total variations regularizer to promote smoothness of the texture. Note that any differentiable loss can be used in combination with radiative backpropagation, including more complex ones built from convolutional neural networks. We render each iteration at a resolution of 1280×720 pixels and double the sample count every 40 iterations starting at 4 samples per pixel. Figure 4 compares the convergence of all methods, both with respect to iteration count and runtime. The optimized texture is displayed at equal time, and corresponding renderings are shown in Figure 1.

In principle, there should be no major difference in convergence per iteration when comparing gradients that are obtained using different techniques. Surprisingly, we observe that biased gradients significantly improve convergence although they are clearly "less correct". We believe that the primal radiance estimates performed during our method's adjoint phase introduce large amounts of variance into gradient estimates that impede convergence. Primal radiance estimates are also implicitly used when a complete simulation is differentiated using AD, as is the case in the current implementation of differentiable rendering in Mitsuba 2. By removing this source of variance, faster progress can be achieved.

Volume optimization. Next, we reproduce the volume albedo and density optimization experiments of Nimier-David et al. [2019]. The former optimizes the spatially-varying albedo of a homogeneous volume to match the appearance of a specified texture. This is useful in the context of 3D printing, where the scattering of the material leads to blurring and a loss of contrast. Optimization can be used to determine which color should be used at each point of the medium to maximize contrast and fidelity. This problem was studied in depth in previous work [Elek et al. 2017; Sumin et al. 2019], and we merely use it here as an illustrative example. The second problem reconstructs density values σ_t of a high-resolution smoke plume from an image of the volume itself. Inverting such complex transport effects would be extremely challenging without differentiable rendering.

In both cases, Mitsuba 2 must maintain the transcript of the entire volumetric transport simulation before being able to perform a reverse-mode traversal. This transcript becomes staggeringly large for any non-trivial volume resolution due to a long sequence of trilinear lookups. Despite being rendered at 256×256 resolution, these examples exhaust GPU memory with as few as 1 sample per pixel[9]. In contrast, radiative backpropagation uses a minimal amount of memory and never requires multiple passes. Its memory usage is essentially independent of the volume resolution, as only the volume and its gradients must be stored.

Performance. To quantify the effect of approximate gradients ("biased I") and pipelining ("biased II") optimizations, we measure the runtime of the primal, adjoint or combined primal / adjoint phases in three types of scenes. The breakdown is plotted in Figure 7. Given a scene where all interactions involve differentiable parameters, approximate gradients will help most since they eliminate a component of the runtime that is quadratic in path length. On the other hand, scenes with costly primal rendering phases may benefit most from pipelining.

We then compare our algorithm's runtime to Mitsuba 2 on four typical applications of differentiable rendering in Figure 8. For a given sample count, we run at least 10 iterations and measure the

[9]For this application, we were not able to complete more than 5 iterations with Mitsuba 2 due to a crash. These few iterations took several times longer than the complete runs of our methods combined.

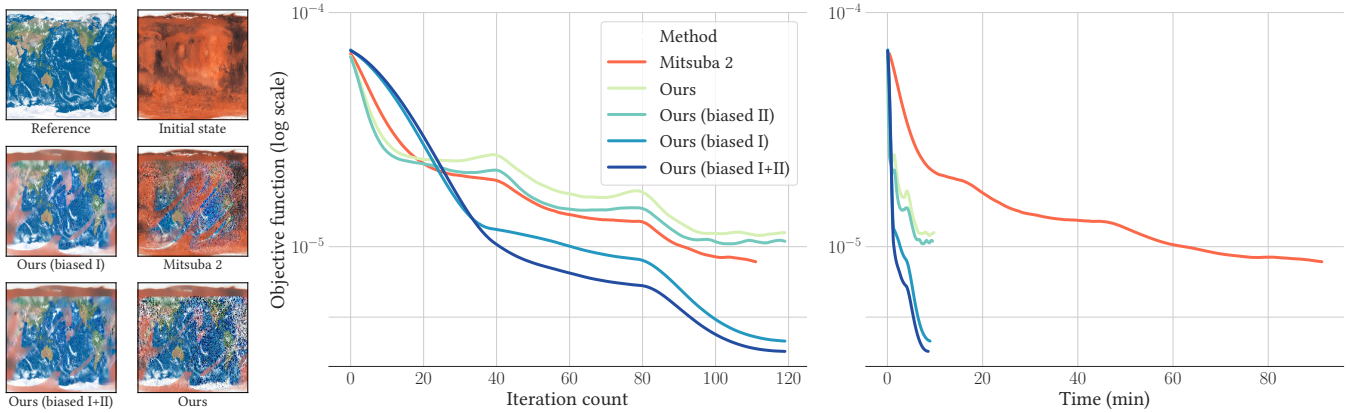


Fig. 4. Recovery of the diffuse globe texture shown in Figure 1 from a single reference image. Despite indirect observation via multiple refractions and reflections in a scene with complex transport, our method is able to closely match the reference image in less than ten minutes (Figure 1 insets). On the left, we show the reconstructed textures after 8.5 minutes. Regions that are not visible (even indirectly) remain close to the initial state. A variant of our technique ("Biased I") removes a significant source of variance by approximating incident radiance with a constant, improving convergence at equal iteration count.

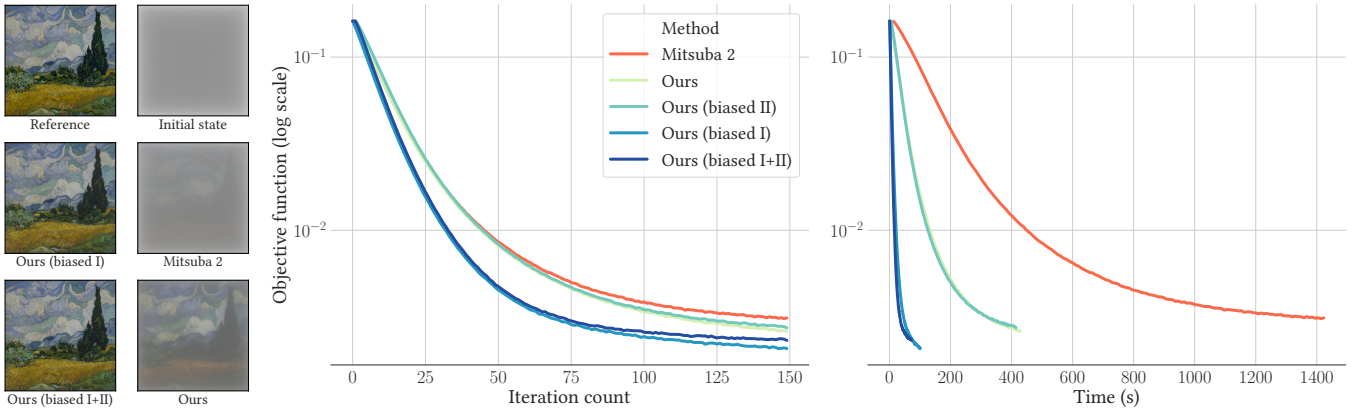


Fig. 5. We reproduce the albedo optimization experiment of Nimier-David et al. [2019], in which the spatially-varying albedo of a homogeneous scattering slab must be modified to match the appearance of a diffuse texture while accounting for subsurface scattering. Starting from a constant gray slab, our method achieves convergence after a single minute of optimization. The resulting media are shown on the left at equal time (74 seconds). In this example, using biased gradients ("Biased I") avoids costly recursive estimation of incident radiance, which dramatically reduces the runtime cost per iteration.

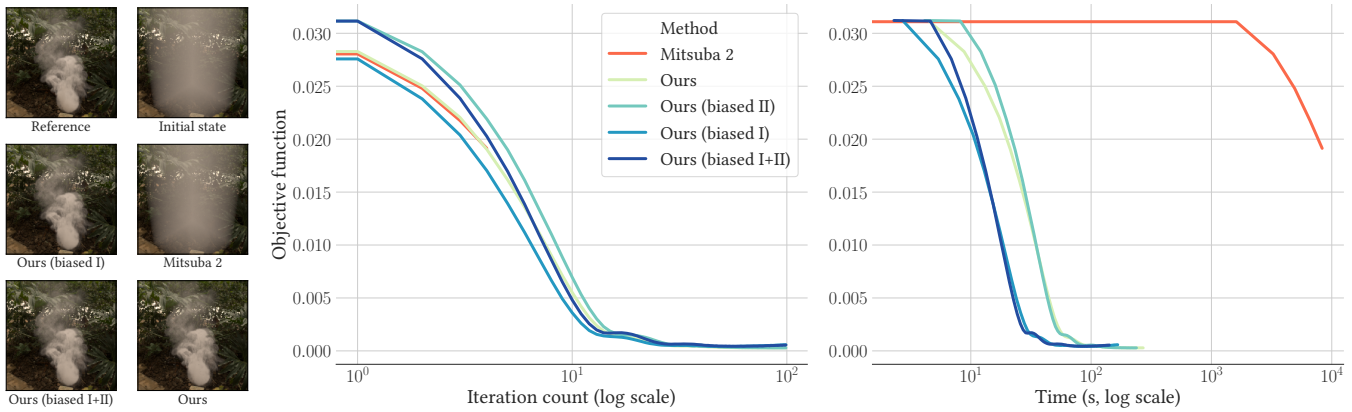


Fig. 6. We also reproduce the volume density optimization experiment of Nimier-David et al. [2019]. In this example, Mitsuba 2's AD-based approach reaches its limits, as the size of the transcript it must store immediately fills the GPU memory, even for low-resolution 128×128 renderings. Insets on the left show convergence at equal time (2.5 minutes), or at the end of the first iteration in case of Mitsuba 2 (26 minutes).

median runtime. We then report the relative speedup of each variant over Mitsuba 2. Both techniques rely on OptiX for ray tracing.

We observe two performance regimes: at low sample counts, even though the full wavefront and AD transcript fit in memory, a constant performance offset arises due to architectural characteristics described in Section 4.1 (wavefront versus megakernel). At higher sample counts, Mitsuba 2's AD approach quickly saturates the available GPU memory, and its computation must therefore be split into several passes. In that regime, we obtain a linear speedup of up to three orders of magnitude. As was shown before, biased variants of our method achieve the highest speedup per iteration, while simultaneously improving convergence in most cases.

These benchmarks were run on an NVIDIA 2080 Ti GPU with 12GiB of RAM. While we do not compare to the Redner differentiable renderer [Li et al. 2018], we expect similar conclusions as it was reported to have performance comparable to Mitsuba 2 [Nimier-David et al. 2019].

6 CONCLUSION

Recent advances have revealed the tremendous potential of differentiable rendering as a tool for solving inverse problems in a variety of scientific disciplines. However, existing approaches to differentiable rendering remain hamstrung by a fundamental performance and scalability bottleneck: reverse-mode differentiation produces vast amounts of intermediate state that rapidly exhausts the memory of even the largest available computing platforms. This currently limits the scope of this technology to relatively simple scenes rendered at low resolutions.

Our article conclusively addresses these limitations. Our main contribution is a new approach to differentiable simulation of light that does not require a transcript of intermediate state, thus avoiding burdensome storage overheads and improving performance by up to three orders of magnitude. This is remarkable because transcript-less differentiation of simulation code was previously only possible in rare cases, one famous example being the adjoint sensitivity method, which relies on the ability to reverse the flow of time.

We show that a different kind of adjoint can also be constructed for steady-state light transport simulations that lack a time dimension. In our case, the adjoint phase emits two differential quantities from sensors and objects that propagate towards each other akin to ordinary light. Their eventual encounter yields a gradient measurement that turns the chain rule from a discrete sum over partial derivatives into a continuous integral on ray space. Another contribution of our work is that it casts the adjoint phase into familiar light transport terms, enabling the application of a large body of prior work on sampling transport integrals.

One current limitation of our work is that the unbiased version of our algorithm has a time complexity that is quadratic in the path length which could become prohibitive (e.g. in highly scattering participating media with light paths involving thousands of interactions). We propose a biased variant that addresses this performance concern and, oddly, appears to generally improve convergence per iteration. Our approach admits an efficient GPU implementation using standard toolkits for megakernel-based rendering, and we demonstrate the automated creation of adjoint shaders building on Mitsuba 2's tracing JIT compiler.

There are many interesting avenues for future work: it would be interesting to see if radiative backpropagation enables simplifications in existing approaches for visibility-related derivatives. Our method currently resembles an ordinary path tracer with multiple importance sampling, and smarter adjoint-specific sampling strategies could be beneficial. Finally, our work focused specifically on the evaluation of gradients, but the underlying optimization problem might be challenging even if perfect gradients were freely available. Further work on techniques to regularize the energy landscape of differentiable rendering is therefore imperative.

ACKNOWLEDGMENTS

We would like to thank Olesya Jakob for designing the scene shown in Figure 1. The diagram in Figure 2 was drawn after the Grey & White room from user Wig42 on BlendSwap. The Mars texture was made available by Solar System Scope. Environment maps from HDRI Haven were used in multiple scenes.

This research was supported by the Swiss National Science Foundation (SNSF) as part of grant 200021_184629.

REFERENCES

Joel Andersson. 2013. *A general-purpose software framework for dynamic optimization.* Ph.D. Dissertation. KU Leuven.

James Arvo. 1995. *Analytic methods for simulated light transport.* Ph.D. Dissertation. Yale University.

Dejan Azinović, Tzu-Mao Li, Anton Kaplanyan, and Matthias Nießner. 2019. Inverse Path Tracing for Joint Material and Lighting Estimation. In *Proceedings of Computer Vision and Pattern Recognition (CVPR), IEEE.*

Lukas Balles and Philipp Hennig. 2018. Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR, 404–413.

Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland. 1992. ADIFOR–generating derivative codes from Fortran programs. *Scientific Programming* 1, 1 (1992), 11–29.

Chengqian Che, Fujun Luan, Shuang Zhao, Kavita Bala, and Ioannis Gkioulekas. 2018. Inverse Transport Networks. *arXiv preprint arXiv:1809.10820* (2018).

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. In *Advances in neural information processing systems.* 6571–6583.

Oskar Elek, Denis Sumin, Ran Zhang, Tim Weyrich, Karol Myszkowski, Bernd Bickel, Alexander Wilkie, and Jaroslav Křivánek. 2017. Scattering-aware Texture Reproduction for 3D Printing. *ACM Transactions on Graphics* 36, 6 (Nov. 2017).

Ioannis Gkioulekas, Anat Levin, and Todd Zickler. 2016. An evaluation of computational imaging techniques for heterogeneous inverse scattering. In *European Conference on Computer Vision.* Springer, 685–701.

Ioannis Gkioulekas, Shuang Zhao, Kavita Bala, Todd Zickler, and Anat Levin. 2013. Inverse Volume Rendering with Material Dictionaries. *ACM Transactions on Graphics* 32, 6, Article 162 (Nov. 2013).

Andreas Griewank and Andrea Walther. 2008. *Evaluating derivatives: principles and techniques of algorithmic differentiation.* Vol. 105. SIAM.

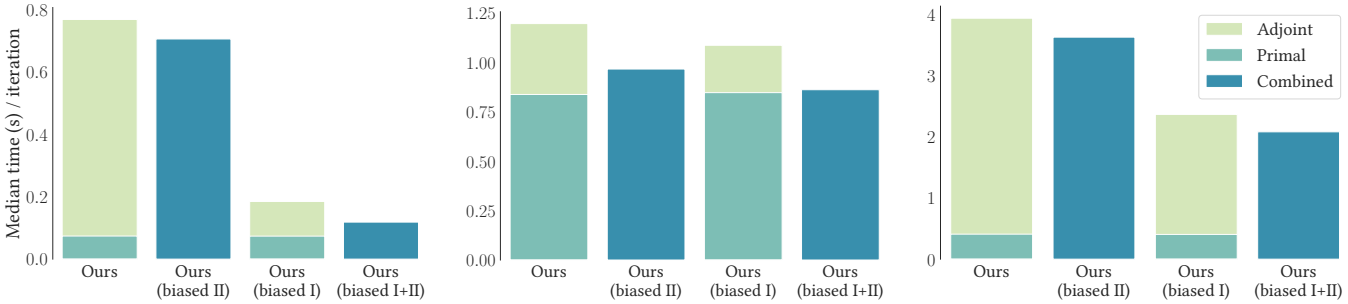
Laurent Hascoet and Valérie Pascual. 2013. The Tapenade automatic differentiation tool: Principles, model, and specification. *ACM Transactions on Mathematical Software (TOMS)* 39, 3 (2013), 20.

Shayan Hoshyari, Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. 2019. Vibration-minimizing motion retargeting for robotic characters. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

Wenzel Jakob. 2013. *Light Transport On Path-Space Manifolds.* Ph.D. Dissertation. Cornell University.

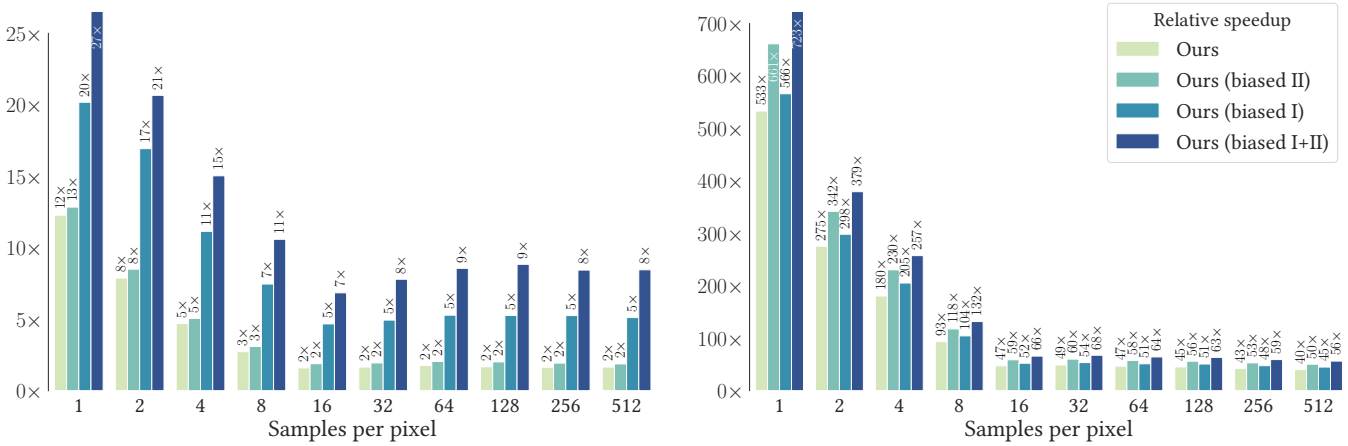
Wenzel Jakob. 2019. Enoki: structured vectorization and differentiation on modern processor architectures. https://github.com/mitsuba-renderer/enoki. (Date accessed: 2020-01-05).

Kevin G Jamieson, Robert Nowak, and Ben Recht. 2012. Query Complexity of Derivative-Free Optimization. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680.

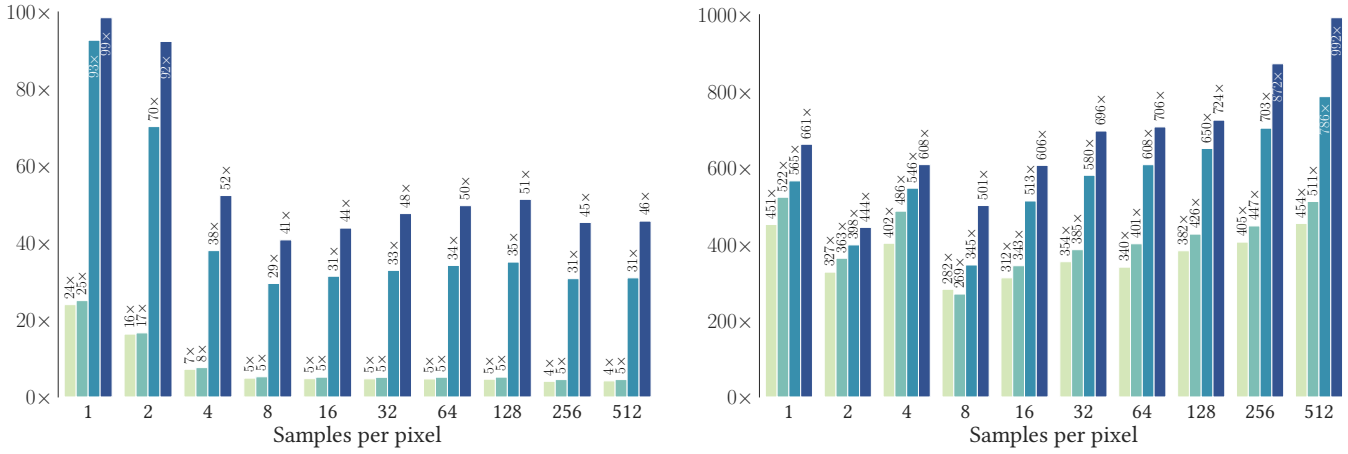


(a) Cornell box (all materials differentiable). (b) Figure 1 scene, optimizing the globe only. (c) Smoke optimization application of Figure 6.

Fig. 7. We break down the running time of radiative backpropagation into primal and adjoint phases. Using biased gradients ("biased I") drastically reduces the cost of the adjoint phase when many differentiable objects are present in the scene (**a** & **c**), while pipelining ("biased II") combines the primal and adjoint phases into a single step (**b**).



(a) Cornell box (all materials differentiable). (b) Figure 1 scene, involving complex light transport.



(c) Optimization of spatially-varying albedo of a homogeneous medium (d) Smoke optimization application of Figure 6.

Fig. 8. We evaluate the relative performance of our method to Mitsuba 2's autodiff-based backend. For each sample count, we time at least 10 iterations of a realistic optimization. We then report the ratio of median iteration runtimes. Radiative backpropagation is up to three orders of magnitude faster.

James T Kajiya. 1986. The rendering equation. In *ACM Siggraph Computer Graphics*, Vol. 20. ACM.
Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2017. Neural 3D Mesh Renderer. *CoRR* abs/1711.07566 (2017). arXiv:1711.07566 http://arxiv.org/abs/1711.07566
Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. 2015. Matching Real Fabrics with Micro-Appearance Models. *ACM Transactions on Graphics* 35, 1 (2015).
Samuli Laine, Tero Karras, and Timo Aila. 2013. Megakernels Considered Harmful: Wavefront Path Tracing on GPUs. In *Proceedings of the 5th High-Performance Graphics Conference (HPG 13)*. Association for Computing Machinery, New York, NY, USA, 137143.
Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable Monte Carlo Ray Tracing Through Edge Sampling. *ACM Transactions on Graphics* 37, 6, Article 222 (Dec. 2018).
Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. 2019. Soft Rasterizer: Differentiable Rendering for Unsupervised Single-View Mesh Reconstruction. *CoRR* abs/1901.05567 (2019). arXiv:1901.05567 http://arxiv.org/abs/1901.05567
Matthew M Loper and Michael J Black. 2014. OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision*. Springer.
Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing Discontinuous Integrands for Differentiable Rendering. *ACM Transactions on Graphics* 38, 6 (Dec. 2019).
Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid control using the adjoint method. In *ACM Transactions On Graphics (TOG)*, Vol. 23. ACM, 449–456.
Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Computer Graphics Forum* 36, 4 (June 2017).
Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Transactions on Graphics* 38, 6 (Dec. 2019).
Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics* 29, 4, Article 66 (July 2010).
Felix Petersen, Amit H. Bermano, Oliver Deussen, and Daniel Cohen-Or. 2019. Pix2Vex: Image-to-Geometry Reconstruction using a Smooth Differentiable Renderer. *CoRR* abs/1903.11149 (2019). arXiv:1903.11149 http://arxiv.org/abs/1903.11149
Alain Petrowski, Gerard Dreyfus, and Claude Girault. 1993. Performance analysis of a pipelined backpropagation parallel algorithm. *IEEE Transactions on Neural Networks* 4, 6 (1993), 970–981.
Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically based rendering: From theory to implementation* (third ed.). Morgan Kaufmann.
Lev Semenovich Pontryagin. 1962. *Mathematical theory of optimal processes*. CRC Press.
Ravi Ramamoorthi, Dhruv Mahajan, and Peter Belhumeur. 2007. A first-order analysis of lighting, shading, and shadows. *ACM Transactions on Graphics (TOG)* 26, 1 (2007), 2.
Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. 2015. A Versatile Scene Model with Differentiable Visibility Applied to Generative Pose Estimation. In *Proceedings of ICCV 2015*.
Denis Sumin, Tobias Rittig, Vahid Babaei, Thomas Nindel, Alexander Wilkie, Piotr Didyk, Bernd Bickel, Jaroslav Křivánek, Karol Myszkowski, and Tim Weyrich. 2019. Geometry-Aware Scattering Compensation for 3D Printing. *ACM Transactions on Graphics* (2019).
Eric Veach. 1997. *Robust Monte Carlo methods for light transport simulation*. Ph.D. Dissertation. Stanford University.
Eric Veach and Leonidas Guibas. 1995. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*. Springer, 145–167.
Zdravko Velinov, Marios Papas, Derek Bradley, Paulo Gotardo, Parsa Mirdehghan, Steve Marschner, Jan Novák, and Thabo Beeler. 2018. Appearance Capture and Modeling of Human Teeth. *ACM Transactions on Graphics* 37, 6, Article 207 (Dec. 2018), 13 pages.
Yu M Volin and GM Ostrovskii. 1985. Automatic computation of derivatives with the use of the multilevel differentiating technique1. Algorithmic basis. *Computers & mathematics with applications* 11, 11 (1985).
R. E. Wengert. 1964. A Simple Automatic Derivative Evaluation Program. *Commun. ACM* 7, 8 (Aug. 1964), 463464. https://doi.org/10.1145/355586.364791
Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. 2019. A Differential Theory of Radiative Transfer. *ACM Trans. Graph.* 38, 6 (2019).
Shaung Zhao, Lifan Wu, Frédo Durand, and Ravi Ramamoorthi. 2016. Downsampling Scattering Parameters for Rendering Anisotropic Media. *ACM Transactions on Graphics* 35, 6 (2016).

A APPENDIX

A.1 Volumetric transport

To analyze the volumetric case, we import the operator formulation of Jakob [2013]. Its generalized scattering operator is given by

$$(\bar{\mathcal{K}}h)(\mathbf{p},\boldsymbol{\omega}) \coloneqq \begin{cases} \displaystyle\int_{S^2} h(\mathbf{p},\boldsymbol{\omega}')\, f_s(\mathbf{p},\boldsymbol{\omega},\boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}'^{\perp}, & \mathbf{p}\in\mathcal{A} \\ \sigma_s(\mathbf{p}) \displaystyle\int_{S^2} h(\mathbf{p},\boldsymbol{\omega}')\, f_p(\mathbf{p},-\boldsymbol{\omega},\boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}', & \mathbf{p}\notin\mathcal{A} \end{cases}$$

which turns incident radiance into either outgoing radiance on surfaces, and into outgoing radiance per unit length in volumes. Here, σ_s is the medium's scattering coefficient and f_p is the phase function[10]. The propagation operator reads

$$(\bar{\mathcal{G}}h)(\mathbf{p},\boldsymbol{\omega}) \coloneqq \int_{\mathbf{p}}^{\mathbf{r}(\mathbf{p},\boldsymbol{\omega})} T(\mathbf{p},\mathbf{p}')\, h(\mathbf{p}',-\boldsymbol{\omega})\,\mathrm{d}\mathbf{p}' + T(\mathbf{p},\mathbf{r}(\mathbf{p},\boldsymbol{\omega}))\, h(\mathbf{r}(\mathbf{p},\boldsymbol{\omega}),-\boldsymbol{\omega}),$$

and returns incident radiance due to both outgoing surface radiance and outgoing radiance per unit length in volumes. The function T is the volumetric transmittance defined as

$$T(\mathbf{a},\mathbf{b}) = e^{-\int_{\mathbf{a}}^{\mathbf{b}} \sigma_t(\mathbf{p})\,\mathrm{d}\mathbf{p}},$$

and references the extinction coefficient σ_t. Using the above definitions, the generalized equilibrium equation[11] reads

$$L_i = \bar{\mathcal{G}}(\bar{\mathcal{K}}L_i + L_e).$$

Differentiating the operators with respect to scene parameters via the product rule yields the following sum of terms:

$$\partial_{\mathbf{x}}(\bar{\mathcal{K}}h)(\mathbf{p},\boldsymbol{\omega}) = \begin{cases} \displaystyle\int_{S^2} \big[\partial_{\mathbf{x}} h(\mathbf{p},\boldsymbol{\omega}')\, f_s(\mathbf{p},\boldsymbol{\omega},\boldsymbol{\omega}') \\ \quad + h(\mathbf{p},\boldsymbol{\omega}')\,\partial_{\mathbf{x}} f_s(\mathbf{p},\boldsymbol{\omega},\boldsymbol{\omega}')\big]\,\mathrm{d}\boldsymbol{\omega}'^{\perp}, & \mathbf{p}\in\mathcal{A} \\ \displaystyle\int_{S^2} \big[\partial_{\mathbf{x}} h(\mathbf{p},\boldsymbol{\omega}')\,\sigma_s(\mathbf{p})\, f_p(\mathbf{p},-\boldsymbol{\omega},\boldsymbol{\omega}') \\ \quad + h(\mathbf{p},\boldsymbol{\omega}')\,\partial_{\mathbf{x}}\sigma_s(\mathbf{p})\, f_p(\mathbf{p},-\boldsymbol{\omega},\boldsymbol{\omega}') \\ \quad + h(\mathbf{p},\boldsymbol{\omega}')\,\sigma_s(\mathbf{p})\,\partial_{\mathbf{x}} f_p(\mathbf{p},-\boldsymbol{\omega},\boldsymbol{\omega}')\big]\,\mathrm{d}\boldsymbol{\omega}', & \mathbf{p}\notin\mathcal{A} \end{cases}$$

and the derivative of the propagation operator is given by

$$\begin{aligned} \partial_{\mathbf{x}}(\bar{\mathcal{G}}h)(\mathbf{p},\boldsymbol{\omega}) = \int_{\mathbf{p}}^{\mathbf{r}(\mathbf{p},\boldsymbol{\omega})} & [\partial_{\mathbf{x}} T(\mathbf{p},\mathbf{p}')\, h(\mathbf{p}',-\boldsymbol{\omega}) + T(\mathbf{p},\mathbf{p}')\,\partial_{\mathbf{x}} h(\mathbf{p}',-\boldsymbol{\omega})]\mathrm{d}\mathbf{p}' \\ & + \partial_{\mathbf{x}} T(\mathbf{p},\mathbf{r}(\mathbf{p},\boldsymbol{\omega}))\, h(\mathbf{r}(\mathbf{p},\boldsymbol{\omega}),-\boldsymbol{\omega}) \\ & + T(\mathbf{p},\mathbf{r}(\mathbf{p},\boldsymbol{\omega}))\,\partial_{\mathbf{x}} h(\mathbf{r}(\mathbf{p},\boldsymbol{\omega}),-\boldsymbol{\omega}). \end{aligned}$$

We observe that both equations contain certain terms with the factor $\partial_{\mathbf{x}} h$, and that the removal of all other terms would yield expressions that the match ordinary scattering and propagation operators applied to the function $\partial_{\mathbf{x}} h$. These other terms only depend on primal quantities (in particular, h) and derivatives of material properties (e.g. $\partial_{\mathbf{x}}\sigma_s(\mathbf{p})$), and the differential form of the operators can thus be cast into the form

$$\begin{aligned} \partial_{\mathbf{x}}\bar{\mathcal{K}}h &= \bar{\mathcal{K}}\partial_{\mathbf{x}} h + \bar{Q}_1 h \\ \partial_{\mathbf{x}}\bar{\mathcal{G}}h &= \bar{\mathcal{G}}\partial_{\mathbf{x}} h + \bar{Q}_2 h \end{aligned}$$

where the operators $\bar{Q}_1$ and $\bar{Q}_2$ model differential radiance that is emitted because optical properties depend on scene parameters $\mathbf{x}$.

[10] Note that its incident argument follows a different sign convention than the BSDF.
[11] The order of operators is reversed compared to the surface case. Details can be found in Jakob's Ph.D. thesis [2013], page 57.

More specifically, $\bar{Q}_1$ describes differential emission due to perturbations in BSDFs, phase functions, and the scattering coefficient, and $\bar{Q}_2$ contains differential emission due to perturbations in transmission along ray segments:

$$(\bar{Q}_1 h)(\mathbf{p}, \boldsymbol{\omega}) = \begin{cases} \displaystyle\int_{S^2} h(\mathbf{p}, \boldsymbol{\omega}')\, \partial_{\mathbf{x}} f_s(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{\omega}')\, \mathrm{d}\boldsymbol{\omega}'^{\perp}, & \mathbf{p} \in \mathcal{A} \\ \displaystyle\int_{S^2} \big[h(\mathbf{p}, \boldsymbol{\omega}')\, \partial_{\mathbf{x}} \sigma_s(\mathbf{p})\, f_p(\mathbf{p}, -\boldsymbol{\omega}, \boldsymbol{\omega}') \\ \quad + h(\mathbf{p}, \boldsymbol{\omega}')\, \sigma_s(\mathbf{p})\, \partial_{\mathbf{x}} f_p(\mathbf{p}, -\boldsymbol{\omega}, \boldsymbol{\omega}')\big]\, \mathrm{d}\boldsymbol{\omega}', & \mathbf{p} \notin \mathcal{A} \end{cases}$$

$$(\bar{Q}_2 h)(\mathbf{p}, \boldsymbol{\omega}) = \int_{\mathbf{p}}^{\mathbf{r}(\mathbf{p}, \boldsymbol{\omega})} \partial_{\mathbf{x}} T(\mathbf{p}, \mathbf{p}')\, h(\mathbf{p}', -\boldsymbol{\omega})\, \mathrm{d}\mathbf{p}' + \partial_{\mathbf{x}} T(\mathbf{p}, \mathbf{r}(\mathbf{p}, \boldsymbol{\omega}))\, h(\mathbf{r}(\mathbf{p}, \boldsymbol{\omega}), -\boldsymbol{\omega}).$$

Both terms can be efficiently sampled using standard volumetric path tracing techniques. We can now derive the differential equilibrium equation:

$$\begin{aligned} \partial_{\mathbf{x}} L_i &= \partial_{\mathbf{x}} \bar{G}(\bar{\mathcal{K}} L_i + L_e) \\ &= \bar{G} \partial_{\mathbf{x}} (\bar{\mathcal{K}} L_i + L_e) + \bar{Q}_2 (\bar{\mathcal{K}} L_i + L_e) \\ &= \bar{G} (\bar{\mathcal{K}} \partial_{\mathbf{x}} L_i + \partial_{\mathbf{x}} L_e + \bar{Q}_1 L_i) + \bar{Q}_2 (\bar{\mathcal{K}} L_i + L_e) \\ &= \bar{G} \bar{\mathcal{K}} \partial_{\mathbf{x}} L_i + \underbrace{\bar{G} \partial_{\mathbf{x}} L_e + \bar{G} \bar{Q}_1 L_i + \bar{Q}_2 (\bar{\mathcal{K}} L_i + L_e)}_{=: \bar{\mathbf{Q}}} \\ &= \underbrace{(I - \bar{G} \bar{\mathcal{K}})^{-1}}_{= \bar{\mathcal{S}}} \bar{\mathbf{Q}}. \end{aligned}$$

where $\bar{\mathcal{S}}$ is the generalized solution operator and $\bar{\mathbf{Q}}$ is an effective emission term that accounts for all sources of emitted differential radiance. As before, differential radiative transfer can thus be understood as the solution of an ordinary transport problem with a modified emission term. The volumetric form of radiative backpropagation then exploits the self-adjoint nature of $\bar{\mathcal{S}}$ to efficiently compute the following inner product:

$$\mathbf{J}^T \boldsymbol{\delta}_{\mathbf{y}} = \langle A_e, \partial_{\mathbf{x}} L_i \rangle = \langle A_e, \bar{\mathcal{S}} \bar{\mathbf{Q}} \rangle = \boxed{\langle \bar{\mathcal{S}} A_e, \bar{\mathbf{Q}} \rangle.}$$

B REVIEW OF THE LAZY JIT AND AD IN MITSUBA 2

Mitsuba's GPU targets perform arithmetic via Enoki's `CUDAArray<T>` type, which lazily fuses operations into larger kernels for later execution on the GPU. For example, line 3 of the C++ fragment:

```
1 using Float = enoki::CUDAArray<float>;
2 Float a = /* .. */, b = /* .. */;
3 Float c = a * b;
```

does not immediately perform a multiplication but rather records that a multiplication should take place when its evaluation can no longer be postponed. Note that `a`, `b`, `c` would typically have millions of entries—one per wavefront element.

Certain operations create barriers that force the system to emit a fused kernel using NVIDIA's PTX intermediate assembly language that is then compiled to machine instructions and executed. An example of this is ray tracing via the OptiX framework, which requires concrete ray origins and directions rather than symbolic descriptions. Communication between separate kernels occurs by reading and writing large GPU-allocated arrays.

In Mitsuba's AD-based targets, the arithmetic types are furthermore wrapped into `DiffArray<T>`, which realizes forward- and reverse-mode differentiation on top of `T`. In the following example,

```
using Float = enoki::DiffArray<enoki::CUDAArray<float>>;
Float a = /* .. */, b = /* .. */;
Float c = a * b;
enoki::backward(c);
```

operations involving `Float` variables are recorded in a transcript used for reverse-mode traversal in `enoki::backward()`. Because `DiffArray<T>` carries out its arithmetic via the underlying type `T`, additional AD-related arithmetic is also fused into PTX kernels. While the above examples were extremely simple, the same principles hold for larger system components: when the system evaluates a rough dielectric microfacet model or samples a direction from an environment map, the implementations return immediately, having recorded their operations symbolically in both AD transcript and fused PTX.

One serious problem with this approach is that the reverse-mode propagation via `enoki::backward()` must occur all the way at the end of differentiable rendering, and the transcript thus becomes very long. Furthermore, ray tracing and virtual function calls constitute two sources of frequent barriers that require flushing queued operations. As a consequence, temporaries and other variables required by `enoki::backward()` can no longer retain their symbolic form—they are evaluated and stored in GPU-resident arrays. Nimier-David et al. [2019] propose to periodically simplify parts of the transcript to reduce memory usage due to these arrays, which helps to a certain extent but ultimately does not solve the issues with memory capacity and bandwidth. Section 4.2 proposes a small modification to Enoki's JIT compiler enabling generation of functions that are usable in a megakernel setting, and which avoid the memory-related issues described above.