Spatiotemporal Bilateral Gradient Filtering for Inverse Rendering

WESLEY CHANG*, University of California San Diego, USA
XUANDA YANG*, University of California San Diego, USA
YASH BELHE*, University of California San Diego, USA
RAVI RAMAMOORTHI, University of California San Diego, USA
TZU-MAO LI, University of California San Diego, USA

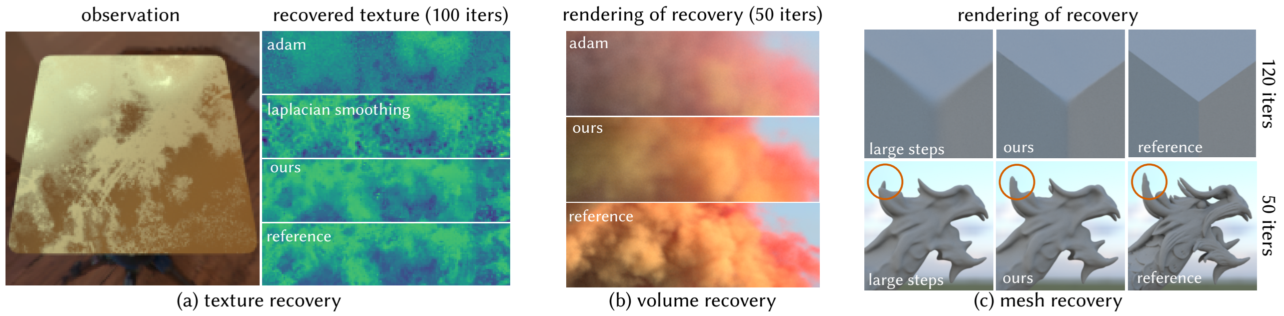


Fig. 1. We introduce a *spatiotemporal* optimizer that generalizes Adam and Laplacian Smoothing (Large Steps). It applies an *anisotropic cross-bilateral filter* to the gradient across space, in addition to temporal filtering (like Adam). Our cross-bilateral filter reduces gradient noise and improves conditioning for anisotropic objectives by imposing a piecewise smoothness prior. Our method enables faster convergence and higher quality inverse rendering of (a) textures, (b) volumes and (c) meshes at very low sample counts; all experiments only use 1 sample per pixel for gradient estimation. (a) For roughness texture recovery after 100 iterations, our method has converged while others have artifacts. (b) For volume density and albedo recovery in just 50 iterations, our method can already recover the rough shape and color; further optimization with higher sample counts recovers details. (c) For mesh recovery our method is able to recover sharp features (top row, cube) and thin structures (bottom row, dragon) faster than competing methods. Scenes adapted from At the Window ©Bernhard Vogl, Autumn Field ©Jarod Guest and Sergej Majboroda, High-Res Smoke Plume ©JangaFX, Kloppenheim 06 ©Greg Zaal, and Asian Dragon ©Stanford Computer Graphics Laboratory.

In inverse rendering, gradient-based methods, which have seen great progress in the recent years, are typically used in conjunction with the Adam optimizer. While Adam usually improves convergence by temporally filtering gradients over previous iterations to reduce noise, it is not tailored to inverse rendering where the target signals (textures, volumes, or geometry) are usually piecewise smooth. Previous work has applied the inverse Laplacian operator to smooth gradients spatially, but this isotropic filtering can often lead to oversmoothing. We propose a *spatiotemporal* optimizer that can significantly speedup the convergence over Adam, by enforcing the optimization parameter updates to be piecewise smooth through a lightweight spatial domain cross-bilateral filter. We discuss different options of combining spatial filtering and Adam's temporal filtering, and provide intuitions for different scenarios. We show that our filtering leads to significantly higher-quality reconstructions in different inverse problems including texture, volume and geometry recovery.

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

*The three authors contributed equally to this research.

Authors' Contact Information: Wesley Chang, wec022@ucsd.edu, University of California San Diego, USA; Xuanda Yang, xuy008@ucsd.edu, University of California San Diego, USA; Yash Belhe, ybelhe@ucsd.edu, University of California San Diego, USA; Ravi Ramamoorthi, ravir@ucsd.edu, University of California San Diego, USA; Tzu-Mao Li, tzli@ucsd.edu, University of California San Diego, USA.

Additional Key Words and Phrases: differentiable rendering, inverse rendering, optimization, preconditioning, bilateral filtering,

ACM Reference Format:
Wesley Chang, Xuanda Yang, Yash Belhe, Ravi Ramamoorthi, and Tzu-Mao Li. 2024. Spatiotemporal Bilateral Gradient Filtering for Inverse Rendering. In *SIGGRAPH Asia 2024 Conference Papers (SA Conference Papers '24), December 3–6, 2024, Tokyo, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3680528.3687606

1 INTRODUCTION

Gradient-based optimization has enabled many inverse rendering applications such as texture, volume, and geometry recovery from observed images. With a few exceptions, most inverse rendering works use the Adam optimizer [Kingma and Ba 2015] to process gradients and update optimization parameters. In this work, we show that we can significantly speedup Adam and other prior work [Nicolet et al. 2021; Osher et al. 2018] over a wide range of inverse rendering tasks, by applying edge-aware spatial filtering, such as a lightweight cross bilateral filter [Eisemann and Durand 2004; Petschnigg et al. 2004] at each iteration, as seen in Figure 1.

Adam can be seen as a *temporal filter* that rescales the gradient component-wise based on its value from previous iterations. This has two major benefits: first, when the gradient is noisy, due to either the stochastic evaluation of the objective function and its gradient or minibatching, temporal filtering reduces noise. Second, it adjusts the learning rate per component to use faster learning rates on gradient components that change slower over time, and vice versa.

However, the per-component adjustment alone is insufficient either when the gradient is highly noisy, or when the loss landscape is highly *anisotropic* — causing the scales of the gradient components to differ greatly. As a result, it often recovers signals with spurious high frequencies.

Our work builds on the idea that *spatially filtering* gradients can significantly speedup optimization [Nicolet et al. 2021; Osher et al. 2018; Renka 2006]. The idea is that if the final target is spatially smooth, enforcing parameter updates to be spatially smooth as well improves convergence. However, existing work always applies *isotropic* spatial filters, posed as an inverse Laplacian operator, which can oversmooth the gradient, substantially hurting convergence when the target has strong edges.

To preserve edges in the target, we instead apply a cross-bilateral filter, which applies a strong filter to the gradients at the smooth regions of the recovered signal, and applies a weak filter to the gradients around the sharp features. Our recovered signal is then *piecewise smooth* [Field 1994], as opposed to the noisy results obtained from Adam or the oversmoothed ones using isotropic filtering. Combining cross bilateral filtering with Adam leads to a few design decisions that lead to significantly different results: spatial filtering can be done before or after temporal filtering, and different parameters can be used to filter different states in Adam. We analyze and discuss different scenarios, and provide intuitions on how spatial filtering and Adam should be combined.

Our optimizer has the following key features:

- It generalizes Adam and Laplacian Smoothing to anisotropic spatiotemporal filtering.
- With high-noise gradients, it recovers higher quality solutions in fewer iterations.
- With low-noise gradients, its non-diagonal preconditioning enables faster convergence.
- The intermediate recoveries are piecewise smooth, making it better suited to early stopping.
- It is widely applicable to inverse recovery including textures, volumes, and meshes.

2 RELATED WORK

Gradient-based optimization. Choosing the right step size for the classical (stochastic) gradient descent method [Robbins and Monro 1951] is difficult. Second-order methods like Newton's method provide the step size adjustment, but computation of the Hessian matrix is expensive, and when the gradient is noisy, the Hessian can also be noisy and nullify the benefit of Newton's method. Thus, adaptive methods [Duchi et al. 2011; Tieleman and Hinton 2012] have been proposed to efficiently adjust per-component step sizes. Adam [Kingma and Ba 2015] combines ideas from previous adaptive methods, it uses exponential moving averages to update both the first and second moment of the gradient, using the latter to adjust step sizes. Many variants of Adam have been proposed for deep learning tasks, further discussed by Liu et al. [2024] and Chen et al. [2024]. We focus on Adam, but our idea is conceptually compatible with other variants.

Our work is highly-related to the idea that spatially filtering the gradient can also serve as a step size adjustment scheme, enforcing the parameters to evolve at similar rates spatially (sometimes known as "Sobolev gradient descent") [Jung et al. 2009; Osher et al. 2018; Renka 2006; Renka and Neuberger 1995] . The same idea also has been applied to inverse rendering recently [Nicolet et al. 2021]. These works use the inverse Laplacian operator as the spatial filter, which can lead to slow convergence at sharp regions of the recovered signal. We show that a cross bilateral filter can converge significantly faster, and discuss the combination of spatial filtering and Adam.

Some work proposed to learn a preconditioner to process gradients for speeding up optimization (e.g., [Andrychowicz et al. 2016; Li et al. 2020, 2023]). Our spatialtemporal filter does not require any training and can be applied to a wide range of tasks. Nevertheless, it could be interesting to combine both methods.

Variance reduction in differentiable rendering. Gradient-based optimization for inverse rendering (e.g., [Azinović et al. 2019; Blanz and Vetter 1999; Gkioulekas et al. 2013; Li et al. 2018]) often relies on stochastic computation of the gradient, since physically-based rendering often relies on Monte Carlo integration [Pharr et al. 2016]. However, high variance in gradients can impede the progress of optimization. Thus, a large body of work has been proposed to reduce their variance, usually by designing specialized estimators for them [Bangaru et al. 2020; Belhe et al. 2024; Chang et al. 2023; Nimier-David et al. 2022; Yu et al. 2022; Zeltner et al. 2021; Zhang et al. 2021a, 2020, 2021b]. In optimization, stochastic variance reduced gradient [Johnson and Zhang 2013] and related ideas [Roux et al. 2012; Shalev-Shwartz and Zhang 2013] that apply control variates to reduce variance of mini-batches have shown asymptotically faster convergence rates for some loss functions.

Our work is complementary to these approaches and can take gradients produced by these methods as input. We provide further variance reduction by spatially filtering the gradients.

Edge-aware filtering. Our work builds on the recent success on the design of image filters that preserve edges while smoothing out noise (e.g., [Barron and Poole 2016; Buades et al. 2005; Paris and Durand 2009; Tomasi and Manduchi 1998]). Originally an expensive filtering operation, after decades of research, edge-aware filtering can now be done in real-time [Chen et al. 2007; Gastal and Oliveira 2012; He et al. 2013; Liu et al. 2020] and is often used in real-time rendering for denoising [Bauszat et al. 2011; Dammertz et al. 2010; Koskela et al. 2019; Schied et al. 2017]. It has also been shown to be helpful for mesh smoothing [Jones et al. 2003; Solomon et al. 2014].

We apply a cross-bilateral filter [Eisemann and Durand 2004; Petschnigg et al. 2004] to the gradient using the current recovered signal as a guide. To speedup the computation for images and volumes, we apply the edge-avoiding À-trous filter [Dammertz et al. 2010]. For meshes, we use Solomon et al. [2014]'s variant.

Piecewise smoothness in natural signals. A key assumption of our work is that we are recovering a *piecewise-smooth* signal. This observation stems from earlier studies on natural image priors [Field 1994; Torralba and Oliva 2003], which suggest that the distribution of the spatial gradients of natural images tends to be *heavy-tailed*. These distributions typically lead to piecewise smooth images with a few strong gradients along the edges, and many small gradients across the smooth regions. Prior work incorporates this prior for image reconstruction [Krishnan and Fergus 2009; Rudin et al. 1992; Simoncelli and Adelson 1996]. In particular, image denoising is a good universal prior for image reconstruction [Romano et al. 2017]. We instead enforce piecewise smoothness on the *gradient*, using an edge-aware filter with the recovered signal serving as a guide for the edges. When combined with early stopping [Yao et al. 2007], that stops the optimization before convergence, our optimizer can be used as a prior to prefer piecewise-smooth solutions.

3 BACKGROUND AND MOTIVATION

3.1 Gradient-based Optimization

Given an objective function $f : \mathbb{R}^n \to \mathbb{R}$, the goal of optimization is to find the parameters θ that minimize (or maximize) $f(\theta)$. If f is differentiable, we can minimize it using gradient-based methods that utilize the gradient $\nabla_\theta f$. The simplest form of gradient descent minimizes f by updating the parameters iteratively in the direction of the negative gradient:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta f_t, \tag{1}$$

where α is the learning rate, and t is the iteration number.

This form of gradient descent is however, not always feasible, nor the most efficient. First, in many optimization problems, including many inverse problems in graphics, the true gradient $\nabla_\theta f$ is often expensive to compute. Instead it is replaced by a noisy, stochastic estimate of the gradient. This form of stochastic gradient descent can still converge [Robbins and Monro 1951], but noisy gradients often greatly reduce convergence speed. Second, it is well known that the negative gradient, which points in the direction of the steepest descent, is not always the best descent direction. For example, if f is quadratic, then the optimal direction is instead the negative gradient multiplied by the inverse Hessian: $-H^{-1}\nabla_\theta f$, which if used, would reach the global minimum in one iteration. In general, if f is anisotropic, that is the scales of the parameters differ in different dimensions, it can help to *precondition* by a matrix P^{-1}:

$$\theta_{t+1} = \theta_t - \alpha P^{-1} \nabla_\theta f_t, \tag{2}$$

which reduces the condition number and anisotropy of f, accelerating convergence [Faragó and Karátson 2002]. Additionally, as long as P^{-1} is positive definite, $-P^{-1}\nabla_\theta f_t$ remains a *descent direction.*[1]

3.2 Adam

The Adam optimizer [Kingma and Ba 2015] attempts to address the issues of gradient noise and anisotropy by filtering gradients temporally across iterations. See Algorithm 1 for the Adam update rule. Note we omitted the bias correction terms for brevity.

Algorithm 1. Adam Update

```
$g_t \leftarrow \nabla_{\theta_t} f$
$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
$\theta_{t+1} \leftarrow \theta_t - \alpha \frac{m_t}{\sqrt{v_t} + \varepsilon}$
```

[1]x is a descent direction if $-\nabla_\theta f^T x > 0$. Since P^{-1} is positive definite, $(-\nabla_\theta f^T)(-P^{-1}\nabla_\theta f) > 0$, therefore, $-P^{-1}\nabla_\theta f$ is a descent direction.

Adam keeps an exponential moving average of the gradient, m_t, which reduces the noise of the gradient while acting as a *momentum* that accelerates optimization [Goh 2017; Pedregosa 2023]. It also does the same for the square of the gradient, v_t, and $m_t/\sqrt{v_t}$ multiplied by the learning rate α becomes the update step. The exponential moving average m_t (or v_t) can be seen as a form of temporal filtering of g_t (or g_t^2), where β_1 (or β_2) controls the filtering strength. Crucially, the ratio $m_t/\sqrt{v_t}$ informally measures the signal-to-noise ratio in the gradient, which helps the optimization by reducing the step size of the parameter update when its gradient is very noisy or the function is very anisotropic.

However, this per-parameter filtering and division by $\sqrt{v_t}$ is a form of *diagonal* preconditioning, which assumes *no correlations* across the parameters θ. This means that each component of θ evolves independently over optimization, resulting in high frequency artifacts if: a) gradients are very noisy, b) f is highly anisotropic or c) optimization is terminated early. Figure 2 and Figure 3 show numerical examples that demonstrate these phenomena. We aim to address these remaining issues (slow convergence and high frequency artifacts) for problems where we can efficiently compute more than just a diagonal preconditioner by exploiting parameter correlations.

4 METHOD

In many problems in graphics, there is a natural notion of spatial coherence: the pixels in an image, texels in a texture, voxels in a volume, and vertices in a mesh, are often all correlated to their spatial neighbors. We exploit this spatial coherence to design an optimizer that generalizes Adam's temporal filtering to anisotropic *spatiotemporal* filtering across parameters (Section 4.1). At each iteration, our optimizer spatially filters gradients to be piecewise smooth, and therefore biases intermediate solutions toward piecewise smoothness. We also show that previous work employing Laplacian smoothing as a gradient preconditioner is a special case of our framework, obtained by restricting the spatial filter to an isotropic filter (Section 4.3).

4.1 Anisotropic spatiotemporal filtering

Algorithm 2. Our Spatiotemporal Update

```
$g_t \leftarrow \nabla_{\theta_t} f$
$\tilde{g}_t \leftarrow \mathrm{filter}(g_t)$                    # Prefilter
$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) \tilde{g}_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) \tilde{g}_t^2$
$\tilde{m}_t \leftarrow \mathrm{filter}(m_t)$                  # Postfilter
$\tilde{v}_t \leftarrow \mathrm{filter}(v_t)$                   # Postfilter
$\theta_{t+1} \leftarrow \theta_t - \alpha \frac{\tilde{m}_t}{\sqrt{\tilde{v}_t} + \varepsilon}$
```

Algorithm 2 describes our method[2]. It extends Adam's temporal gradient filtering by applying *up to* three spatial filters: a "prefilter" on g_t before the exponential moving averages, and two "postfilters" (on m_t and v_t respectively) after the exponential moving averages, resulting in a *spatiotemporal* filter. In our grid-based applications

[2]To correct for bias, after postfiltering set $\tilde{m} \leftarrow \tilde{m}/(1-\beta_1^t)$ and $\tilde{v} \leftarrow \tilde{v}/(1-\beta_2^t)$.

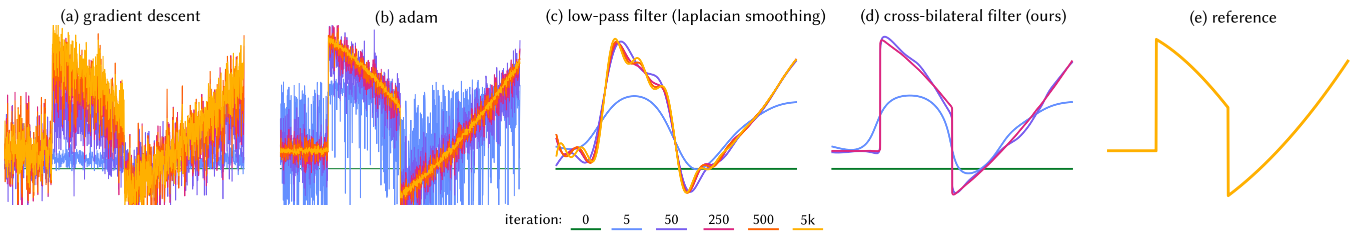


Fig. 2. Anisotropic objective with zero noise in gradients. Here, we optimize an anisotropic objective function $f(\theta) = (\theta - \theta^*)^T \mathbf{A}^T \mathbf{A}(\theta - \theta^*)$. Each element of $\mathbf{A}$ is drawn from a standard normal distribution. The reference $\theta^* \in \mathbb{R}^{1000}$ (e) is piecewise smooth. The gradients $\nabla_\theta f$ have zero noise. We use four different optimizers (with individually tuned hyperparameters). (a) Gradient descent cannot find a global learning rate that works well for all parameters. (b) Adam estimates learning rates for each dimension independently, resulting in high frequency fluctuations. (c) Applying Laplacian Smoothing over space (θ) after Adam's temporal smoothing results in a much smoother parameter trajectory over time; however, it oversmooths edges. (d) Our cross-bilateral filter preconditions the high frequency gradients (resulting from anisotropy) to much lower frequency ones, while also preserving edges due to the edge-aware filter. It converges near edges much faster (250 iterations) as compared to the others which are unable to do so even after 5000 iterations.

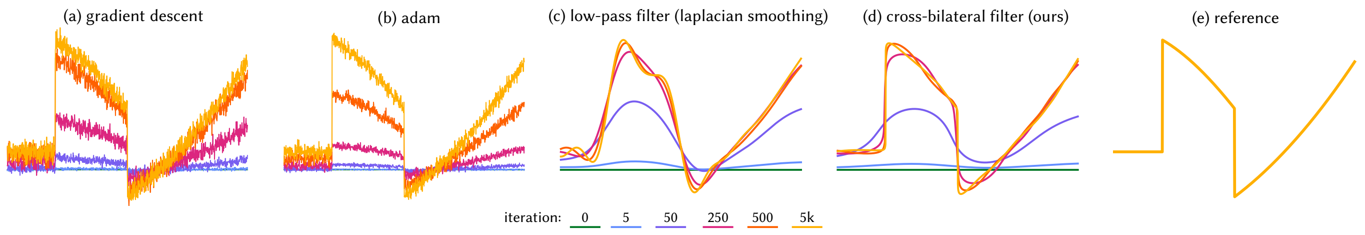


Fig. 3. Isotropic objective with noisy gradients. Here, we examine the effect of noisy gradients for optimization of isotropic objective $f(\theta) = \|\theta - \theta^*\|^2$ with piecewise smooth θ^* (see reference (e)). We simulate noisy gradients by adding zero mean Gaussian noise. Here too, we optimally tune all hyperparameters for the four optimizers. (a) Despite the simple isotropic loss, gradient descent isn't able to converge to the right solution due to noisy gradients. (b) This problem persists with Adam too, however, the temporal filtering reduces the noise compared to (a). (c) Laplacian Smoothing over-smoothes the edges and is unable to recover them well. (d) Our cross-bilateral filter denoises the gradients well, while preserving edges enabling piecewise-smooth recovery.

detailed in Section 5.1, we find applying only the postfilters to perform the best (Figure 8), while the exact set of filters used in our mesh application is described in detail in Section 5.2. We analyze the differences between pre- and post-filtering in detail in the applications and supplementary materials. Similar to the temporal filter, the spatial filters also reduce noise in the gradient by smoothing the gradient using information from spatial neighbors.

Although our framework is not restricted to particular types of filters, we analyze the use of the cross-bilateral filters [Eisemann and Durand 2004; Petschnigg et al. 2004] in this work. In particular, our filter function is:

$$\tilde{h}(x) = \frac{1}{z(x)} \int_{\Omega} h(y) w_s(x, y) w_d(\theta(y), \theta(x))\, \mathrm{d}y, \tag{3}$$

where h is the function to be filtered, Ω is the spatial domain, w_s is a spatial isotropic low-pass filter, w_d is the data term that prevents smoothing across discontinuities, and $z(x)$ is the normalization term[3]. For our grid-based applications, w_s and w_d are:

$$w_s(x, y) = \exp\left(-\frac{||x - y||}{\sigma_s}\right),\ w_d(\theta_x, \theta_y) = \exp\left(-\frac{||\theta_x - \theta_y||}{\sigma_d}\right), \tag{4}$$

where σ_s and σ_d control the scale of the weights. For our mesh application, we instead use the generalized bilateral filter method from Solomon et al. [2014], which we discuss in Section 5.2.

Our cross-bilateral filter filters *gradients* $\nabla_\theta f$ spatially, using their *parameter values* θ to prevent over smoothing across edges in the parameters. Conversely, a standard bilateral filter would filter gradients using the *gradient values* for edge stopping. Using a cross-bilateral filter instead of a standard bilateral filter is key, since not only are parameter values less noisy than their gradients, but are also often in a lower dynamic range than them. Figure 4 (c) and (d) show a comparison between these two for a texture recovery example. For all filtering operations, we use the same parameters θ for the data term w_d. This is critical for postfiltering, since it ensures our method makes bounded steps, like Adam (proof in Suppl. Sec. 1.3). Violating this significantly degrades recovery, see Section 5.1. Since our method directly filters gradients and moments, it requires the same storage as Adam.

Our filter when applied as a preconditioner to the gradient can be written as matrix-vector product $(D^{-1}K)\nabla_\theta f$ (e.g., [Milanfar 2012]). Here, K is a symmetric positive definite affinity matrix accounting for w_s and w_d, and D is a diagonal matrix accounting for z. Interestingly, even though the matrix $D^{-1}K$ is not symmetric, all its eigenvalues are real, non-negative, and bounded by 1 [Milanfar 2012], justifying its use as a preconditioner. More importantly, since K includes non-diagonal elements which capture correlations across the parameters, our method performs *non-diagonal* preconditioning.

4.2 Advantages of our method

4.2.1 Noise removal. In the image processing community, cross-bilateral filtering is known for its effectiveness at removing noise, while preserving sharp boundaries. For optimization, compared to Adam, cross-bilateral filtering is much more efficient at noise removal in the gradient, and compared to isotropic low-pass filters,

[3] $z(x) = \int_{\Omega} w_s(x, y) w_d(\theta(y), \theta(x))\, \mathrm{d}y$.

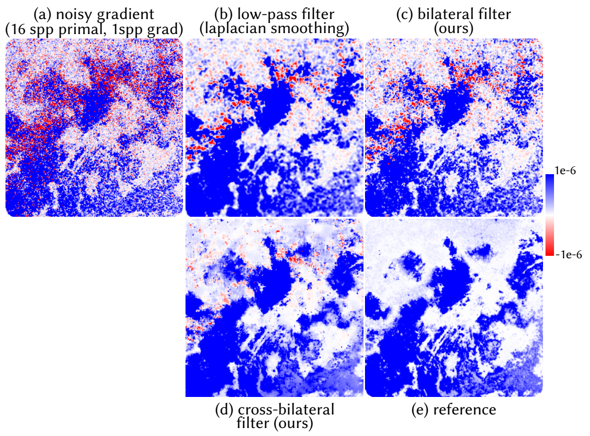


Fig. 4. Denoising with spatial filters. We denoise noisy gradients (a) (from Fig. 5) using spatial filtering only. The reference (e) is purely positive (blue), so all negative regions (red) have the wrong sign, which hurts optimization. (b) The low-pass filtering reduces noise but blurs over fine structures in the gradient at the bottom. (c) Our bilateral filter preserves these fine structures but still has some noise. (d) Our cross-bilateral filter not only preserves the fine structures, it does so while significantly reducing noise in the gradient.

it does not pollute gradients between parameters that are very different (Figure 3). Practically, noise removal improves the signal-to-noise ratio resulting in faster convergence, see Section 5.

4.2.2 Non-diagonal preconditioning. We have found our method's non-diagonal preconditioning be helpful for anisotropic objectives with parameter correlations. To isolate the effect of noise removal from the effect of better preconditioning, we construct a numerical example with noise-free gradients in Figure 2. Despite the gradients being noise-free, Adam struggles to handle the strong anisotropy, whereas our method with its non-diagonal preconditioning converges in a much smoother manner.

4.2.3 Piecewise smoothness prior. Our cross-bilateral filter preconditions gradients to be piecewise smooth. This results in an optimization trajectory with piecewise smooth intermediate solutions (Figures 2 and 3). Thus, applying early stopping [Yao et al. 2007] with our method is preferable, since the piecewise smooth solutions it prefers align well with natural signals (Section 2).

4.3 Relation to Laplacian Smoothing

Previous work has applied Laplacian smoothing to the gradients [Osher et al. 2018; Renka 2006]. Nicolet et al. [2021] further combined this with Adam to smooth out the sparse gradients in inverse rendering of meshes. These methods apply a preconditioner of the form $(I+\lambda L)^{-1}$, where L is the Laplacian matrix and the scalar λ controls smoothing strength. This has the effect of applying an isotropic spatial filter. Precisely, it is well-known (e.g., [Bhat et al. 2008]) that the *screened Poisson equation* has a closed-form solution in 2D and 3D Euclidean domains. In 2D, the solution is a convolution with a kernel $\frac{1}{2\pi\lambda}K_0(2\pi\sqrt{\frac{1}{\lambda}\left(x^2+y^2\right)})$ where K_0 is the zeroth order modified Bessel function of the second kind, which has the approximate shape of e^{-r}/r where $r=\sqrt{x^2+y^2}$. Our spatial smoothing term w_s is of the form e^{-r}, highly-related to Laplacian Smoothing.

Similar to our method, Laplacian Smoothing reduces noise and sparsity in the gradients, as well as anisotropy in f. Since it promotes smooth gradients, Laplacian Smoothing rapidly converges when the target is smooth. However, for piecewise smooth targets, it is slow to converge near edges, due to oversmoothed gradients. Our spatial filter is edge-aware due to the data term w_d, which prevents over-smoothing, see Figures 2 and 3.

For meshes, several works have additionally applied different discrete Laplacian operators. Dou et al. [2024] use the cotangent Laplacian for micro-mesh construction, but this has shown limited benefits in inverse rendering of standard meshes [Nicolet et al. 2021]. An et al. [2023] propose to adaptively set the smoothing kernel size σ_s to help preserve high frequency features. However, this is still fundamentally an *isotropic* filter and is thus orthogonal to our work.

5 APPLICATIONS

We apply our method to inverse recovery of textures, volumes and meshes. For textures and volumes (Section 5.1), we recover parameters on a regular grid (2D texture and 3D voxel grid). For mesh recovery, we optimize vertex positions, which necessitates the use of a different bilateral filter [Solomon et al. 2014]. We discuss it and our connection with previous work for mesh optimization [Nicolet et al. 2021] in Section 5.2. We also analyze the hyperparameters introduced by our method in Section 3 of the supplemental and bias in Section 4 of the supplemental.

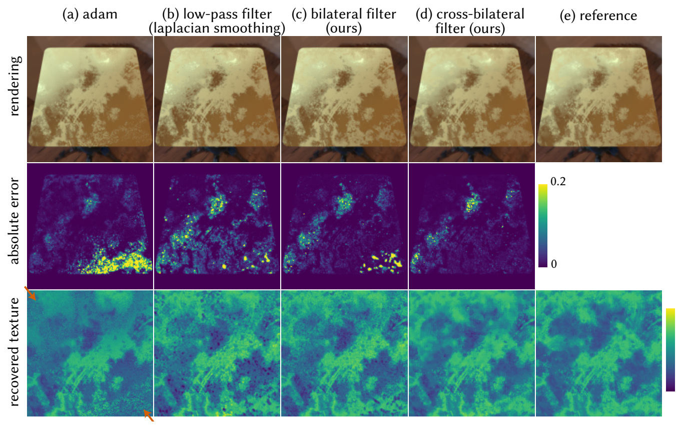
5.1 Grid-based applications

Our optimizer enables higher quality inverse recovery with low sample budgets, all while requiring fewer iterations for convergence. To ensure optimization is fast, spatial filtering must be fast too. We leverage previous work in real-time denoising [Dammertz et al. 2010] and implement our cross-bilateral filter as a series of à-trous filters. We discuss further details including the overhead introduced by our method in the following paragraphs. After that, we present recovery results for inverse recovery of textures and volumes.

À-trous cross-bilateral filtering of gradients. Depending upon the application, the gradient $\nabla_\theta f_t$ and its associated buffers m_t and v_t, are defined on regular 2D or 3D grids. We implement spatial filtering by applying a sequence of À-trous cross-bilateral filters, each with an increasing support but a fixed number of non-zero elements, similar to Dammertz et al. [2010]. In particular, at each step we use a dilated 3x3 filter for 2D filtering (or a 3x3x3 filter in 3D). The number of filters F in the sequence indirectly specifies σ_s.

Performance. We implement our optimizer in PyTorch [Paszke et al. 2019] with a custom cross-bilateral filtering CUDA kernel written in Slang [Bangaru et al. 2023; He et al. 2018]. All of our inverse rendering experiments were conducted using Mitsuba 3 [Jakob et al. 2022] on an NVIDIA GeForce RTX 3080 GPU. Our method introduces little overhead (< 4%) for both 2D and 3D filtering (Table 1 in supplemental), so all comparisons are equal-iteration.

Baselines. We compare our optimizer (with postfiltering only) to the two techniques we generalize: Adam and Laplacian Smoothing. We also demonstrate the benefit of using a cross-bilateral filter over a standard bilateral filtering with an ablation. We tune the following


(a) adam
(b) low-pass filter (laplacian smoothing)
(c) bilateral filter (ours)
(d) cross-bilateral filter (ours)
(e) reference
rendering
absolute error
recovered texture
0.2
0
1
0

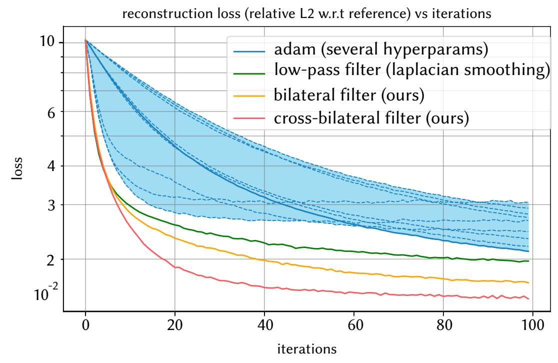


Fig. 5. Inverse texture recovery. We attempt to recover the roughness texture of a metallic plate from a single reference image (e) using noisy gradients (16 samples per pixel for primal and 1 sample per pixel for gradient). To find the optimal hyperparameters for Adam, we performed a grid search over the learning rate: 0.001, 0.01, 0.05 and β_1 : 0.2, 0.5, 0.9, and show recovery for the best result ($\alpha = 0.01$ and $\beta_1 = 0.2$). (a) Adam is unable to converge in several regions (see arrows). (b) Laplacian Smoothing fares better in these regions, but still has high error in others (see error image). (c) Our bilateral filter improves upon Laplacian Smoothing (compare error images). (d) Our cross bilateral filter is able to match the reference the closest, both in terms of loss and the quality of recovered texture. On the right, we see that the loss curve follows the qualitative trends we discussed above; each different component added to our method helps improve recovery. Scene adapted from At the Window ©Bernhard Vogl.

hyperparameters for all methods per experiment: learning rate α, temporal filtering coefficient β_1, data term scale σ_d and spatial filter support F.

We set β_2 satisfying $1 - \beta_1 = \sqrt{1 - \beta_2}$. Lower values of β_1 and β_2 reduce temporal filtering in m_t and v_t. While this decreases bias accumulated in m_t and v_t from previous iterations, it increases noise in them. Our experiments use low iteration counts (<200) for fast recovery. For this setup we found all optimizers converge fastest with low values of $\beta_1 \in (0.2, 0.5)$. This is because higher values increase temporal bias, slowing down convergence within the limited iteration budget, resulting in worse recovery.

5.1.1 Inverse texture recovery. We apply our method to recover the roughness texture of a plate from a single view in Figure 5. We use 16 samples per pixel for forward rendering and 1 sample per pixel for the gradient. All methods except Adam use $\beta_1 = 0.2$ and $\alpha = 0.1$ and; for Adam we found $\alpha = 0.01$ worked best. The texture is a 512^2 2D grid. We set $F = 5$ for cross-bilateral filtering, while $F = 3$ worked best for Laplacian Smoothing and standard bilateral filtering. For the cross-bilateral filter, we use $\sigma_d = 0.1$ and for the standard bilateral filter we use $\sigma_d = 1$; for both, we filter in the log domain (for the standard bilateral filter, since gradients can be negative, we take the absolute value before the logarithm).

Our cross-bilateral filter can recover the roughness texture, including all its fine details, better than competing methods. Laplacian Smoothing oversmooths and standard bilateral filtering is noisier than our method. Adam has the strongest artifacts due to its inability to filter the noisy gradients effectively.

5.1.2 Inverse volume recovery. Next, we apply our method to recover the volume density σ_t and albedo ρ from 64 rendered views in Figures 6 and 7; at each iteration, we compute the loss for all views. For both experiments, the grid size for ρ and σ_t is 64^3. We found $\beta_1 = 0.2$ to work best for all methods. We use $\alpha = 0.008$ for Adam and $\alpha = 0.02$ for the others. For filtering, we use $F = 3$ for all methods. For the data term we use $\sigma_d = 0.001$ for ρ and $\sigma_d = 0.2$ for σ_t with our cross-bilateral filter; for the standard bilateral filter, we use $\sigma_d = 0.001$ for ρ and $\sigma_d = 2 \cdot 10^{-7}$ for σ_t.

We demonstrate two key benefits of our method in the two examples. First, in Figure 6, we show it can handle noisy gradients much better than Adam (similar to our numerical example in Figure 3). Second, in Figure 7 we show that even with much cleaner gradients, our method's preconditioning allows it to converge much faster than Adam (similar to our numerical example in Figure 2).

Prefiltering and postfiltering ablation. In texture and volume recovery, our method uses postfiltering only. As shown in Figure 8, with optimally tuned hyperparameters, postfiltering and prefiltering perform similarly (as does applying both the postfilter and the prefilter), but postfiltering converges slightly faster. In Section 1.2 of the supplemental, we show postfiltering is more adaptive to noise in the low β_1 regime, which might explain its benefits.

Postfiltering m_t and v_t. For spatial filtering, it is crucial to filter *both* m_t and v_t to ensure bounded steps at each iteration, similar to Adam; violating this results in instability during training, see Figure 9. We show that filtering them both with the same weights ensures bounded steps in Section 1.3 of the Supplemental.

Purely spatial filtering. Setting $\beta_1 = \beta_2 = 0$ turns off temporal filtering completely. This setting is particularly interesting, because it makes the optimizers *memoryless*, eliminating the need to maintain buffers for m_t and v_t. The resulting 2/3rd memory savings can be useful for large-scale inverse optimization. Due to spatial filtering, postfiltering can still dynamically adjust learning rates, unlike Adam which makes constant sized steps, resulting in faster convergence (see Section 1.1 and Figure 1 in Supplemental.).

5.2 Inverse rendering of meshes

In this section, we discuss the exact connection between our optimizer and Large Steps [Nicolet et al. 2021]. We show that their

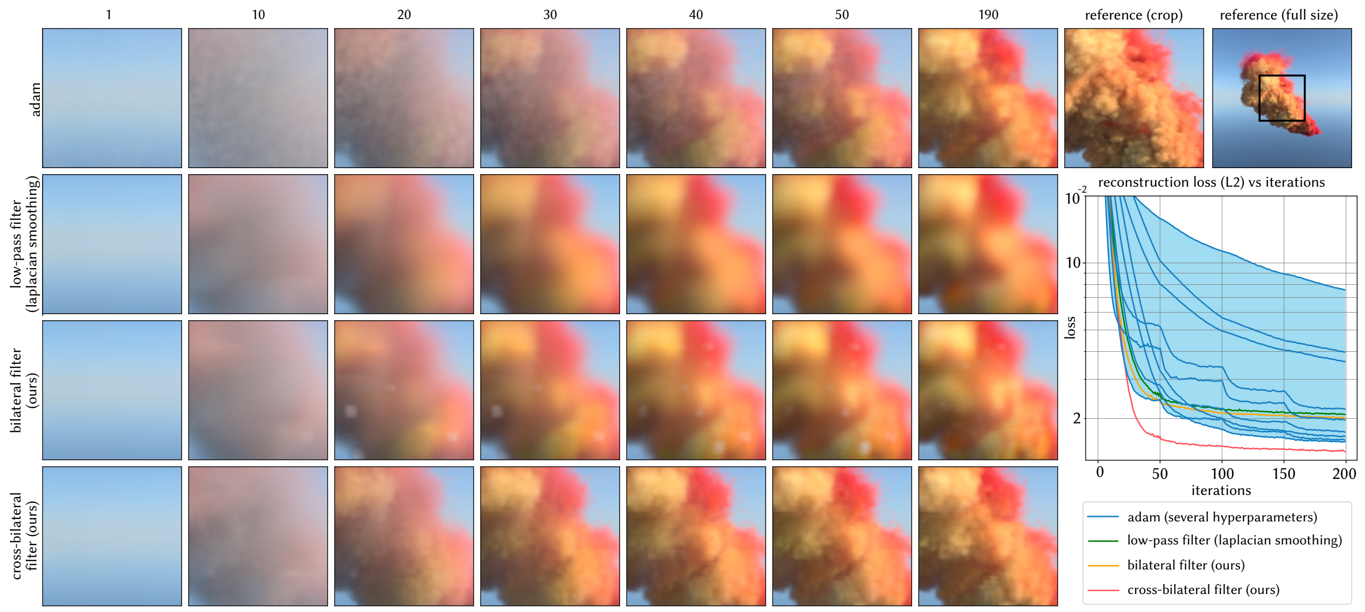

1
10
20
30
40
50
190
reference (crop)
reference (full size)
adam
low-pass filter (laplacian smoothing)
bilateral filter (ours)
cross-bilateral filter (ours)
reconstruction loss (L2) vs iterations
loss
iterations
adam (several hyperparameters)
low-pass filter (laplacian smoothing)
bilateral filter (ours)
cross-bilateral filter (ours)

Fig. 6. Low sample count inverse volume recovery. The objective is to estimate the density σ_t and albedo ρ (both volumetric) from 64 views. The gradients are very noisy, resulting from multiple scattering and low sample counts (16/1 samples per pixel for primal/gradient). We compare our cross-bilateral filter with Adam, Laplacian Smoothing and an ablation that uses a standard bilateral filter; we plot the reconstruction loss (L2) on the right. We use optimal hyperparameters for all methods; for Adam, we show the loss curves for all the hyperparameters we searched over, and show results for the best one. Adam shows high-frequency noise in its recovery. Both Laplacian Smoothing and standard bilateral filtering oversmooths high-frequency details. Our cross-bilateral filter is able to denoise gradients and recover a piecewise smooth solution much faster than other methods (~50 iterations); it does not oversmooth nor does it have high frequency artifacts. Scene adapted from Autumn Field ©Jarod Guest and Sergej Majboroda, and High-Res Smoke Plume ©JangaFX.

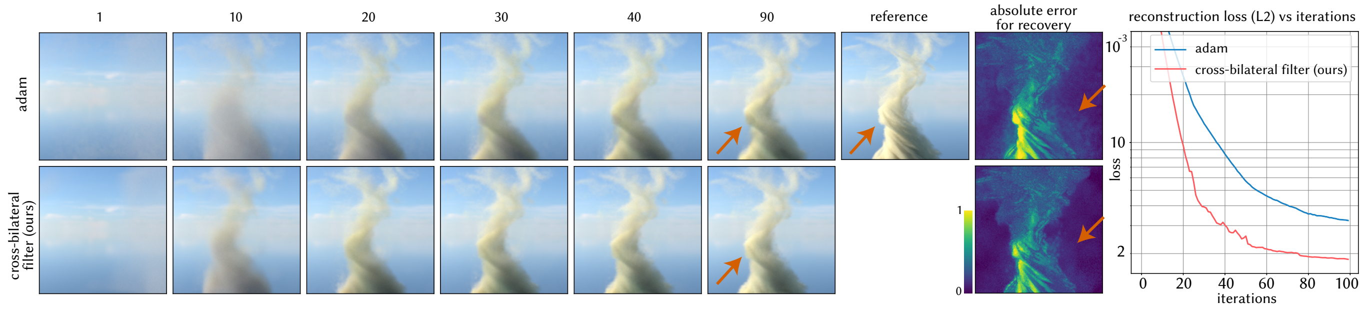

1
10
20
30
40
90
reference
absolute error for recovery
reconstruction loss (L2) vs iterations
adam
cross-bilateral filter (ours)
loss
iterations

Fig. 7. High sample count inverse volume recovery. Similar to Fig. 7, we want to recover σ_t and ρ for the observed volume. This time, we use much higher sample counts (128/32 samples per pixel for primal/gradient) to reduce the effect of gradient noise and highlight the benefits of our piecewise-smooth gradient preconditioning. Despite the gradients being much less noisy, Adam still has slower convergence than our method. Its final recovery has higher loss (see arrow at 90 iterations) and more high frequency artifacts due to non-uniform convergence than our method (see arrow in error image). Our method's preconditioning produces piecewise smooth intermediate solutions, which improves convergence even with less noisy gradients as we saw in Figure 2. Scene adapted from Autumn Field ©Jarod Guest and Sergej Majboroda and Dust Devil ©JangaFX.

method can be cast into ours by selecting a specific set of filters. We provide an ablation on the effect of different filtering decisions and show that our method can converge faster than Large Steps for both piecewise smooth objects and objects with thin geometry.

5.2.1 Large Steps as spatiotemporal gradient filtering. As discussed by Nicolet et al. [2021], in inverse rendering of geometry, gradients are heavily concentrated on the sparse silhouettes of objects. Naïvely applying gradient descent causes local vertex perturbations that quickly lead to flipped triangles and tangled up meshes. As a result, Nicolet et al. [2021] propose to precondition the gradient by a matrix of the form $(I + \lambda L)^{-2}$, which is the preconditioner discussed in Section 4.3 applied *twice*. However, instead of directly applying this preconditioner on the gradients with respect to vertices $\mathbf{x}$ and then performing an Adam update, they do the following:

(1) Apply a reparameterization, by letting $\mathbf{u} = (I + \lambda L)\mathbf{x}$.
(2) Compute a gradient for $\mathbf{u}$ as $\frac{\partial \mathbf{x}}{\partial \mathbf{u}} \cdot \nabla_{\mathbf{x}} f$, where $\frac{\partial \mathbf{x}}{\partial \mathbf{u}} = (I + \lambda L)^{-1}$.
(3) Apply a UniformAdam update on $\mathbf{u}$, which is the same as an Adam update except the component-wise division by $\sqrt{v_t}$ is replaced with the component-wise max $\sqrt{||v_t||_\infty}$.
(4) Convert $\mathbf{u}$ back to $\mathbf{x}$ with $\mathbf{x} = (I + \lambda L)^{-1}\mathbf{u}$.

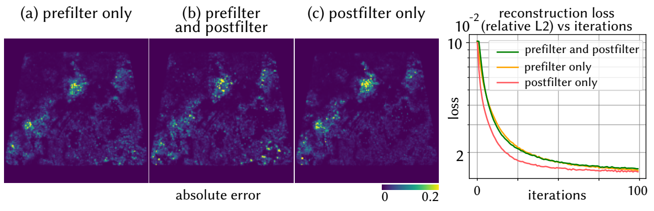


Fig. 8. Prefiltering and postfiltering for texture recovery. We found that all combinations of prefiltering and postfiltering (including both together) worked well for texture recovery. There are still some differences across the methods. Because of the improved learning rate adaptivity for postfiltering in the presence of noise and anisotropy (see Section 1 in supplemental), we found that it was easiest to tune the learning rate for it. Also, when applying both the prefilter and postfilter, the effective filtering width doubles; to compensate for this, we set F to $F/2$ for each of them.

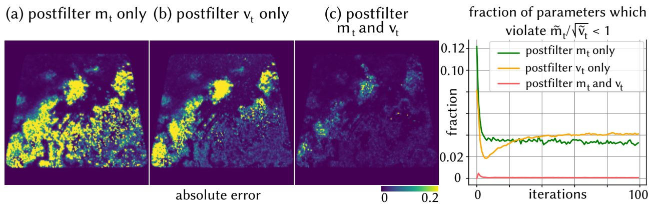


Fig. 9. Postfiltering m_t and v_t. Our method (Algorithm 2) postfilters both m_t and v_t using the same cross-bilateral filter Equation (3). Using the same filter is essential, since this ensures $|\tilde{m}_t|/\sqrt{\tilde{v}_t} < 1$ in the postfiltered step is bounded, similar to Adam which ensures $|m_t|/\sqrt{v_t} < 1$. Filtering only one of m_t or v_t violates this (see graph on right), resulting in much larger step sizes that significantly worsen recovery (left).

Since $\mathbf{x}$ is a linear function of $\mathbf{u}$, we can use the property that $\Delta\mathbf{x} = (I + \lambda L)^{-1}\Delta\mathbf{u}$ to write their update in terms of $\mathbf{x}$ and cast it into our framework as shown in Algorithm 3.

Algorithm 3. Large Steps Update

```
$g_t \leftarrow \nabla_{\mathbf{x}} f_t$
$\nabla_{\mathbf{u}} f_t \leftarrow (I + \lambda L)^{-1} g_t$          # $\frac{\partial \mathbf{x}}{\partial \mathbf{u}} = (I + \lambda L)^{-1}$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\nabla_{\mathbf{u}} f_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)\nabla_{\mathbf{u}} f_t^2$
$\tilde{m}_t \leftarrow (I + \lambda L)^{-1} m_t$            # $\Delta\mathbf{x} = (I + \lambda L)^{-1}\Delta\mathbf{u}$
$\tilde{v}_t \leftarrow ||v_t||_\infty$                      # UniformAdam
$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \alpha \frac{\tilde{m}_t}{\sqrt{\tilde{v}_t} + \varepsilon}$
```

Comparing with Algorithm 2 we see that $\frac{\partial \mathbf{x}}{\partial \mathbf{u}}$, from the chain rule due to the reparameterization, takes the place of the *prefiltering* step, while converting $\mathbf{u}$ back to $\mathbf{x}$ applies the first *postfilter*. The infinity norm on v_t from UniformAdam serves as the second *postfilter*. Thus, Large Steps can be seen as a spatiotemporal optimizer that uses Laplacian smoothing and an infinity norm as its filters.

5.2.2 *Bilateral gradient filtering for meshes.*

For our method, we replace the postfilter on m_t (Algorithm 3, line 5) with a cross-bilateral filter. To apply our cross-bilateral filter on meshes, we use the generalized bilateral filter from Solomon et al. [2014]. We use vertex normals $\hat{\mathbf{n}}$ for the data term. We compute this filter by first sampling $N = 32$ directions $s_i \in S^2$ on the unit sphere using the Sobol sequence [Sobol' 1967], then computing

$$\tilde{\mathbf{h}} = \frac{\sum_{i=1}^{N} \Phi(\hat{\mathbf{n}}, s_i)(I + \lambda L)^{-1} W(\hat{\mathbf{n}}, s_i)\mathbf{h}}{\sum_{i=1}^{N} \Phi(\hat{\mathbf{n}}, s_i)(I + \lambda L)^{-1} W(\hat{\mathbf{n}}, s_i)\mathbf{1}}, \tag{5}$$

where $\mathbf{h}$ is the vector to be filtered, $\mathbf{1}$ is the vector of ones, $W_i = W(\hat{\mathbf{n}}, s_i)$ is the diagonal data matrix, and $\Phi_i = \Phi(\hat{\mathbf{n}}, s_i)$ is the diagonal partition of the unity matrix. Each diagonal element of W_i is computed using the von Mises-Fisher kernel between $\hat{\mathbf{n}}$ and s_i: $\exp(\hat{n} \cdot s_i/\sigma_d)$, and similarly for Φ_i: $\exp(\hat{n} \cdot s_i/\sigma_\phi)$. Although this filter appears to be more expensive to compute compared to simply applying Laplacian smoothing, we can compute it efficiently with minimal overhead. This is done by concatenating all vectors $W(\hat{\mathbf{n}}, s_i)\mathbf{h}$ and $W(\hat{\mathbf{n}}, s_i)\mathbf{1}$ into a single matrix:

$$M = \begin{bmatrix} | & | & & | & | \\ W(\hat{\mathbf{n}}, s_1)\mathbf{h} & W(\hat{\mathbf{n}}, s_i)\mathbf{1} & \cdots & W(\hat{\mathbf{n}}, s_N)\mathbf{h} & W(\hat{\mathbf{n}}, s_N)\mathbf{1} \\ | & | & & | & | \end{bmatrix}, \tag{6}$$

and computing $(I + \lambda L)^{-1} M$ using a single matrix solve. The rest of the filter is then computed using component-wise operations. The overhead of our method over Large Steps is minor (< 10%, see Table 2 in supplemental).

5.2.3 *Results.*

We implement our method in PyTorch [Paszke et al. 2019] using the Cholesky solver based on CHOLMOD [Chen et al. 2008] following Nicolet et al. [2021]. We optimize for meshes using Mitsuba 3 [Jakob et al. 2022] with projective sampling [Zhang et al. 2023]. All experiments ran on an NVIDIA GeForce RTX 3080 GPU.

Mesh recovery. We compare our method against Large Steps on two mesh recovery examples: a cube (Figure 10) and a dragon (Figure 11). For both examples and methods, we use 16 samples per pixel for the forward rendering and 1 sample per pixel for the gradient. We set $\alpha = 1 \cdot 10^{-2}$ for the cube, $\alpha = 3 \cdot 10^{-2}$ for the dragon and $\beta_1 = 0.9, \beta_2 = 0.99$ for both. We apply an exponential learning rate scheduler that reduces the learning rate to a factor of 0.6 at the end of the optimization. We set $\lambda = 19$. We initialize using a sphere with approximately 10k vertices and upsample the dragon at iterations 10, 30, and 60, by remeshing to reduce the average edge length by a factor of 2 every time. The cube does not require remeshing. We use L1 as the loss function and optimize the cube using 10 views and the dragon using 25 views, rendering all views at every iteration.

For our method, we follow Large Steps (Algorithm 3), except we replace the postfilter on m_t with the mesh cross-bilateral filter. We set $\sigma_\phi = 0.4$ and $\sigma_d = 0.5$.

The cube (Figure 10) contains sharp edges and so using our edge-aware cross-bilateral filter is important to prevent the filtering from smoothing over the edges. The dragon (Figure 11) on the other hand contains less sharp edges, but also has thin structures such as scales and horns and contains many more silhouettes. Due to the sparse nature of silhouette gradients, gradient filtering can be seen as a way to interpolate gradients in the interior using values on the silhouettes. Since the silhouettes at any given viewpoint take a form of curves along the surface, this means that vertices that are near but not on the silhouettes receive no silhouette gradients. As a result, Laplacian smoothing will smooth the gradients so that these vertices do receive gradients, but this also means the gradients

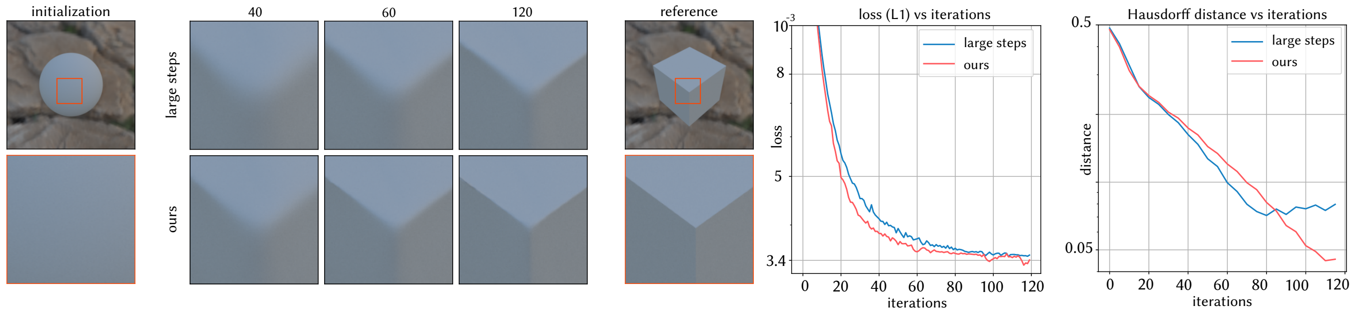


Fig. 10. Inverse mesh recovery (i) The objective is to recover the mesh vertices of a cube from multiple views (10 views used, 1 shown). The gradients are sparse, due to the nature of silhouette gradients. We compare our cross-bilateral filtering method with Large Steps, where both are initialized with a sphere. We visualize the optimization progress over multiple iterations, and plot the L1 loss as well as Hausdorff (geometric) distance on the right. The cube is a good example of a piecewise smooth mesh with hard edges and so our edge-aware cross-bilateral filter is able to converge quickly, while Large Steps over-smooths the edges. Scene adapted from Kloppenheim 06 ©Greg Zaal.

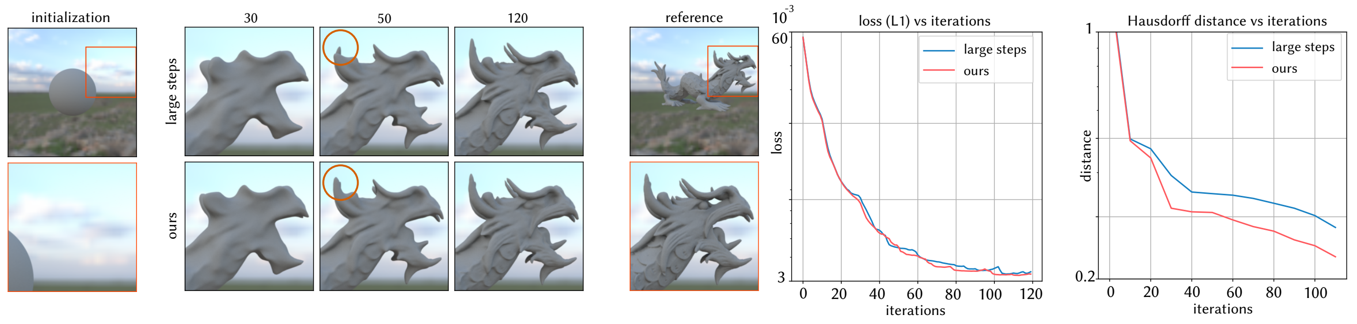


Fig. 11. Inverse mesh recovery (ii) We use the same setup as Figure 10, but optimize for a dragon instead (25 views used, 1 shown). While the dragon does not contain hard edges unlike the cube, it contains thin structures and many more silhouettes. The Laplacian preconditioning from Large Steps smooths the gradients at the silhouettes, causing them to be diluted. Our method is better able to retain the silhouette gradients, resulting in faster convergence on the horn of the dragon and other sharp features. Scenes adapted from Kloppenheim 06 ©Greg Zaal, and Asian Dragon ©Stanford Computer Graphics Laboratory.

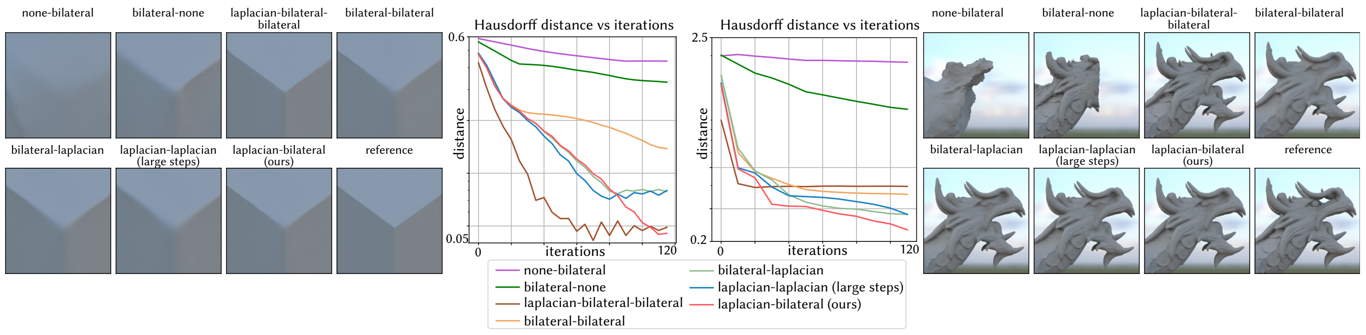


Fig. 12. Prefiltering and Postfiltering for Mesh Recovery. Using the examples from Figure 10 and Figure 11, we compare the effect of different filtering decisions in Algorithm 2. We compare 7 different configurations, each of which are denoted by the prefilter and the postfilter on m_t. For example, "none-bilateral" uses no prefilter, and a cross-bilateral postfilter. "laplacian-bilateral-bilateral" additionally uses a a cross-bilateral postfilter on v_t, whereas all others simply apply the UniformAdam infinity norm. We find that using both a prefilter and postfilter is crucial to provide sufficient smoothing, and using a cross-bilateral filter on the postfilter converges the fastest. In the case of the cube, using a bilateral filter on v_t can provide comparable convergence, though the aggressive updates can sometimes cause small cracks in the edges. Scenes adapted from Kloppenheim 06 ©Greg Zaal, and Asian Dragon ©Stanford Computer Graphics Laboratory.

on the silhouette vertices are diluted. Thus, our method is able to converge faster, since it helps preserve the values on the silhouettes when there are thin structures, such as the horns of the dragon.

Prefiltering and postfiltering ablation. By casting Large Steps into our framework shown in Algorithm 2, an interesting question arises: what combination of filters should be used? We compare 7 different configurations in Figure 12. In addition to the previous experimental settings, we adjust the learning rate for each configuration. Empirically, we see that both a prefilter and postfilter is needed to provide sufficient smoothing. Interestingly, keeping the prefilter as a Laplacian is important, as is is likely that a bilateral prefilter is not sufficient to diffuse the silhouette gradients. We find that applying a Laplacian prefilter and a bilateral postfilter, as well as keeping the UniformAdam infinity norm performs the best across both examples.

6 LIMITATIONS AND FUTURE WORK

Non-spatial parameters. Our method assumes that the parameters exhibit piecewise smoothness spatially. It is unclear whether our method can still be helpful when the parameters lack obvious spatial coherence. Defining proper notion of spatial proximity to generalize our filters would be an exciting research avenue.

Spatiotemporal hyperparameters. Our method introduces two new hyperparameters σ_s (F) and σ_d. Like learning rate adjustment, tuning them can be tedious and recent work on hyperparameter optimization might be helpful [Chandra et al. 2022]. Theoretically, it is also unclear what the ideal tradeoff between stronger temporal filtering over spatial filtering is. Both reduce noise in the gradient, but they introduce different biases. A deeper analysis here could help us pick better hyperparameters.

7 CONCLUSION

We present a spatiotemporal optimizer for inverse recovery. We enforce piecewise smooth gradients, which are well suited to natural signals. Our method enables faster convergence and improves recovery, especially at low sample counts, without any extra storage and with very little overhead. We believe spatiotemporal gradient filtering can also be applied to other gradient-based optimization methods with spatial coherence across parameters.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants 2105806, 2127544, and 2238839, gifts from Adobe, Qualcomm, and Google, the Ronald L. Graham Chair, and the UC San Diego Center for Visual Computing. We thank the anonymous reviewers for their feedback and Keenan Crane for valuable discussions during the project.

REFERENCES

Hyeonjang An, Wonjun Lee, and Bochang Moon. 2023. Adaptively weighted discrete Laplacian for inverse rendering. *Visual Comput.* 39, 8 (2023).

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, Vol. 29.

Dejan Azinović, Tzu-Mao Li, Anton Kaplanyan, and Matthias Nießner. 2019. Inverse Path Tracing for Joint Material and Lighting Estimation. In *Computer Vision and Pattern Recognition.* 2447–2456.

Sai Bangaru, Lifan Wu, Tzu-Mao Li, Jacob Munkberg, Gilbert Bernstein, Jonathan Ragan-Kelley, Fredo Durand, Aaron Lefohn, and Yong He. 2023. SLANG.D: Fast, Modular and Differentiable Shader Programming. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 42, 6 (2023), 1–28.

Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased Warped-Area Sampling for Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6 (2020), 245:1–245:18.

Jonathan T Barron and Ben Poole. 2016. The fast bilateral solver. In *European Conference on Computer Vision.* 617–632.

Pablo Bauszat, Martin Eisemann, and Marcus Magnor. 2011. Guided image filtering for interactive high-quality global illumination. *Comput. Graph. Forum (Proc. EGSR)* 30, 4 (2011), 1361–1368.

Yash Belhe, Bing Xu, Sai Praveen Bangaru, Ravi Ramamoorthi, and Tzu-Mao Li. 2024. Importance Sampling BRDF Derivatives. *ACM Trans. Graph.* 43, 3 (2024).

Pravin Bhat, Brian Curless, Michael Cohen, and C Lawrence Zitnick. 2008. Fourier analysis of the 2D screened Poisson equation for gradient domain problems. In *European Conference on Computer Vision.* 114–128.

Volker Blanz and Thomas Vetter. 1999. A morphable model for the synthesis of 3D faces. In *SIGGRAPH.* 187–194.

Antoni Buades, Bartomeu Coll, and J-M Morel. 2005. A non-local algorithm for image denoising. In *Computer Vision and Pattern Recognition*, Vol. 2. IEEE, 60–65.

Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. 2022. Gradient Descent: The Ultimate Optimizer. In *Advances in Neural Information Processing Systems*, Vol. 35. 8214–8225.

Wesley Chang, Venkataram Sivaram, Derek Nowrouzezahrai, Toshiya Hachisuka, Ravi Ramamoorthi, and Tzu-Mao Li. 2023. Parameter-space ReSTIR for Differentiable and Inverse Rendering. In *SIGGRAPH Conference Proceedings.* 10 pages.

Jiawen Chen, Sylvain Paris, and Frédo Durand. 2007. Real-time Edge-aware Image Processing with the Bilateral Grid. *ACM Trans. Graph. (Proc. SIGGRAPH)* 26, 3 (2007).

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. 2024. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems*, Vol. 36.

Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Softw.* 35, 3, Article 22 (2008), 14 pages.

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik Lensch. 2010. Edge-avoiding À-Trous wavelet transform for fast global illumination filtering. In *High Performance Graphics.* 67–75.

Yishun Dou, Zhong Zheng, Qiaoqiao Jin, Rui Shi, Yuhan Li, and Bingbing Ni. 2024. Differentiable Micro-Mesh Construction. In *Computer Vision and Pattern Recognition.* 4294–4303.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* 12 (2011), 2121–2159.

Elmar Eisemann and Frédo Durand. 2004. Flash Photography Enhancement via Intrinsic Relighting. *ACM Trans. Graph. (Proc. SIGGRAPH)* 23, 3 (2004), 673–678.

István Faragó and János Karátson. 2002. *Numerical solution of nonlinear elliptic problems via preconditioning operators: Theory and applications.* Vol. 11.

David J Field. 1994. What is the goal of sensory coding? *Neural computation* 6, 4 (1994), 559–601.

Eduardo S. L. Gastal and Manuel M. Oliveira. 2012. Adaptive Manifolds for Real-time High-dimensional Filtering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 31, 4 (2012), 33:1–33:13.

Ioannis Gkioulekas, Shuang Zhao, Kavita Bala, Todd Zickler, and Anat Levin. 2013. Inverse Volume Rendering with Material Dictionaries. *ACM Trans. Graph.* 32, 6 (2013), 162:1–162:13.

Gabriel Goh. 2017. Why momentum really works. *Distill* 2, 4 (2017), e6.

Kaiming He, Jian Sun, and Xiaoou Tang. 2013. Guided image filtering. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 6 (2013), 1397–1409.

Yong He, Kayvon Fatahalian, and Tim Foley. 2018. Slang: Language Mechanisms for Extensible Real-time Shading Systems. *ACM Trans. Graph. (Proc. SIGGRAPH)* 37, 4 (2018), 141:1–141:13.

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. 2022. *Mitsuba 3 renderer.* https://mitsuba-renderer.org.

Rie Johnson and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems.* 315–323.

Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. 2003. Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graph. (Proc. SIGGRAPH)* 22, 3 (2003), 943–949.

Miyoun Jung, Ginmo Chung, Ganesh Sundaramoorthi, Luminita A Vese, and Alan L Yuille. 2009. Sobolev gradients and joint variational image segmentation, denoising, and deblurring. In *Computational Imaging VII*, Vol. 7246. 131–143.

Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Matias Koskela, Kalle Immonen, Markku Mäkitalo, Alessandro Foi, Timo Viitanen, Pekka Jääskeläinen, Heikki Kultala, and Jarmo Takala. 2019. Blockwise Multi-Order Feature Regression for Real-Time Path-Tracing Reconstruction. *ACM Trans. Graph.* 38, 5, Article 138 (2019), 14 pages.

Dilip Krishnan and Rob Fergus. 2009. Fast image deconvolution using hyper-Laplacian priors. In *Advances in Neural Information Processing Systems*, Vol. 22.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37, 6 (2018), 222:1–222:11.

Yang Li, Aljaz Bozic, Tianwei Zhang, Yanli Ji, Tatsuya Harada, and Matthias Nießner. 2020. Learning to optimize non-rigid tracking. In *Computer Vision and Pattern Recognition*. 4910–4918.

Yichen Li, Peter Yichen Chen, Tao Du, and Wojciech Matusik. 2023. Learning preconditioners for conjugate gradient PDE solvers. In *International Conference on Machine Learning*. 19425–19439.

Hong Liu, Zhiyuan Li, David Leo Wright Hall, Percy Liang, and Tengyu Ma. 2024. Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training. In *International Conference on Learning Representations*.

Wei Liu, Pingping Zhang, Xiaolin Huang, Jie Yang, Chunhua Shen, and Ian Reid. 2020. Real-time image smoothing via iterative least squares. *ACM Trans. Graph.* 39, 3 (2020), 1–24.

Peyman Milanfar. 2012. A tour of modern image filtering: New insights and methods, both practical and theoretical. *IEEE signal processing magazine* 30, 1 (2012), 106–128.

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large Steps in Inverse Rendering of Geometry. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40, 6 (2021).

Merlin Nimier-David, Thomas Müller, Alexander Keller, and Wenzel Jakob. 2022. Unbiased Inverse Volume Rendering with Differential Trackers. *ACM Trans. Graph. (Proc. SIGGRAPH)* 41, 4, Article 44 (2022), 20 pages.

Stanley Osher, Bao Wang, Penghang Yin, Xiyang Luo, Farzin Barekat, Minh Pham, and Alex Lin. 2018. Laplacian Smoothing Gradient Descent. *arXiv preprint arXiv:1806.06317* (2018).

Sylvain Paris and Frédo Durand. 2009. A Fast Approximation of the Bilateral Filter Using a Signal Processing Approach. *Int. J. Comput. Vision* 81, 1 (2009), 24–52.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*. 8024–8035.

Fabian Pedregosa. 2023. A Hitchhiker's Guide to Momentum. In *The Second Blogpost Track at ICLR 2023*.

Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe, and Kentaro Toyama. 2004. Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. (Proc. SIGGRAPH)* 23, 3 (2004), 664–672.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* (3rd ed.). Morgan Kaufmann Publishers Inc. 1266 pages.

Robert J Renka. 2006. A simple explanation of the Sobolev gradient method. *Unpublished, University of North Texas* (2006).

Robert J Renka and JW Neuberger. 1995. Minimal surfaces and Sobolev gradients. *Siam journal on scientific computing* 16, 6 (1995), 1412–1427.

Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *Ann. Math. Statist.* 22, 3 (1951), 400–407.

Yaniv Romano, Michael Elad, and Peyman Milanfar. 2017. The little engine that could: Regularization by denoising (RED). *SIAM Journal on Imaging Sciences* 10, 4 (2017), 1804–1844.

Nicolas Roux, Mark Schmidt, and Francis Bach. 2012. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems*, Vol. 25.

Leonid I Rudin, Stanley Osher, and Emad Fatemi. 1992. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena* 60, 1-4 (1992), 259–268.

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *High Performance Graphics*. 2.

Shai Shalev-Shwartz and Tong Zhang. 2013. Stochastic dual coordinate ascent methods for regularized loss minimization. *J. Mach. Learn. Res.* 14, 1 (2013).

Eero P Simoncelli and Edward H Adelson. 1996. Noise removal via Bayesian wavelet coring. In *International Conference on Image Processing*, Vol. 1.

I.M Sobol'. 1967. On the distribution of points in a cube and the approximate evaluation of integrals. *U. S. S. R. Comput. Math. and Math. Phys.* 7, 4 (1967), 86–112.

Justin Solomon, Keenan Crane, Adrian Butscher, and Chris Wojtan. 2014. A general framework for bilateral and mean shift filtering. *arXiv preprint arXiv:1405.4734* 1, 2 (2014), 3.

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. , 26–31 pages.

Carlo Tomasi and Roberto Manduchi. 1998. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*. 839–846.

Antonio Torralba and Aude Oliva. 2003. Statistics of natural image categories. *Network: computation in neural systems* 14, 3 (2003), 391.

Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. 2007. On early stopping in gradient descent learning. *Constructive Approximation* 26, 2 (2007), 289–315.

Zihan Yu, Cheng Zhang, Derek Nowrouzezahrai, Zhao Dong, and Shuang Zhao. 2022. Efficient Differentiation of Pixel Reconstruction Filters for Path-Space Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 41, 6, Article 191 (2022).

Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. 2021. Monte Carlo estimators for differential light transport. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (2021), 1–16.

Cheng Zhang, Zhao Dong, Michael Doggett, and Shuang Zhao. 2021a. Antithetic sampling for Monte Carlo differentiable rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (2021), 1–12.

Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. 2020. Path-Space Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 39, 6 (2020), 143:1–143:19.

Cheng Zhang, Zihan Yu, and Shuang Zhao. 2021b. Path-space differentiable rendering of participating media. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (2021), 1–15.

Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob. 2023. Projective Sampling for Differentiable Rendering of Geometry. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 42, 6 (2023).