Higher-order Differentiable Rendering

Zican (Robert) Wang
University College London
robert.wang.19@ucl.ac.uk

Michael Fischer
University College London
m.fischer@cs.ucl.ac.uk

Tobias Ritschel
University College London
t.ritschel@ucl.ac.uk

Abstract

We derive methods to compute higher order differentials (Hessians and Hessian-vector products) of the rendering operator. Our approach is based on importance sampling of a convolution that represents the differentials of rendering parameters and shows to be applicable to both rasterization and path tracing. We further suggest an aggregate sampling strategy to importance-sample multiple dimensions of one convolution kernel simultaneously. We demonstrate that this information improves convergence when used in higher-order optimizers such as Newton or Conjugate Gradient relative to a gradient descent baseline in several inverse rendering tasks.

1. Introduction

Inverse rendering is concerned with optimizing the parameters of a scene to match a reference. This can be a useful tool when the true parameters of the scene are unknown or non-trivial to set for a human or artist, and therefore need to be inferred from observations or measurements. Examples for such scenarios are multiview-reconstruction [16] or the recovery of illumination and reflectance properties [36]. Inverse rendering works by optimizing an objective function that measures the difference between the current state and the reference. Differentiable rendering has recently become a popular tool for this optimization, as automatic differentiation (AD) frameworks have become more widespread.

However, differentiating the rendering process is far from trivial, as the rendering function [10] has zero and/or undefined gradients and step edges. Moreover, as many rendering operations rely on integration, discontinuities in the rendering pipeline cause problems for AD-engines, as we can no longer naïvely exchange the integral- and derivative-computations. Recent research therefore has designed a plethora of *differentiable* renderers [14], which compute gradients in various ways, typically incurring more [13, 18] or less [5, 7, 8] implementation and compute effort.

A surprising insight is that, while these special renderers allow deriving gradients w.r.t. the scene parameters by differentiating (and back-propagating through) the rendering operator, virtually all of them are limited to first-order derivatives, and hence to expressing the gradient solely locally at a point in the parameter space. At the time of writing, no attempts to derive the Hessians required for higher-order inverse rendering methods have been published. However, decades of optimization research have shown the potential of higher-order methods in convergence and robustness [20].

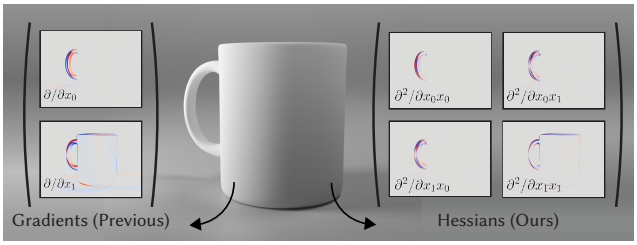


Figure 1. Our approach allows sampling the Hessian for inverse rendering, here for the task of rotating the cup around its horizontal(x_0) and vertical axes (x_1). The estimated positive and negative gradients are shown in blue and red, respectively.

In this paper, we argue that these benefits are also applicable to inverse rendering scenarios and show that they translate to net-gains in optimization time and performance in differentiable rendering. We tackle the Achilles' heel of higher-order optimizers – their increased per-iteration cost and computational / storage requirements – by developing efficient Monte Carlo (MC) estimators of the required higher-order quantities that can be importance-sampled with established techniques [21].

Our method leads to speedups of 2.71× over previous methods, and net-gains in optimization time, speed and robustness while only assuming the rendering operator to be a black box that can be point-sampled.

2. Previous Work

Gradient-based Optimization, the main workhorse of modern neural network training and inverse rendering, uses the gradient of the objective function to take iterative steps in the parameter space towards an improved solution until a

(local) minimum is found. The exact nature of these steps varies with the optimizer that is being used [25].

First-order optimizers are simple to implement and cheap to execute but disregard important higher-order information about the shape of the objective function that could aid optimization, such as the second-order derivative, or curvature. This usually comes at the expense of higher iteration counts, as many small steps are needed to converge to the solution. In contrast, higher-order optimizers incorporate information about the shape of the objective function, which usually allows them to take bigger steps in parameter space, leading to fewer iterations until convergence.

For the specific case of *second-order* optimization, this additional information often is provided via the Hessian H (or approximations thereof) and the Hessian-vector product (HVP). The Hessian contains the second-order derivatives of the objective function w.r.t. the optimization parameters, and can be interpreted as the curvature of the objective function. As such, it can be used to inform the optimizer about how quickly the current gradient is changing and thus, in turn, about how large the optimization step should be.

Notation The following will use lowercase boldface to denote vectors and uppercase sans to denote matrices, respectively. *Operators*, formally defined as functions acting on functions, will be denoted in uppercase Roman lettering in order to avoid confusion with regular functions.

Gradient descent should be familiar to most readers, so we here recall only its terminology: The first-order Taylor expansion of the cost function f at position $\boldsymbol{\theta}$ is

$$f(\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}^t) + g(\boldsymbol{\theta}^t)^\mathsf{T}(\boldsymbol{\theta} - \boldsymbol{\theta}^t),$$

where g is the gradient $\nabla f = \mathrm{d}f/\mathrm{d}\boldsymbol{\theta}$ of f. The minimum is where the derivative is zero, which we can solve for as

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} f(\boldsymbol{\theta}) = 0 \approx g(\boldsymbol{\theta}^t). \tag{1}$$

As f is a linear function only locally, we only make small steps with step size γ, by an update direction $-g(\boldsymbol{\theta}^t)$, as in

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \gamma g(\boldsymbol{\theta}^t).$$

Newton's method is one of the most-used second-order optimizers. This term is derived from the second-order Taylor expansion of the objective f around a point $\boldsymbol{\theta}$:

$$f(\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}^t) + g(\boldsymbol{\theta}^t)^\mathsf{T}(\boldsymbol{\theta} - \boldsymbol{\theta}^t) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^t)^\mathsf{T}\mathsf{H}(\boldsymbol{\theta}^t)(\boldsymbol{\theta} - \boldsymbol{\theta}^t),$$

where H is $\nabla^2 f = \mathrm{d}^2 f/\mathrm{d}^2\boldsymbol{\theta}$, the Hessian of f.

Ideally, we would like our update step to take us to an optimum. There, the derivative necessarily is zero:

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} f(\boldsymbol{\theta}) = 0 \approx g(\boldsymbol{\theta}^t) + \mathsf{H}(\boldsymbol{\theta}^t)(\boldsymbol{\theta} - \boldsymbol{\theta}^t). \tag{2}$$

Solving for $\boldsymbol{\theta}$ then yields the update rule

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \gamma \mathsf{H}^{-1}(\boldsymbol{\theta}^t) g(\boldsymbol{\theta}^t), \tag{3}$$

where $\mathbf{v} = -\mathsf{H}^{-1}(\boldsymbol{\theta}^t) g(\boldsymbol{\theta}^t)$ is called the *Newton direction*.

Newton's method requires the computation, storage and inversion of the full Hessian, which quickly becomes a bottleneck in higher dimensions, as the Hessian for an n-dimensional optimization problem is in $\mathbb{R}^{n \times n}$.

Newton Conjugate Gradient [29], upgrades Newton's method in two ways. First, it solves for Newton's direction iteratively as per the linear equation $\mathsf{H}(\boldsymbol{\theta})\mathbf{v} = -g(\boldsymbol{\theta})$ using conjugate directions [29]. Second, instead of an arbitrary step length γ, it also decides the scalar α by which we move along this direction $\mathbf{v}$. To derive α, first consider the Taylor expansion:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha} f(\boldsymbol{\theta} + \alpha\mathbf{v}) \approx g(\boldsymbol{\theta})^\mathsf{T}\mathbf{v} + \alpha\mathbf{v}^\mathsf{T}\mathsf{H}(\boldsymbol{\theta})\mathbf{v} = 0 \tag{4}$$

which can be re-arranged to

$$\alpha = -\frac{g^\mathsf{T}(\boldsymbol{\theta})\mathbf{v}}{\mathbf{v}^\mathsf{T}\mathsf{H}(\boldsymbol{\theta})\mathbf{v}}. \tag{5}$$

The update rule for Newton Conjugate Gradient is more involved: First, we maintain the direction $\mathbf{v}$ and a residual $\mathbf{r}$, that are initialized to be the gradient at the initial positions:

$$\mathbf{v}^0 = \mathbf{r}^0 = -g(\boldsymbol{\theta}^t) \tag{6}$$

We then find the α by equation Eq. 5 and update the point:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \alpha\mathbf{v}^t. \tag{7}$$

The next direction is chosen by updating the residual $\mathbf{r}$ and computing a new conjugate search direction $\mathbf{v}$ with the Fletcher-Reeves formula [9]:

$$\begin{aligned}
\mathbf{r}^{t+1} &= \mathbf{r}^t - \alpha\mathsf{H}(\boldsymbol{\theta}^t)\mathbf{v}^t \\
\beta &= \frac{\mathbf{r}^{t+1,\mathsf{T}}\mathbf{r}^{t+1}}{\mathbf{r}^{t,\mathsf{T}}\mathbf{r}^t} \\
\mathbf{v}^{t+1} &= \mathbf{r}^{t+1} + \beta\mathbf{v}^t.
\end{aligned}$$

Hessian-vector product approximations go one step further by entirely avoiding to compute H when producing $\mathsf{H}\mathbf{v}$. Pearlmutter [22] and Werbos [33] discuss different options to do so, but a simple option is central differences

$$\mathsf{H}(\boldsymbol{\theta})\mathbf{v} \approx \lim_{\varepsilon \to 0} \frac{g(\boldsymbol{\theta} + \varepsilon\mathbf{v}) - g(\boldsymbol{\theta} - \varepsilon\mathbf{v})}{2\varepsilon}. \tag{8}$$

Hessians in vision and graphics have been shown to be a powerful tool in the past. Chan et al. [3], for instance, achieve faster convergence for total-variation denoising by using a Hessian-based approach. Similarly, [12] utilizes a Hessian-free Newton method in image deconvolution tasks. In computer graphics, [26] uses approximate Hessians to drive a quasi-Newton method for texture parameterization and surface mappings. For 3D shape manipulation, [11] uses a Newton-type optimization scheme to help preserve geometric features during deformation. In the field of optical flow estimation, both [37] and Werlberger et al. [34] make use of second-order information.

Additionally, several works have explored Hessian-based algorithms in machine learning for vision tasks. Yao et al. [35] introduce a Hessian-based pruning method for convolutional neural networkss (CNNs) in image classification, while Ramesh et al. [24] utilize Hessian-based optimization techniques to improve the training of diffusion models for image generation. Additionally, Desai et al. [6] introduces an algorithm that derives Hessians for C code.

Hessians in inverse rendering , however, have received surprisingly little attention, potentially due to their additional implementation overhead and computational complexity. In addition to the fact that conventional AD systems are mostly designed for first-order gradient computations, calculating the second-order information requires storing the whole forward- and first-derivative–graph in memory, which can lead to exponential memory growth. Nicolet et al. [17] are the closest to our work by approximating second-order steps for mesh reconstruction. However, they set the Hessian to the identity to avoid computational expense and instead work with a Laplacian regularizer, which works for their formulation and the case of mesh optimization, but is unclear how to translate to general problems.

For derivative-free gradient estimators, it is equally unclear how second-order information would be computed. The zeroth-order estimators simultaneous perturbation stochastic approximation (SPSA) [30] and finite differences (FD) estimate a first-order gradient, whose second-order derivative naturally is zero. Extending these estimators to second-order information requires prohibitive amounts of function evaluations. ZeroGrads [8], which learns a neural network that fits the cost landscape, uses ReLU non-linearities, whose second-order derivative equally decays to zero.

3. Our approach

We first describe the computation of gradients using importance sampling of a combined gradient-smoothing operator from previous work [4, 7] which we then extend to Hessians and in a next step to Hessian-vector products.

3.1. Background

Rendering equation The rendering equation (RE) [10] describes the radiance L leaving a point $\mathbf{x}$ in the scene into the direction ω_o as

$$L(\mathbf{x}, \omega_\mathrm{o}; \boldsymbol{\theta}) = \int_\Omega \underbrace{f_\mathrm{r}(\omega_\mathrm{i}, \omega_\mathrm{o}) L(\mathbf{y}, \omega_\mathrm{i}; \boldsymbol{\theta})}_{R(\omega_\mathrm{i}; \boldsymbol{\theta})} \mathrm{d}\omega_\mathrm{i}, \tag{9}$$

where $\boldsymbol{\theta}$ are the parameters of the scene we would like to optimize, such as object geometry, reflectance or light emission. This integral over Ω, *i.e.*, all incoming ω_i directions that multiply the radiance field arriving from that direction from the closest other point $\mathbf{y}$ in direction ω_i with the bi-directional reflectance distribution function (BRDF) f_r, has no analytical closed-form solution, and hence usually is approximated – both in forward and inverse rendering – via MC methods. We will shorthand the entire integrand as R.

Problem statement We would now like to apply a differential operator D to the RE, as in

$$\mathrm{D}\, L(\mathbf{x}, \omega_\mathrm{i}; \boldsymbol{\theta}). \tag{10}$$

If D was the gradient operator $\partial L / \partial \boldsymbol{\theta}$, this would be differentiable rendering, for other operators, this becomes higher-order differentiable rendering.

The trouble is that we cannot move D, be it gradient or higher-order, into the integral, as in many cases (*e.g.*, for BRDF or spatial derivatives), the integrand is discontinuous in $\boldsymbol{\theta}$, so

$$\mathrm{D}\, L(\mathbf{x}, \omega_\mathrm{i}; \boldsymbol{\theta}) \neq \int_\Omega \mathrm{D}\, R(\omega_\mathrm{i}; \boldsymbol{\theta}) \mathrm{d}\omega_\mathrm{i}. \tag{11}$$

However, the right-hand side of the above expression is exactly the quantity that naïvely-applied AD computes [19, 32], leading to wrong gradients in (any-order) differentiable rendering.

Solution The idea is to enforce the property that prevents differentiation – smoothness –, so that we actually can differentiate. To that end, assume a further linear operator S that is smoothing any function in $\boldsymbol{\theta}$. This provides a smooth rendering equation $\bar{L}$

$$\bar{L}(\mathbf{x}, \omega_\mathrm{o}; \boldsymbol{\theta}) = \int_\Omega \mathrm{S}\, R(\omega_\mathrm{i}; \boldsymbol{\theta}) \mathrm{d}\omega_\mathrm{i}. \tag{12}$$

Smoothing can be achieved by convolution, so for any function f in any dimension

$$\mathrm{S}\, f(\boldsymbol{\theta}) = \int_\Theta \kappa(\boldsymbol{\tau}) f(\boldsymbol{\theta} - \boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau}, \tag{13}$$

where κ is a smoothing kernel, such as a Gaussian, which we use in this work. This convolved integrand is now

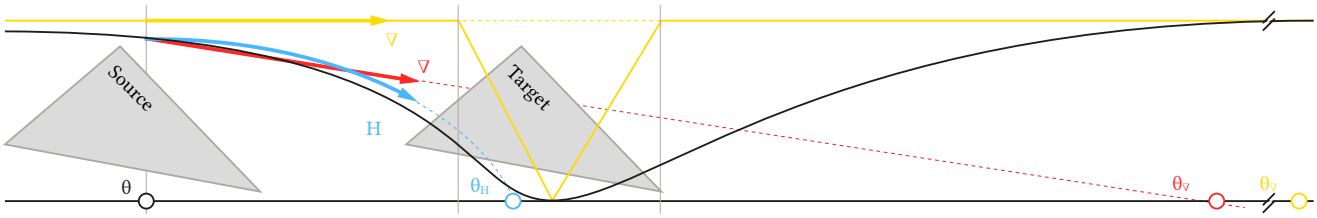


Figure 2. Comparison of classic (yellow) and smooth (red) gradients, as well as our Hessians (blue) on an inverse rendering problem to change the initial parameter $\boldsymbol{\theta}$ so that the left triangle overlaps the right one. Classic gradients are zero almost everywhere (plateaus) except where the triangles already overlap. These methods do not converge. Smooth gradients point into the right direction, but make steps far from the optimum (dotted line). An update taking into account the curvature of the loss landscape lands at a point very close to the target. We incorporate this curvature information via our Hessians.

smooth, which, according to Leibniz' rule, allows us to move the differential operator into the integral

$$\mathrm{D}\,\bar{L}(\mathbf{x},\omega_\mathrm{o};\boldsymbol{\theta}) = \int_\Omega \mathrm{D}\,\mathrm{S}\,R(\omega_\mathrm{i};\boldsymbol{\theta})\mathrm{d}\omega_\mathrm{i}, \tag{14}$$

which, after rearranging and expanding, yields an integral that can be approximated via MC:

$$\mathrm{D}\,\bar{L}(\mathbf{x},\omega_\mathrm{o};\boldsymbol{\theta}) = \int_\Omega \int_\Theta \mathrm{D}\,\kappa(\boldsymbol{\tau})R(\omega_\mathrm{i};\boldsymbol{\theta}-\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\omega_\mathrm{i}. \tag{15}$$

MC here means to take random samples from the product space of light paths and scene parameters. This works best if we can importance-sample for the integrand. The integrand here is a product of four terms. Sampling for the incoming radiance and BRDF terms has been investigated in the rendering community [31] and is not our consideration here, so we simply adopt these strategies. Sampling for the application of the differential operator to the smoothing kernel is the essence of the problem at hand. Depending on the choice of differential operator, we will derive three sampling strategies for the three resulting estimators next.

Conclusion In conclusion, to perform efficient and practical any-order differentiation of the RE, we would need to implement two functions: first, a convolution kernel that combines smoothing and the desired differentiation and second, a function to sample from that kernel for importance sampling. We will now do so for the gradient (Sec. 3.2), Hessian (Sec. 3.3) and HVP (Sec. 3.4).

3.2. Gradients

Operator: For first-order gradient descent, Fischer and Ritschel [7] have differentiated using the gradient operator

$$\mathrm{D}^\mathrm{G} = \nabla = \partial/\partial\tau_i \in (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}^n), \tag{16}$$

which maps a scalar function of n dimensions to an n-dimensional gradient vector field.

The combination of smoothing and differentiation is

$$\mathrm{D}^\mathrm{G}\,\kappa(\boldsymbol{\tau}) = \nabla_i \mathcal{N}(\boldsymbol{\tau},\sigma) = -\frac{\tau_i}{\sigma^2}\cdot\mathcal{N}(\boldsymbol{\tau},\sigma). \tag{17}$$

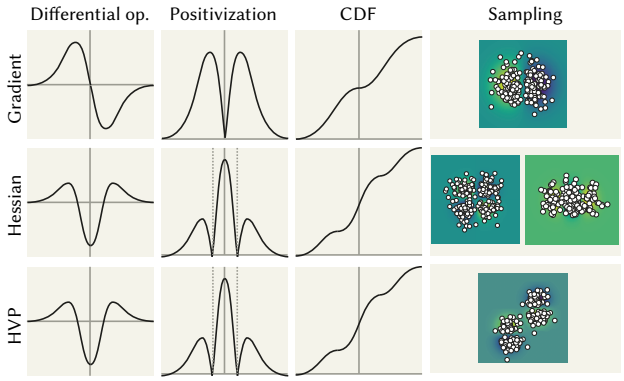


Figure 3. Derivation of smooth differentiation by convolution for three differential operators (rows) involves three steps (columns): Defining the operator to combine smoothing and differentiation (1st col.), positivization and normalization to become a probability density function (PDF) (2nd col.), creating an inverse mapping (3rd col.), which finally allows sampling (4th col.).

For a derivation, please see the supplemental, Sec. A.

Sampling: For sampling a one-dimensional Gaussian gradient, we can use inverse transform sampling via the Smirnov transform [7]. To this end, the integrand has to be a PDF, *i.e.*, positive and integrating-to-1 [21]. Subsequently, we compute the integral of a positivized version of p^G, the cumulative distribution function (CDF) P^G, and invert it as

$$P^{\mathrm{G},-1}(\xi) = \begin{cases} -\sqrt{2\sigma^2\log(2\xi)} & \text{if } \xi \le 0.5, \\ +\sqrt{2\sigma^2\log(1-\xi)} & \text{else.} \end{cases} \tag{18}$$

This derivation applies to the dimension which is being differentiated. Separability of multi-dimensional Gaussians ensures that the other dimensions remain to be a Gaussian distribution, and we can sample these dimensions independently. For a derivation of this result, see supplemental Sec. B.

3.3. Hessians

Operator: The differential operator for Hessians is

$$\mathrm{D}^{\mathrm{H}} = \nabla^2 = \partial^2/\partial\tau_i\partial\tau_j \in (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}^{n\times n}), \tag{19}$$

which maps a scalar function in n dimensions to its $n \times n$-element Hessian field. The combination of second-order derivatives and Gaussian smoothing is

$$\mathrm{D}^{\mathrm{H}}\,\kappa_{i,j}(\boldsymbol{\tau}) = \begin{cases} \left(-\frac{1}{\sigma^2} + \frac{\tau_i^2}{\sigma^4}\right)\cdot\mathcal{N}(\boldsymbol{\tau},\sigma) & \text{if } i = j, \\ \frac{\tau_i\tau_j}{\sigma^4}\cdot\mathcal{N}(\boldsymbol{\tau},\sigma) & \text{else.} \end{cases} \tag{20}$$

For a derivation, see supplemental Sec. C.

Sampling: We first need to positivize as the function is signed, then scale the function so that it is a valid distribution. This is done differently for diagonal and off-diagonal elements. For the diagonal case, we construct the CDF of a second-order derivative P_{ii}^{H} of the 2D Gaussian:

$$P_{ii}^{\mathrm{H}}(\tau_i = u) = \begin{cases} -\frac{u}{4\sigma}\exp\left(\frac{1}{2} - \frac{u^2}{2\sigma^2}\right) & \text{if } u < -\sigma, \\ 0.5 + \frac{u}{4\sigma}\exp\left(\frac{1}{2} - \frac{u^2}{2\sigma^2}\right) & \text{if } u \in [-\sigma,\sigma] \\ 1 - \frac{u}{4\sigma}\exp\left(\frac{1}{2} - \frac{u^2}{2\sigma^2}\right) & \text{if } u > \sigma. \end{cases} \tag{21}$$

See supplemental Sec. D for a derivation of this result. As this is a transcendental equation, there is no analytical form of expressing the inverse CDF [2]. Instead, we 1D-tabulate its values on the range 10σ for every i. The inverse function is found by searching in that range, and storing the pre-sorted inverse indices for $O(1)$ access.

The off-diagonals are the product of two partial gradients that we have already derived in Eq. 17. Fortunately, the product of two independent distributions can be sampled by sampling each one independently. Thus, no extra derivation is required here. For sampling in higher dimensions, similar to the gradient sampler in Sec. B, the rest of the dimensions are sampled from a Gaussian distribution.

In all methods, we exploit the symmetry in the Hessian matrix when sampling and estimating.

3.4. Hessian-vector product

Operator: As explained in Sec. 2, avoiding to store the full Hessian is possible by using HVPs:

$$\mathrm{D}^{\mathrm{HVP}} = \nabla^2\mathbf{v} = \partial^2/\partial\tau_i\partial\boldsymbol{\tau}\cdot\partial/\partial\boldsymbol{\tau}. \tag{22}$$

In essence, a HVP is the directional gradient of the gradient. As there is one direction and the gradient has n dimensions, the HVP is an n-dimensional vector, too. In contrast, a Hessian is the non-directional gradient of the gradient, *i.e.*, the gradient along all n dimensions, and as such in $\mathbb{R}^{n\times n}$.

Sampling: To sample the smooth HVP, we simply need the directional central differences of an estimator of gradients, which we already have. So the estimator is the difference of two first-order estimators, evaluated at positions shifted from the current solution along the gradient direction [22, 33]:

$$\mathrm{D}^{\mathrm{HVP}}\,\kappa(\boldsymbol{\tau}) = \frac{\mathrm{D}^{\mathrm{G}}\,\kappa(\boldsymbol{\tau} - \varepsilon\mathbf{v}) - \mathrm{D}^{\mathrm{G}}\,\kappa(\boldsymbol{\tau} + \varepsilon\mathbf{v})}{2\varepsilon}. \tag{23}$$

3.5. Aggregate

After describing all options, we summarize their key property – the number of function evaluations required – in Tab. 1. Function evaluations require execution of the black-box rendering engine, the most costly part of the optimization, and hence should be minimized as much as possible.

Table 1. Time complexity of different variants of estimators.

	no IS	IS	AIS
Fin. Diff.	n	n	n
FR22 [7]	1	1	1
Our Grad.	1	n	1
Our Hess.	1	n^2	1
Our HVP	1	n	1

Classic finite differences (first row in Tab. 1) take opposing samples in each dimension and hence, for an n-dimensional problem, require $2n$ function evaluations for a single gradient sample. All other methods (second row in Tab. 1 onward) are based on MC, so we can get a gradient estimate using a fixed number of M function evaluations. This, however, comes at the cost of variance, as explained before, which can be reduced with importance sampling, but that is done in all dimensions of the differential quantity independently, so it requires as much evaluations as these have elements (second column in Tab. 1). For a Hessian, this can be substantial, n^2.

As a compromise between low number of function evaluations and low variance, we propose "aggregate" importance sampling of the differential quantity (Fig. 4): instead of importance-sampling each dimensional optimally but individually, we importance-sample w.r.t. the *average* of the convolution across *all* dimensions. This average is a single function again, and with our combined sampling strategy can again be sampled with one function call per iteration (last column of Tab. 1). We implement this in two simple steps: first we roll a dice to decide which element of the differential representation to choose (*e.g.*, which matrix element out of n^2 in the Hessian) and then we compute the convolution sample for all n^2 but with the same function value that is only evaluated once. This leads to an increased variance again, as the average kernel is not identical to the individual kernels but shares many properties, *e.g.*, they are all zero in value at position 0. As a linear combination of unbiased estimators, this is still unbiased. We show in Sec. 4.3 that this pays off. Note, that Tab. 1 is both time

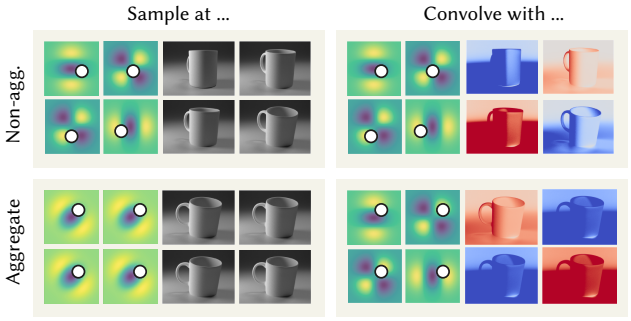


Figure 4. Aggregate and non-aggregate sampling for a 2D optimization space and, consequentially, a 2×2 Hessian: Without aggregation, importance sampling is done for each element of the Hessian independently (four points at different positions in each kernel), leading to four rendering calls and four different teapots. Each value is then weighted by the kernel (blue and red colors denote positive and negative). In aggregate sampling, we importance-sample the average of all four stencils, resulting in a single sample location and hence four times the same teapot that can be rendered in one call, weighted with four different kernels.

and space complexity, except for the aggregate importance sampling of the HVP, where the space complexity stays at $O(n)$ while time complexity is reduced to $O(1)$.

4. Evaluation

4.1. Methods

Our evaluation compares the performance of gradients (input to gradient descent), Hessians and non-aggregate as well as aggregate Hessian-vector product (both input to higher-order optimizers).

Baselines An established inverse rendering solution is `Mitsuba` with gradient descent. `FR22` is using gradient descent with Fischer and Ritschel [7] MC gradients. `OursG`, `OursH`, `OursHVP` and `OursHVPA` are our approach for gradient, Hessians and Hessian-vector products, and aggregates, respectively. `OursG` is used with gradient descent (GD), the others in combination with conjugate gradient (CG). We have also experimented with `LBFGS` as detailed in the supplemental, but do not report it here as it almost never converges and never at competitive speed.

Tasks We tackle both artificial problems with known analytic Hessians and Hessian-vector products as well as real inverse rendering tasks.

As a simple first test case, we optimize the smooth, quadratic potential(QUAD[1]) function $ax_0^2 + bx_1^2 + cx_0x_1$ in $\mathbb{R}^2$, with the fixed variables $a = 5, b = 5, c = 7.5$.

Second, we optimize the classic plateau-demonstration task from [7]: the 2D position of some boxes is optimized to match a reference. This is already a much harder task, as there is a plateau in the cost landscape when the squares do not overlap (almost always the case in the initial configuration), as the image-space error does not change. We study the case of one (BOX2[3]) and five (BOX10[5]) squares, with two and ten dimensions to optimize, respectively.

For real inverse problems we study the optimization of reflectance, light and geometry. These scenes are rendered using Mitsuba. In the MUG[7] task, we optimize the vertical rotation of a coffee cup such that it matches a reference. In the SHADOW[9] task, we optimize the position of a sphere that is unobserved in the rendered image, such that the shadow it casts matches a reference shadow. In the BUNNY[11] task, the x and z position and the rotation around the z axis of the Stanford bunny are optimized so that they match the reference image. This task is designed specifically for `Mitsuba` to converge[12], thus it is setup with no plateaus in the loss function.

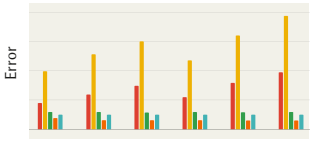
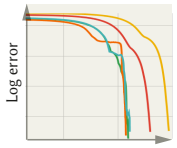
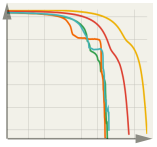
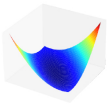
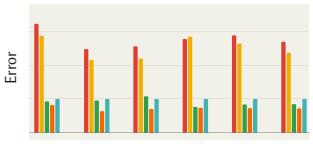
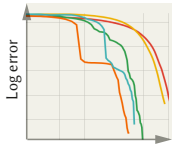
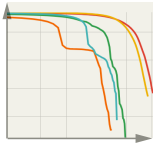
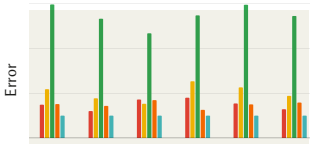
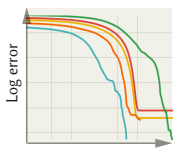
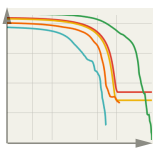
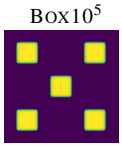
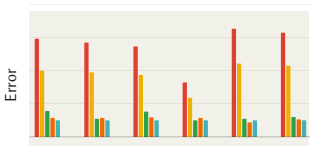
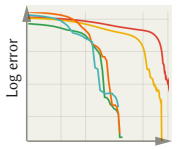
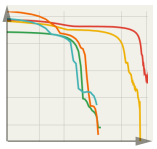
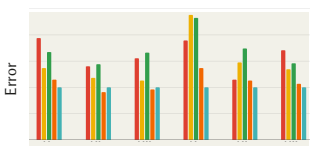
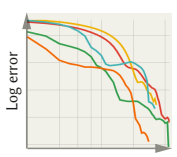
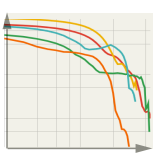
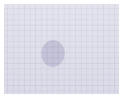
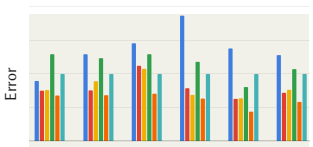
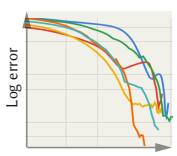
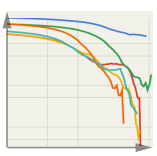
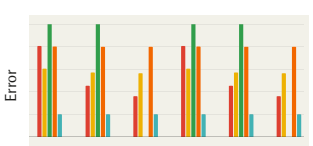
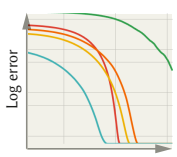
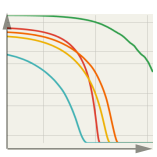
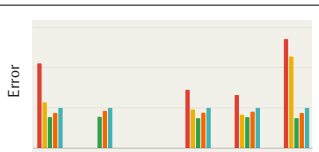
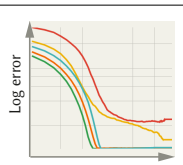
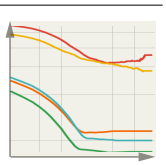
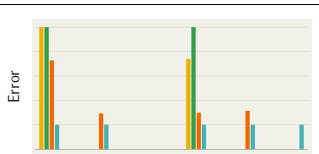
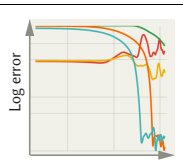
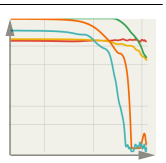
To test the scaling with dimensionality, we also optimize the pixels of a 32×32 texture(TEXTURE[13]), with 1024 parameters.

Finally, we learn a CNN[15] to predict scene parameters from images using inverse rendering without scene parameter labels. In this case, the optimization is learning in the stricter sense: we tune parameters of a deep architecture instead of scene parameters directly. The CNN takes as input an image of a mug and produces as output the orientation for a mug. As a loss, these parameters are inserted into a renderer, differentiated by different approaches and compared to a target. The CNN has 267 745 parameters, learned by differentiating through an image loss and rendering of the scene given the estimated parameters. Another CNN (CNN5[19]) is trained to predict the rotation, position and color of the mug with 268 773 parameters.

Differentials of this function are computed as detailed in supplemental Sec. E, where we derive a grey-box approach that samples only the black-box part (the rendering) and combines its differentials with analytic differentials for the white-box part (the CNN). Importantly, this idea works for gradients, as well as for higher order differentials such as the ones we study here.

Metrics Our main measure of success is convergence speed in wall-clock and the difference of the parameters found to the true optimal parameters (which we know in all inverse rendering tasks as we created the scenes). A secondary measure of success is the image difference, as during optimization, we do not have access to the parameters which are to be considered hidden. All metrics are evaluated across an ensemble of 20 runs averaged across 10 steps in time. In convergence plots, the ensemble median is shown at every point in time, averaged across 20 time steps.

Table 2. Quantitative results of different methods on different tasks (rows) and their convergence plots. We report convergence time in wall-clock units, in ratio to the overall best method, `OurHVPA`. In the numerical columns, .9 and .99 report the time taken to achieve 90 and 99% error reduction from the initial starting configuration, respectively, while the bar plots graphically show these findings. The line plots report image- and parameter-space convergence in the left and right column, respectively, on a log-log scale.

Task	Method	Image error			Parameter error			
		.9	.99	.999	.9	.99	.999	
QUAD[1]	Mitsu	— [2]	—	—	—	—	—	  
	FR22	1.80	2.38	2.98	2.20	3.17	3.90	
	OurG	3.98	5.13	6.02	4.72	6.42	7.77	
	OurH	1.17	1.17	1.15	1.18	1.16	1.17	
	OurHVP	**0.76**	**0.62**	**0.62**	**0.63**	**0.59**	**0.60**	
	OurHVPA	1.00	1.00	1.00	1.00	1.00	1.00	
BOX2[3]	Mitsu	—	—	—	—	—	—	  
	FR22	3.24	2.49	2.57	2.78	2.89	2.70	
	OurG	2.88	2.16	2.21	2.86	2.65	2.38	
	OurH	0.93	0.95	1.08	0.76	0.84	1.84	
	OurHVP	**0.81**	**0.64**	**0.70** [4]	**0.74**	**0.72**	**0.72**	
	OurHVPA	1.00	1.00	1.00	1.00	1.00	1.00	
BOX10[5]	Mitsu	—	—	—	—	—	—	  
	FR22	1.49	1.20	1.71	1.80	1.54	1.29	
	OurG	2.18	1.77	1.53	2.53	2.26	1.88	
	OurH	5.97	5.33	4.68	5.48	5.96	5.45	
	OurHVP	1.51	1.44	1.69[6]	1.26	1.50	1.58	
	OurHVPA	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	
MUG[7]	Mitsu	—	—	—	—	—	—	  
	FR22	5.92	5.69	5.46	3.28	6.53	6.29	
	OurG	3.99	3.91	3.74	2.36	4.42	4.30	
	OurH	1.57	1.09	1.53	1.00	1.08	1.20	
	OurHVP	1.15	1.14	1.18	1.14	**0.87**	1.05	
	OurHVPA[8]	**1.00**	**1.00**	**1.00**	**1.00**	1.00	**1.00**	
SHAD[9]	Mitsu	—	—	—	—	—	—	  
	FR22	1.94	1.40	1.55	1.89	1.14	1.71	
	OurG	1.37	1.18	1.13	2.38	1.47	1.34	
	OurH	1.67	1.44	1.66	2.32	1.74	1.46	
	OurHVP	1.14	**0.91**	**0.96**	1.37	1.13	1.07	
	OurHVPA	**1.00**	1.00	1.00	**1.00**	**1.00**	**1.00**[10]	
BUNNY[11]	Mitsu[12]	0.90	1.29	1.46	1.87	1.38	1.28	  
	FR22	0.75	0.75	1.12	0.78	0.63	0.71	
	OurG	0.76	0.89	1.07	0.69	0.64	0.76	
	OurH	1.29	1.23	1.29	1.18	0.80	1.07	
	OurHVP	**0.68**	**0.68**	**0.70**	**0.63**	**0.44**	**0.58**	
	OurHVPA	1.00	1.00	1.00	1.00	1.00	1.00	
TEXTURE[13]	Mitsu	—	—	—	—	—	—	  
	FR22	4.04	2.27	1.80	4.04	2.27	1.80	
	OurG	3.03	2.86	2.82	3.03	2.86	2.82	
	OurH[14]	40.12	39.78	—	40.12	39.78	—	
	OurHVP	4.01	4.00	4.00	4.01	4.00	4.00	
	OurHVPA	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	
CNN[15]	Mitsu	—	—	—	—	—	—	  
	FR22	2.11[16]	—	—	1.45	1.32	2.71	
	OurG	1.14	—	—	0.96	0.83	2.28	
	OurH	**0.77**	**0.77**	—	**0.74**	**0.77**	**0.74**	
	OurHVP	0.87	0.93[17]	—	0.88	0.91	0.87	
	OurHVPA	1.00	1.00	— [18]	1.00	1.00	1.00	
CNN5[19]	Mitsu	—	—	—	—	—	—	  
	FR22	—	—	—	—	—	—	
	OurG	18.68	—	—	3.70	—	—	
	OurH	19.00	—	—	5.79	—	—	
	OurHVP	3.64[20]	1.46	—	1.49	1.58	—	
	OurHVPA	**1.00**	**1.00**	—	**1.00**	**1.00**	**1.00**	

4.2. Results

The main results of our evaluation are summarize in Tab. 2 where empty cells did not converge or the method is not applicable to that task.

On average across tasks and methods, our premiere method `oursHVPA` speeds up the convergence by a factor of 2.71. This ratio varies between different tasks.

In general, our methods, in particular `OursHVP` and `OursHVPA`, lead the level of error reduction across all time budgets, as seen by comparing the convergence curves vertically where they turn lowest consistently across the horizontal time.

We benefit most in the artificial QUAD task, but MUG and SHADOW are both real rendering problems, where the former is around seven times faster, the latter one only 60 %. We see that `OurG` typically cannot outperform `FR22`, which uses an approximation of the correct high-dimensional gradient kernel only. Doing it "right" only pays off when going to higher order. We also note that `OursHVP` outperforms `OursH`, a gain due to less function evaluations (recall that the metric is wall-clock, the iteration count for both would be very close). The log-log plots show that the higher-order variants enabled by our approach are, as expected, much faster, but also converge slightly less stable.

While tasks like QUAD and BOX2 are not applicable to inverse renderers such as `Mitsuba`[2], our methods treat any loss function as a black-box and retrieve derivatives via sampling.

Comparing the performance of `OurHVP` and `OurHVPA` across a task with two and ten dimensions, we find the expected relation: `OurHVP` is faster in low dimensions[4] than in higher dimensions for a similar task[6].

We notice that the `OurHVPA` is not out-performing the `OurHVP` and `OurH` in the MUG task[8]. This is because there is a single scene parameter for this task, thus, the higher order derivative is all in order of one. With a higher variance for `OurHVPA`, it could be a little slower. The effect of using the aggregate starts to show from SHAD, where `OurHVPA` reaches the $1:1000$ convergence the fastest[10].

For all the rendering tasks, only the BUNNY task converges for `Mitsuba`[12]. Since the other tasks have plateaus in the loss function, the analytical gradient from inverse rendering is effectively zero, so it would be hard to converge within a reasonable time. The log-log plot shows in addition that derivatives from the inverse rendering were slower than the sampled ones. On the contrary, our methods not only smooth out the plateaus, but also improve the speed of the derivative computation.

The drawback of constructing full Hessians is seen in the TEXTURE task, where `OurH` is more than 40 times slower than the HVP due to the matrix size[14].

For the most demanding task, CNN and CNN5, and even our best method[18] is not able to reduce the error below $1:1000$. At the $1:1000$-level, the more elaborate differentials[17] is not faster than the basic one, but faster than first-order[16].

Finally, the advantage of our ability to compute HVPs is seen when comparing CNN and CNN5: while a CNN producing a single parameter is doable with Hessians[17], but if we go to higher dimensions, the aggregate strategy pays of again and is markedly faster[20].

In conclusion, albeit convergence varies with problem characteristics and dimensionality, we find that our best method, `OurHVPA` converges most reliably, beating the other algorithms by an average factor of $2.71\times$ in wall-clock units.

4.3. How much variance is reduced?

In Fig. 5 we perform a variance analysis of the different estimator on different differentiable quantities. We see that all estimators converge linearly in a log-log plot. The optimal estimator (dotted) has the lowest variance and hence would lead to the least noise in optimization, but at the expense of evaluating quadratically many elements for Hessians. Our aggregate sampling (solid line) performs slightly worse, but much, better than uniform sampling (thin line) would do.

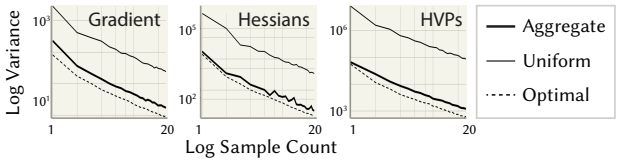


Figure 5. Estimator variance (lines) for different operators (plots).

5. Conclusion

The availability of second-order gradient information gives rise to an exciting avenue of future research in the field of inverse rendering. While these algorithms are not new (higher-order optimization has existed for decades or even centuries), their use in the computer vision and graphics community remains limited, due to a) the convenience of simple first-order gradient descent and its widespread adoption by machine learning and autodiff frameworks, and b) the increased per-iteration computational cost that is usually associated with higher-order methods. With our efficient estimators, we have shown the latter to be negligible in real-world optimization scenarios in this work.

We hope that this will inspire future research into unbiased, low-variance estimators of even-higher-order optimization methods.

References

[1] Albert S Berahas, Richard H Byrd, and Jorge Nocedal. Derivative-free optimization of noisy functions via quasi-

newton methods. *SIAM Journal on Optimization*, 29(2):965–993, 2019. 11

[2] John P Boyd. *Solving transcendental equations: the Chebyshev polynomial proxy and other numerical rootfinders, perturbation series, and oracles*. SIAM, 2014. 5

[3] Tony F Chan, Gene H Golub, and Pep Mulet. A nonlinear primal-dual method for total variation-based image restoration. *SIAM Scientific Computing*, 20(6):1964–1977, 1999. 3

[4] Swarat Chaudhuri and Armando Solar-Lezama. Smooth interpretation. *ACM Sigplan Notices*, 45(6):279–291, 2010. 3

[5] Thomas Deliot, Eric Heitz, and Laurent Belcour. Transforming a non-differentiable rasterizer into a differentiable one with stochastic gradient estimation. In *ACM i3D*, 2024. 1

[6] Deshana Desai, Etai Shuchatowitz, Zhongshi Jiang, Teseo Schneider, and Daniele Panozzo. Acorns: An easy-to-use code generator for gradients and hessians. *SoftwareX*, 17: 100901, 2022. 3

[7] Michael Fischer and Tobias Ritschel. Plateau-reduced differentiable path tracing. In *CVPR*, pages 4285–4294, 2023. 1, 3, 4, 5, 6, 10

[8] Michael Fischer and Tobias Ritschel. Zerograds: Learning local surrogates for non-differentiable graphics. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 43(4):1–15, 2024. 1, 3

[9] Roger Fletcher and Michael JD Powell. A rapidly convergent descent method for minimization. *The computer journal*, 6 (2):163–168, 1963. 2

[10] James T Kajiya. The rendering equation. In *Proc. SIGGRAPH*, 1986. 1, 3

[11] Martin Kilian, Niloy J Mitra, and Helmut Pottmann. Geometric modeling in shape space. In *SIGGRAPH*, pages 64–es, 2007. 3

[12] Dilip Krishnan and Rob Fergus. Fast image deconvolution using hyper-laplacian priors. *Proc. NeurIPS*, 22, 2009. 3

[13] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proc. ICCV*, 2019. 1

[14] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *Proc. ECCV*, 2014. 1

[15] James Martens. New insights and perspectives on the natural gradient method. *J Machine Learning Res*, 21(146):1–76, 2020. 11

[16] 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. *Comm ACM*, 65(1), 2021. 1

[17] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Trans. Graph. (Proc. SIGGRAPH Asis)*, 40(6):1–13, 2021. 3

[18] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans. Graph.*, 38(6), 2019. 1

[19] 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. 3

[20] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999. 1, 11

[21] Art Owen and Yi Zhou. Safe and effective importance sampling. *J of the American Statistical Association*, 95(449), 2000. 1, 4

[22] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994. 2, 5

[23] Michael JD Powell. Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical programming*, 14: 224–248, 1978. 11

[24] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1 (2):3, 2022. 3

[25] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 2

[26] Pedro V Sander, John Snyder, Steven J Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *Proc. SIGGRAPH*, pages 409–416, 2001. 3

[27] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24 (111):647–656, 1970. 11

[28] Jack Sherman. Adjustment of an inverse matrix corresponding to changes in the elements of a given column or row of the original matrix. *Annu. Math. Statist.*, 20:621, 1949. 11

[29] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain. 1994. 2

[30] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans Automatic Control*, 37(3):332–341, 1992. 3

[31] Eric Veach and Leonidas J Guibas. Optimally combining sampling techniques for monte carlo rendering. In *POrioc. SIGGRAPH*, pages 419–428, 1995. 4

[32] 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. 3

[33] Werbos. Backpropagation: Past and future. In *IEEE 1988 Int Conf Neural Networks*, pages 343–353. IEEE, 1988. 2, 5

[34] Manuel Werlberger, Werner Trobin, Thomas Pock, Andreas Wedel, Daniel Cremers, and Horst Bischof. Anisotropic huber-l1 optical flow. In *BMVC*, volume 1, page 3. Citeseer, 2009. 3

[35] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael Mahoney. Adahessian: An adaptive second order optimizer for machine learning. In *Proc. AAAI*, volume 35, pages 10665–10673, 2021. 3

[36] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. Inverse global illumination: Recovering reflectance models of real scenes from photographs. In *Proc. SIGGRAPH*, pages 215–224, 1999. 1

[37] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime TV-L1 optical flow. In *DAGM Pattern Recognition*, pages 214–223, 2007. 3