Parameter-space ReSTIR for Differentiable and Inverse Rendering

Wesley Chang
wec022@ucsd.edu
University of California San Diego
USA

Venkataram Sivaram
ves223@ucsd.edu
University of California San Diego
USA

Derek Nowrouzezahrai
derek@cim.mcgill.ca
McGill University
Canada

Toshiya Hachisuka
thachisu@uwaterloo.ca
University of Waterloo
Canada

Ravi Ramamoorthi
ravir@cs.ucsd.edu
University of California San Diego
USA

Tzu-Mao Li
tzli@ucsd.edu
University of California San Diego
USA

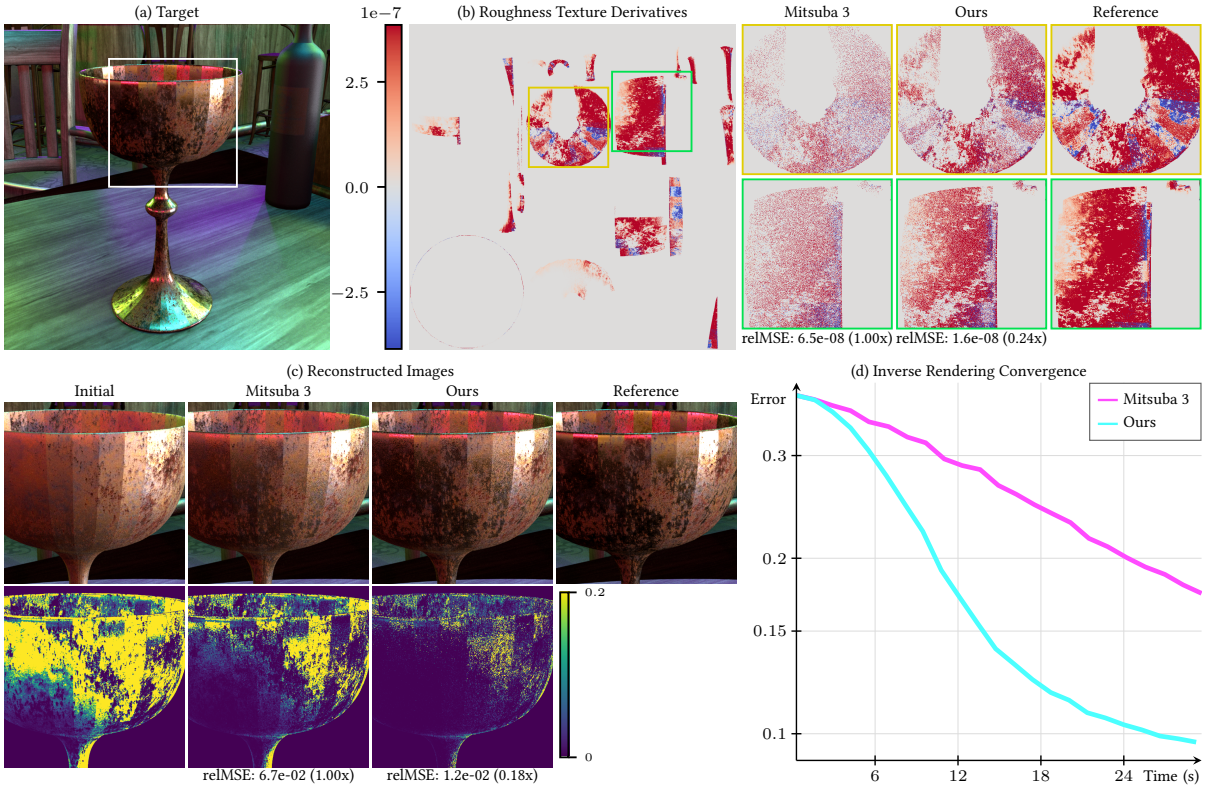

Figure 1: We optimize a 2048×2048 roughness texture from (a) a single view of the Chalice, reusing samples across optimization iterations to (b) reduce derivative error in equal-time compared to Mitsuba 3. This leads to (d) faster inverse rendering convergence, and (c) images that more closely match our target. Scene adapted from Chalice, goblet, cup ©SusanKing, Bistro ©Amazon Lumberyard, and Ballroom ©Sergej Majboroda.

ABSTRACT

Differentiable rendering is frequently used in gradient descent-based inverse rendering pipelines to solve for scene parameters – such as reflectance or lighting properties – from target image inputs. Efficient computation of accurate, low variance gradients is critical for rapid convergence. While many methods employ variance reduction strategies, they operate independently on each gradient descent iteration, requiring large sample counts and computation. Gradients may however vary slowly between iterations, leading to unexplored potential benefits when reusing sample information to exploit this coherence. We develop an algorithm to reuse Monte Carlo gradient samples between gradient iterations, motivated by reservoir-based temporal importance resampling in forward rendering. Direct application of this method is not feasible, as we are computing *many* derivative estimates (i.e., one per optimization parameter) instead of a *single* pixel intensity estimate; moreover, each

SIGGRAPH '23 Conference Proceedings, August 06–10, 2023, Los Angeles, CA, USA

ACM ISBN 979-8-4007-0159-7/23/08.
https://doi.org/10.1145/3588432.3591512

of these gradient estimates can affect multiple pixels, and gradients can take on negative values. We address these challenges by reformulating differential rendering integrals in parameter space, developing a new resampling estimator that treats negative functions, and combining these ideas into a reuse algorithm for inverse texture optimization. We significantly reduce gradient error compared to baselines, and demonstrate faster inverse rendering convergence in settings involving complex direct lighting and material textures.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**.

KEYWORDS

differentiable rendering, inverse rendering, resampling

ACM Reference Format:

Wesley Chang, Venkataram Sivaram, Derek Nowrouzezahrai, Toshiya Hachisuka, Ravi Ramamoorthi, and Tzu-Mao Li. 2023. Parameter-space ReSTIR for Differentiable and Inverse Rendering. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings (SIGGRAPH '23 Conference Proceedings), August 06–10, 2023, Los Angeles, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3588432.3591512

1 INTRODUCTION

Efficiently computing accurate derivatives of an image loss with respect to scene parameters is a topic of growing importance in inverse rendering. Modern differentiable rendering algorithms estimate these derivatives via Monte Carlo integration, and excessive variance in derivative estimation leads to inefficient optimization in its application to inverse rendering. While some work has addressed the efficiency of differentiable rendering [Zhao et al. 2021, 2020], to our knowledge, *all existing methods exclusively treat a single image at a time* during optimization.

Iterative optimization updates the target image estimate by stepping along the estimated gradient of the loss function. Since the image changes slowly between iterations, it is likely wasteful to completely discard samples from previous iterations when computing the current iteration's gradients. We present an efficient method to *reuse samples across iterations* and demonstrate its benefits in the inverse rendering setting.

We are motivated by the recent ReSTIR work in forward rendering [Bitterli et al. 2020; Lin et al. 2022], which reuses samples across frames (temporal reuse) and from neighboring pixels (spatial reuse), to greatly boost the effective number of samples at each pixel.

We devise a novel adaptation of the temporal reuse in ReSTIR to enable sample reuse across gradient descent iterations in inverse rendering. An efficient application of ReSTIR to gradient-descent iterations is not immediately evident: first, each pixel needs estimation of a large number of derivatives and storing many reservoirs per pixel leads to both an intractable memory footprint and computation time; second, unlike forward rendering where integrands are non-negative, gradient estimates can yield negative values, and ReSTIR can only handle non-negative integrands. To rectify these limitations, we make the following contributions:

- *Parameter-Space Differentiable Rendering (Sec. 2):* Forward rendering outputs pixel intensities whereas differentiable rendering computes vector-valued derivatives of a loss function with respect to parameters, i.e., the loss gradient in *parameter space*. We show how this discrepancy in output spaces (i.e., images versus parameters) necessitates a reformulation of the underlying integrals to adapt ReSTIR to differentiable rendering.
- *Resampling with Positive and Negative Functions (Sec. 3):* We extend generalized resampled importance sampling (GRIS) [Lin et al. 2022] to functions that may be both positive and negative, introducing a *positivization technique* for resampling.
- *Application to Inverse Rendering with Textures (Sec. 4 and 5):* Finally, we develop a ReSTIR-based differentiable rendering algorithm that reuses samples across gradient descent iterations (Sec. 4), and demonstrate its effectiveness for optimizing textures that modulate Disney's principled BSDF [Burley 2012].

We demonstrate our method on challenging scenarios of single image inverse rendering with complex direct lighting and high-resolution material textures. Our results in Fig. 1 and Sec. 5 show substantial reductions in gradient error and faster convergence compared to the baseline implementation.

1.1 Relation to Previous Work

Much of existing work in differentiable rendering treat the problem of discontinuities [Bangaru et al. 2020; Li et al. 2018; Loper and Black 2014; Loubet et al. 2019; Zhang et al. 2020], whereas we focus on the piecewise continuous portion of the rendering integrand and leave combination with these methods to future work.

Zeltner et al. [2021] addressed the intricacies behind developing Monte Carlo derivative estimators, and Zhang et al. [2021] and Yu et al. [2022] applied antithetic sampling to exploit symmetries in these derivatives. Our work addresses the *reuse* of information *across* gradient descent iterations, and is orthogonal to these works.

Conventional automatic differentiation applied to path tracing results in memory usage that is linear in the number of scattering events, and so radiative and path replay backpropagation [Nimier-David et al. 2020; Vicini et al. 2021] address this by recomputing required information instead of caching it at path vertices. We focus on direct lighting and so memory is less of an issue, but it is possible to combine our method with these approaches to extend to indirect illumination [Lin et al. 2022; Ouyang et al. 2021].

Nimier-David et al. [2021] proposed a texture-space sampling method that converges more uniformly in applications of material reconstruction from video. Our parameter-space formulation is relevant here, but with the different goal of reusing samples across gradient descent iterations. Consequently, we require a mathematical reformulation of derivative computations to define a target function suitable for resampling.

2 PARAMETER-SPACE DIFFERENTIABLE RENDERING

Physically-based differentiable rendering techniques typically estimate derivatives of the rendering integral per pixel in *image space*, as this approach allows for straightforward computation of an image loss for inverse optimization. We show that this approach is not necessarily suitable in combination with a ReSTIR-like algorithm, and we instead propose to directly estimate derivatives in *parameter space*, which enables intuitive and efficient sample reuse.

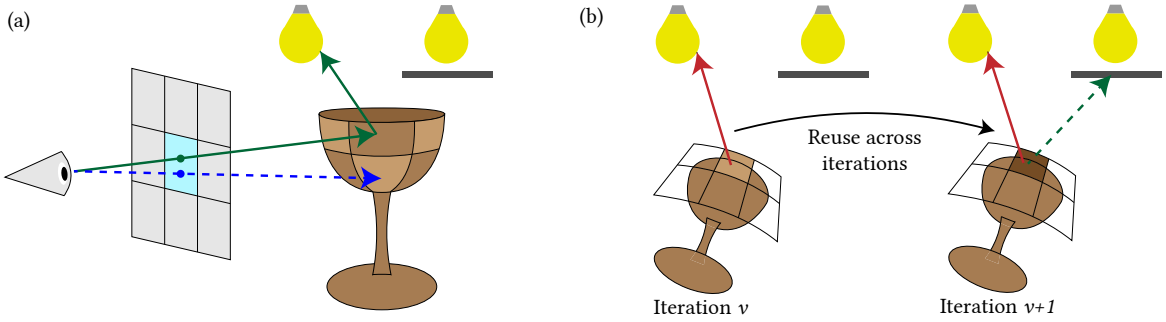

Figure 2: Overview. We aim to recover parameters such as the color texture of the chalice. Since multiple texels may be visible through each pixel (e.g. the texel touched by the green path and the texel touched by the dashed blue path in (a), among others), direct application of ReSTIR to differentiable rendering would result in many reservoirs per pixel, each corresponding to the derivative of the pixel intensity with respect to each texel. We therefore reformulate differentiable rendering in parameter space, which allows us to instead store one reservoir for each parameter, or texel, which accumulates derivative estimates from all relevant pixels. Our algorithm first (a) generates paths from the camera, and resamples these candidates in each texel's reservoir. We then (b) reuse samples at each texel across gradient descent iterations. For example, if we draw an occluded sample (dashed green path) at iteration $v+1$, we can still obtain an unoccluded sample by reusing the sample from iteration v (red path). Finally, we use each final sample to compute the derivatives at the texel.

2.1 Forward and differentiable rendering

Forward rendering techniques synthesize images by estimating intensities I_j for pixels $j = 1, \dots, n$ with integrals [Kajiya 1986; Veach 1998] of the form

$$I_j(\boldsymbol{\pi}) = \int_\Omega h_j(\mathbf{x}) f_c(\mathbf{x}, \boldsymbol{\pi}) \, \mathrm{d}\mu(\mathbf{x}), \tag{1}$$

where Ω is the space of light paths, $\mathbf{x}$ is an individual path, h_j is the filter for pixel j, and f_c is the measurement contribution function. We use $\boldsymbol{\pi}$ to denote the set of scene parameters and assume that the filter h_j is independent of $\boldsymbol{\pi}$. We refer to the space spanned by the parameters $\boldsymbol{\pi}$ as the *parameter space*. Monte Carlo rendering algorithms typically output an image by estimating the above integral per pixel j.

Differentiable rendering techniques estimate partial derivatives of these integrals with respect to individual parameters π_i (e.g. texels in a texture) for parameter indices $i = 1, \dots, \ell$,

$$\begin{aligned} \partial_{\pi_i} I_j(\boldsymbol{\pi}) &= \partial_{\pi_i} \left[\int_\Omega h_j(\mathbf{x}) f_c(\mathbf{x}, \boldsymbol{\pi}) \, \mathrm{d}\mu(\mathbf{x}) \right] \\ &= \int_\Omega h_j(\mathbf{x}) \partial_{\pi_i} f_c(\mathbf{x}, \boldsymbol{\pi}) \, \mathrm{d}\mu(\mathbf{x}), \end{aligned} \tag{2}$$

where $\partial_{\pi_i} := \partial / \partial_{\pi_i}$. When f_c is discontinuous with respect to the parameter being differentiated, the integral may contain Dirac delta distributions and require special treatment [Li et al. 2018; Zhang et al. 2019]. We focus on the case where f_c is continuous with respect to π_i (e.g., BRDF parameters), and leave discontinuity handling to future work. Compared to estimating intensities, estimating derivatives requires a separate integral *per parameter*, resulting in a gradient $\partial_{\boldsymbol{\pi}} I_j = \left(\partial_{\pi_1} I_j, \dots, \partial_{\pi_\ell} I_j\right)$ per pixel.

Most differentiable rendering algorithms estimate the entire gradient independently at each pixel. Direct application of ReSTIR poses a problem here. In ReSTIR, each and every pixel needs to store a sample in its *reservoir*, resulting in storage cost proportional to the number of pixels. This cost quickly precludes its application to differentiable rendering where each pixel would now need to retain reservoirs that store samples for derivatives with respect to *all the parameters*. For applications such as optimization of scene textures, the number of parameters is proportional to the number of texels, i.e., easily in the millions. The storage cost of this naïve application is proportional to the number of pixels times the number of parameters. While it is possible to reduce this cost by keeping only reservoirs for parameters that contribute to each pixel, determining which parameters affect each pixel prior to performing differentiable rendering is generally not feasible. See Fig. 2 for a visual example of the problem.

2.2 From Pixel-centric to Parameter-centric Estimators

We observe that, in inverse rendering via differentiable rendering, the final target quantity is a *single* gradient vector of a loss function with respect to the parameters, rather than the gradients of all the pixels individually. In other words, the output of differentiable rendering lives in the *parameter space*, while forward rendering outputs pixels. To recapitulate, inverse rendering techniques aim to solve the minimization problem

$$\hat{\boldsymbol{\pi}} = \arg\min_{\boldsymbol{\pi}} \mathcal{L}(\boldsymbol{I}(\boldsymbol{\pi}), \boldsymbol{G}) \tag{3}$$

where $\mathcal{L}$ is a differentiable loss function, $\boldsymbol{I}$ is the rendered image, and $\boldsymbol{G}$ is a reference image. We solve this problem by using gradient-based optimization methods, which need a gradient of the loss function with respect to parameters. The derivative of the loss function with respect to a parameter π_i is then

$$\partial_{\pi_i} \mathcal{L} = \partial_{\boldsymbol{I}} \mathcal{L} \cdot \partial_{\pi_i} \boldsymbol{I} = \sum_{j=1}^{n} \partial_{I_j} \mathcal{L} \cdot \partial_{\pi_i} I_j, \tag{4}$$

or the dot product between the *adjoint rendering* $\partial_{\boldsymbol{I}} \mathcal{L}$ [Nimier-David et al. 2020] and the derivatives of all the pixels $\partial_{\pi_i} \boldsymbol{I}$. When estimating the dot product using Monte Carlo, we require that

$\boldsymbol{I}$ and $\partial_{\pi_i}\boldsymbol{I}$ are uncorrelated [Azinović et al. 2019] and that $\partial_{\boldsymbol{I}}\mathcal{L}$ is affine in $\boldsymbol{I}$ in order for the result to be unbiased. Commonly used loss functions such as the l^2-norm and relMSE satisfy these requirements. Differentiable rendering estimates the derivatives per pixel $\partial_{\pi_i} I_j$ independently. Each parameter in each pixel involves estimation of Equation 2.

We noticed that one can rewrite Equation 4 as a single integral by substituting Equation 2 and rearranging terms:

$$\begin{aligned}\partial_{\pi_i}\mathcal{L} &= \sum_{j=1}^{n} \partial_{I_j}\mathcal{L} \int_\Omega h_j(\mathbf{x})\partial_{\pi_i} f_c(\mathbf{x},\boldsymbol{\pi})\,\mathrm{d}\mu(\mathbf{x}) \\ &= \int_\Omega \underbrace{\left[\sum_{j=1}^{n} \partial_{I_j}\mathcal{L}\cdot h_j(\mathbf{x})\right]}_{\mathrm{w}(\mathbf{x})} \partial_{\pi_i} f_c(\mathbf{x},\boldsymbol{\pi})\,\mathrm{d}\mu(\mathbf{x}) \\ &= \boxed{\int_\Omega \mathrm{w}(\mathbf{x})\partial_{\pi_i} f_c(\mathbf{x},\boldsymbol{\pi})\,\mathrm{d}\mu(\mathbf{x}),} \end{aligned} \tag{5}$$

where w represents the weight of path $\mathbf{x}$ due to the adjoint rendering and pixel filter of its location on the image plane. The function w hides the sum over pixels. We refer to this integral as the **parameter-space differential rendering equation**.

This formulation clarifies that one does not need to first estimate a derivative per pixel and sum them over all the pixels to estimate a derivative of the loss, but one can just directly estimate it by a single integral. When performing Monte Carlo integration for the parameter π_i, a sampled path $\mathbf{x}$ has the contribution equal to the differential measurement $\partial_{\pi_i} f_c$ times the weight w when the path has non-zero contribution to the parameter (e.g., a non-occluded path where a vertex lies at a surface point that uses parameter π_i). The differential measurement $\partial_{\pi_i} f_c$ represents how a change in the parameter affects the radiance carried by the path, and the weight w characterizes how this change affects the loss $\mathcal{L}$. The method of Nimier-David et al. [2021] for textures is a special case of our formulation. This formulation allows our ReSTIR-based method to keep only one reservoir per parameter, rather than a reservoir per parameter per pixel.

3 GRIS FOR DERIVATIVES

We present our extension to generalized resampled importance sampling (GRIS) [Lin et al. 2022] for estimating derivatives.

3.1 Review of RIS, GRIS, and ReSTIR

Resampled importance sampling (RIS) as presented by Talbot et al. [2005] approximately samples from a distribution proportional to a target function q by resampling from a pool of independently sampled candidates $\{x_1,\ldots,x_M\}$ generated from another source distribution p. When a single candidate is selected, this process forms a *one-sample* RIS estimator:

$$\langle F\rangle_{\mathrm{ris}} = \frac{f(x_z)}{q(x_z)}\frac{1}{M}\sum_{s=1}^{M}\frac{q(x_s)}{p(x_s)}, \tag{6}$$

where f is the integrand, and x_z is the resampled element. If $p>0$ and $q>0$ whenever $f\neq 0$, this estimator is unbiased, with variance

$$V\left[\langle F\rangle_{\mathrm{ris}}\right] = \frac{1}{M}V\left[\frac{f}{p}\right] + \left(1-\frac{1}{M}\right)V\left[\frac{f}{q^*}\right], \tag{7}$$

where q^* represents the *normalized* target density. Intuitively: as the number of candidates M increases, the closer the sample x_z becomes to being distributed according to q^* rather than p. Often q is chosen to approximate f so that the variance of importance sampling according to q^* is significantly lower than sampling according to p. In RIS, the exact probability density of the selected sample $p_z(x_z)$ cannot be computed in a closed form [Bitterli et al. 2020].

To increase the number of candidates for each pixel in rendering, ReSTIR [Bitterli et al. 2020] reuses samples generated from neighboring pixels (spatial) and previous frames (temporal) as candidates in RIS. To avoid storing every candidate in memory, ReSTIR leverages weighted-reservoir sampling [Chao 1982]. Each pixel stores a reservoir, which contains the selected sample x_z, a sum of weights $w_{sum}=\sum_s w_s$, where $w_s = q(x_s)/p(x_s)$, and the number of candidates M. The sample is selected in a single pass over the candidates by accumulating w_{sum}, incrementing M, and replacing the sample in the reservoir with candidate x_s with probability $w_s/\sum_{t\le s} w_t$.

Because ReSTIR reuses samples from spatiotemporal neighbors which are themselves resampled previously, two issues arise: 1) the source distribution p becomes intractable since p_z cannot be evaluated, and 2) the samples x_s can be correlated and can originate from different domains Ω_s, while RIS assumes a single source domain. GRIS addresses these issues using the following estimator:

$$\langle F\rangle_{\mathrm{gris}} = f(y_z)W \text{ with } W = \frac{1}{q(y_z)}\sum_{s=1}^{M} m_s(y_s)q(y_s)W_s\left|\frac{\partial T_s}{\partial x_s}\right|, \tag{8}$$

where $y_s = T_s(x_s)$, $T_s:\Omega_s\to \mathrm{dom}\, f$ is a *shift mapping* [Kettunen et al. 2015] that maps samples x_s into y_s in the integrand's domain (i.e., transforms a path generated at a neighboring pixel to a path for the current pixel), $|\partial T_s/\partial x_s|$ is the Jacobian of this mapping, and the *unbiased contribution weight* W is an unbiased estimator of the intractable reciprocal density $1/p_z(y_z)$. Then, in subsequent resampling passes, y_z and W from the previous pass can be used as x_s and W_s, respectively. For the multiple importance sampling (MIS) [Veach and Guibas 1995] weight m_s, Lin et al. [2022] generalized Talbot's [2005] MIS weights, which follows the standard balance heuristic:

$$m_s(y) = \frac{q_{\leftarrow s}(y)}{\sum_{t=1}^{M} q_{\leftarrow t}(y)}, \tag{9}$$

but replaces densities with $q_{\leftarrow s}(y)$, which evaluates the target function at x in the original domain of the s-th sample:

$$q_{\leftarrow s}(y) = \begin{cases} q_s(T_s^{-1}(y))\left|\partial T_s^{-1}/\partial y\right|, & \text{if } y\in T_s(\mathrm{supp}\ x) \\ 0 & \text{otherwise.}\end{cases} \tag{10}$$

3.2 Positivized RIS

RIS and GRIS have only been applied to problems in forward rendering where the integrand f is non-negative, which is no longer the case for derivatives. To be precise, while f can be signed and RIS will still be a valid estimator, the target function q must be

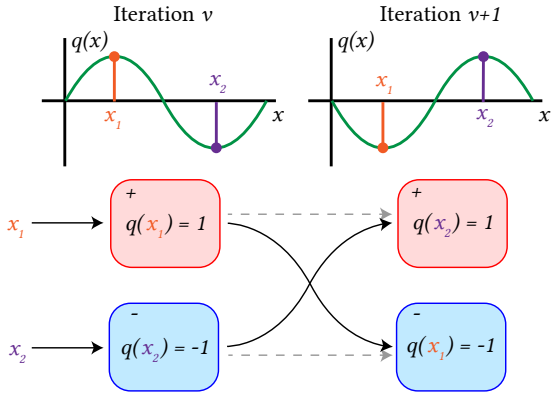

Figure 3: PGRIS WITH CHANGING TARGET FUNCTIONS. At iteration v, the target function is $q(x) = \sin(x)$, and so sample x_1 is positive and sample x_2 is negative. At iteration $v+1$, the target function changes to $q(x) = -\sin(x)$. If we only reuse x_1 in the positive (red) estimator and x_2 in the negative (blue) estimator (gray dashed arrows only), then both samples have zero contribution due to incorrect signs. Instead, we reuse both samples in both estimators to ensure they end up in the estimator with the correct sign.

non-negative, since it represents an unnormalized probability distribution. One possible workaround is to use a target function that is always non-negative. For example, we can take some function g that approximates f, and set the target function to be $q = |g|$. This approach leads to an unbiased estimator, but unlike the case with non-negative functions, the resulting RIS estimator never achieves zero variance even if we set $q = |f|$ and $M = \infty$. The variance of this RIS estimator converges only to the variance of a single sample drawn exactly from the target density, which is $V\left[f/|f|^*\right]$ (Equation 7). In other words, the variance will never approach zero due to differences in sign.

We borrow ideas from a method known as *positivization* [Owen and Zhou 2000; Owen 2013] to handle this problem. The positivization technique decomposes the integrand into positive and negative parts as $f(x) = f_+(x) - f_-(x)$ where we define $f_+(x) = \max(f(x), 0)$ and $f_-(x) = \max(-f(x), 0)$. By constructing densities p_+ and p_- proportional to f_+ and f_-, it is possible to obtain a zero-variance estimator using two samples: one each from the positive and negative densities.

One challenge in positivization is that sampling from densities p_+ and p_- is generally difficult for arbitrary functions. We propose to apply RIS to approximately solve this problem. By applying positivization to the target function q, we obtain a **positivized RIS (PRIS)** estimator whose variance converges to zero with an increasing number of candidates:

$$q_+(x) = \max(q(x), 0) \qquad q_-(x) = \max(-q(x), 0)$$

$$\langle F\rangle_{\text{pris}} = \frac{f(x_{z+})}{q_+(x_{z+})}\frac{1}{M}\sum_{s=1}^{M}\frac{q_+(x_s)}{p(x_s)} + \frac{f(x_{z-})}{q_-(x_{z-})}\frac{1}{M}\sum_{s=1}^{M}\frac{q_-(x_s)}{p(x_s)}, \quad (11)$$

where q is some signed function approximating f. We use the same source pdf p for both positive and negative estimators. Additionally, we use the same set of candidates for both estimators. Since each candidate is either positive or negative (or zero), it only has a non-zero contribution in at most one of the two estimators.

Similarly, we apply positivization to GRIS, to obtain a **positivized GRIS (PGRIS)** estimator:

$$\langle F\rangle_{\text{pgris}} = \frac{f(y_{z+})}{q_+(y_{z+})}\sum_{s=1}^{M} m_s(y_s)q_+(y_s)W_s\left|\frac{\partial T_s}{\partial x_s}\right| + \frac{f(y_{z-})}{q_-(y_{z-})}\sum_{s=1}^{M} m_s(y_s)q_-(y_s)W_s\left|\frac{\partial T_s}{\partial x_s}\right|. \quad (12)$$

One detail is in the reuse of samples as candidates. In ReSTIR and GRIS, selected samples are usually reused as candidates in subsequent resampling steps. At the same time, the target functions can change in subsequent resampling steps. In our application (discussed in Sec. 4), selected samples might change the signs between resampling steps. It is thus important to use selected samples from *both* positive and negative estimators as candidates for *both* estimators in the current step. For example, a previously positive sample in the positive estimator may end up being a negative sample used now in the negative estimator. This additional step ensures that all candidates have a non-zero selection probability in either estimator. Fig. 3 illustrates an example where this step is required to ensure both selected samples are not discarded due to changing signs.

PGRIS MIS Weights. The MIS weight m_s in PGRIS is a straightforward substitution of the one in GRIS. The main difference is that the positive and negative estimators are considered different MIS strategies and both must be considered in the MIS weights. We denote this with $q_{\leftarrow s,\pm}$, which evaluates either q_+ or q_- based on whether q_s is a positive or negative strategy. Based on this idea, our weight function m_s evaluates to

$$m_s(y) = \frac{N_s q_{\leftarrow s,\pm}(y)}{\sum_{t=1}^{M} N_t q_{\leftarrow t,\pm}(y)}, \quad (13)$$

where N_s are the *confidence weights* of strategy s, which are the total number of candidates capped to a maximum M_{cap} to limit the weight of the reused samples (*M-capping*) [Lin et al. 2022].

4 APPLICATION TO TEXTURE OPTIMIZATION

We provide an application of our parameter-space formulation and PGRIS estimator to an inverse rendering problem of determining textures. We focus on optimizing BRDF parameter textures; thus the number of parameters is proportional to the number of texels. While ReSTIR stores one reservoir per pixel and performs spatiotemporal reuse, we store two reservoirs (positive and negative) per-texel and reuse samples across gradient-descent iterations. Algorithm 1 describes our reservoir data structure in more detail. Unlike ReSTIR, we generate candidates from the camera but store reservoirs at each texel (see Section 4.1), and so when parallelizing the algorithm across pixels, synchronization is required at each texel, since multiple candidates from different pixels may have non-zero contribution at the same texel (see Fig. 2).

Algorithm 1: Parameter-space Reservoir

```
class Reservoir:
    x ← 0          # Selected sample
    w_sum ← 0      # Sum of weights (q / p)
    M ← 0          # Number of candidates / confidence weight
    W ← 0          # Unbiased contribution weight
    lock ← 0       # Lock to synchronize the reservoir update

    function update(x_s , w_s):
        lockReservoir(lock)
        w_sum ← w_sum + w_s
        if rand() < (w_s / w_sum):
            x ← x_s
        unlockReservoir(lock)
```

4.1 Candidate Generation

While directly generating resampling candidates at each texel is possible, not all texels will be visible from the camera and generating candidates for invisible texels is wasteful as their contributions to the loss will be zero. We thus propose to generate candidates by tracing rays from the camera and sharing the same set of candidates at all texels. This approach is equivalent to having the number of candidates equal to the number of pixels (n) times the number of samples per pixel (m). The resulting estimator for parameter π_i is

$$\langle \partial_{\pi_i} \mathcal{L} \rangle = \frac{g_i(x_z)}{q_i(x_z)} \frac{1}{M} \sum_{j=1}^{n} \sum_{k=1}^{m} \frac{q_i(x_{j,k})}{p_j(x_{j,k})}, \tag{14}$$

where we set $M = mn$, a sample $x_{j,k}$ is the kth path sample generated through the jth pixel, and $g_i(x) = \mathrm{w}(x)\partial_{\pi_i} f_c(x)$ is our parameter-space integrand for parameter π_i. While only a few of $x_{j,k}$ will have non-zero contributions to a given texel, the cost of generating the candidates is amortized across all the parameters by sharing the candidates. The above shows an RIS estimator for brevity, but we extend it to PRIS in practice as we discussed in the previous section. Algorithm 2 describes this procedure in pseudocode.

Algorithm 2: Candidate Generation

```
function generateCandidates():
    rs+ ← array of Reservoir[Texture.size()]
    rs− ← array of Reservoir[Texture.size()]

    foreach pixel j in Image:
        for k in 1 to m:                                   ◃ Equation 14
            x_{j,k} ~ p_j # Sample candidate from camera
            foreach texel i where q_i(x_{j,k}) ≠ 0:
                rs+[i].update(x_{j,k} , max(q_i(x_{j,k}), 0) / p_j(x_{j,k}))
                rs−[i].update(x_{j,k} , max(−q_i(x_{j,k}), 0) / p_j(x_{j,k}))

    foreach texel i in Texture:
        # Compute M and W for both positive and negative reservoirs
        foreach r in { rs+[i] , rs−[i] }:
            r.M ← mn                                       ◃ Equation 14
            r.W ← r.w_sum / (q_i(r.x)· r.M)

    return rs+ , rs−
```

Target function. We define our target function q as the signed luminance of our parameter-space integrand g (i.e., RGB values can be negative). Note that setting the target function equivalent to the integrand typically brings no benefit in standard RIS with a single candidate distribution. In this case, RIS reduces to just importance sampling using the candidate distribution p with an unnecessary overhead of resampling. Only the candidate distribution p is relevant to the RIS estimator in this case. In our case, however, selected samples are also used as candidates in the next resampling step, and the candidate distribution p itself also converges to the target q. Our estimator thus still improves over iterations as it converges to a perfect importance sampling estimator of the target p in the limit; Lin et al. [2022] makes the same observation within the forward rendering setting.

Algorithm 3: Reuse Across Iterations

```
1  function reuse(rsCur+ , rsCur− , rsPrev+ , rsPrev−):
2      rs+ ← array of Reservoir[Texture.size()]
3      rs− ← array of Reservoir[Texture.size()]
4
5      foreach texel i in Texture:
6          # Merge current reservoirs and apply MIS
7          foreach r in { rsCur+[i] , rsCur−[i] }:
8              y ← r.x # No shift mapping required
9              m ← m_cur(y) # MIS weight at cur. iteration      ◃ Equation 13
10             rs+[i].update(y , m · max(q_i(y), 0)· r.W)      ◃ Equation 12
11             rs−[i].update(y , m · max(−q_i(y), 0)· r.W)
12
13         # Merge prior reservoirs
14         foreach r in { rsPrev+[i] , rsPrev−[i] }:
15             r.M ← min(r.M , M_cap) # M−capping
16             x ← r.x
17             y ← T_cur(x) # Shift to current iteration
18             m ← m_prev(y) # MIS weight at prev. iteration   ◃ Equation 13
19             J ← |∂T_cur/∂x| # Jacobian of the shift mapping
20             rs+[i].update(y , m · max(q_i(y), 0)· r.W · J)  ◃ Equation 12
21             rs−[i].update(y , m · max(−q_i(y), 0)· r.W · J)
22
23         # Compute M and W for both positive and negative reservoirs
24         rs+[i].M ← rsCur+[i].M + rsPrev+[i].M
25         rs−[i].M ← rsCur−[i].M + rsPrev−[i].M
26         foreach r in { rs+[i] , rs−[i] }:
27             r.W ← r.w_sum / q_i(r.x)                        ◃ Equation 12
28
29     return rs+ , rs−
```

4.2 Sample Reuse

We adapt temporal reuse in ReSTIR by treating different iterations as different frames. We reuse samples from the previous gradient-descent iteration at the current iteration. Algorithm 3 shows pseudocode for this procedure which resamples candidates from the current and previous iterations by merging their reservoirs. Lines 7-11 apply MIS (m_{cur} represents m_s with the current iteration's target function in the numerator, see Equation 13) to the current samples. Since the MIS weights require evaluating the target function at the previous iteration, we not only store the previous set of reservoirs for each texel, but also the texture itself. Lines 14-21 merge the previous iteration reservoirs by applying our PGRIS estimator (Equation 12). We shift each sample to the current iteration

and apply MIS. As discussed in Section 3, we reuse both positive and negative samples from the previous iteration for both reservoirs in the current iteration to handle sign changes. Finally, lines 24-27 compute the new candidate count M, as well as the unbiased contribution weight W. The final derivatives can then be computed using W by applying Equation 12.

To reuse a sample from the previous iteration, we use the random replay shift mapping [Hua et al. 2019; Lin et al. 2022; Manzi et al. 2016] which copies and replays the random numbers used to generate the sample. We chose random replay for its simplicity and generalizability to many scenarios, especially for direct lighting. One disadvantage of random replay is that it generally does not save any computation cost. Despite its computational inefficiency, we still benefit from reuse since the distribution of candidates converges to the target distribution as we discussed above. Other types of shift mapping may further reduce the computation cost. The survey by Hua et al. [2019] lists some common shift mappings.

5 RESULTS AND DISCUSSION

We implemented our method on top of a direct lighting integrator in Mitsuba 3 [Jakob et al. 2022b], using its GPU backend. We found that, at low sample counts, Mitsuba's JIT compiler [Jakob et al. 2022a] spends most of the computation time on *tracing (recording the computation graph) and compiling* the code to GPU kernels, rather than actually *executing* the kernels. While the JIT compiler caches the compilation step when possible, further work is required to cache the tracing step to reduce this overhead. As a result, in order to measure the efficiency of applying ReSTIR, we excluded this overhead, and report the time spent exclusively on executing the GPU rendering kernels. All experiments ran on an NVIDIA GeForce RTX 2080 Ti.

We evaluate our method against Mitsuba 3's base direct lighting integrator across a few scenes with complex direct lighting and materials. We compare both equal-time derivative estimates, as well as equal-time inverse rendering. Each experiment optimizes a single 2048^2 texture encoding parameters of Disney's principled BSDF [Burley 2012], including the base color (albedo), roughness, and anisotropy, from a single view. For all cases, we use 32 spp for the forward primal rendering pass (one "sample" consists of a light sample and a BSDF sample, combined with MIS). For the derivative pass, both methods use detached sampling [Zeltner et al. 2021]. We set M-cap to 32 when optimizing for base color, and 16 otherwise.

To compare equal-time derivatives, we use 1 spp for our method and increase the spp of the derivative pass of Mitsuba 3 so that the time to compute a single iteration is roughly the same. We run our method for 20 iterations, and using the texture at the 20th iteration, compute the baseline and reference derivatives for comparison.

To compare inverse rendering performance, we use the Adam optimizer [Kingma and Ba 2015] with learning rates of 0.1 for the christmas tree scene, 0.005 for the tire scene, and 0.01 for the others. We use 1 spp per iteration for the derivative pass of both methods and run the optimization for a fixed amount of time. The loss function is relMSE.

We also provide additional results in the supplemental material.

Scene: Chalice. Figure 1 (a) involves the recovery of the roughness texture of the chalice with many colored lights. Detached sampling in Mitsuba 3 in this case leads to high variance at estimating the roughness derivative at low roughness, so the insets (b) show that Mitsuba 3 computes noisy gradients. By reusing samples, our method greatly reduces variance of gradient estimates, resulting in (d) faster inverse rendering.

Scenes: Tire and Ashtray. Figure 4 (a) also optimizes for roughness with complex lighting. The right edge of the tire shown in (c) contains very low roughness, resulting in a few yellow specular highlights. The yellow insets in (b) show that Mitsuba 3 at low sample count frequently computes derivatives with the wrong sign—comparing with the reference, some regions that are red should be blue, and vice-versa—resulting in a slow and noisy optimization trajectory in (d). As a result, only our method is able to reconstruct the yellow highlights in the given time. The ashtray scene in Figure 5, which optimizes for the anisotropy parameter, has similar behavior, where incorrect derivative signs slow Mitsuba 3's progress.

Scene: Christmas Tree. Figure 6 (a) optimizes for the base color of the pine needles, uv-mapped to a single texture, lit by lights on the tree. The thin needles with light sources close by create a scene with very challenging visibility. As a result, at low sample count, (b) Mitsuba 3 computes extremely sparse derivatives. While our method at 1 spp is still relatively far from the reference and the reconstructed image (c) looks similar to the one by Mitsuba 3, our derivatives still have over four times lower error than Mitsuba 3's estimates. This reduction in error allows our method to (d) continue reducing loss, after Mitsuba 3 has already converged.

Ablation: Positivization. Figure 7 shows the reconstructed image after inverse rendering from Figure 5 with and without our PGRIS estimator. *With Positivization* uses PGRIS while *Without Positivization* uses GRIS with the target function as the absolute value of the integrand, as discussed in Section 3.2. GRIS without our positivization technique leads to higher variance in gradient estimates, which manifests as additional noise in the final reconstructed rendering.

Discussion: Comparison to Adam. The Adam optimizer [Kingma and Ba 2015] reuses past gradients by keeping exponential moving averages of the gradient and squared gradient, tuned with hyperparameters β_1 and β_2. A core difference between our method and Adam is that we reuse samples to reduce variance at each iteration without bias, effectively increasing sample count. In contrast, Adam simply averages gradients, which reduces noise but cannot reconstruct missing or poorly-sampled regions of the derivatives. As a result, our method still provides significant improvements when used on top of Adam. We experimented with different values of $\beta_1 \in [0.8, 0.95]$ and $\beta_2 \in [0.99, 0.9999]$ and found that our method consistently outperforms the baseline, with less than a 10% difference in the reconstruction errors reported in the paper.

6 CONCLUSION AND FUTURE WORK

We presented a novel adaptation of ReSTIR to differentiable rendering that reuses samples across iterations of gradient descent. Using our parameter-space reformulation of differentiable rendering, we developed a practical resampling algorithm that leverages positivization to achieve theoretical zero-variance convergence of resampled derivative estimates.

Limitations. Our main assumption is that gradients acquired in consecutive iterations are sufficiently correlated. However, this may not hold at high learning rates; in this case, parameters can change by large margins, leading to large gradient differences between iterations. Nevertheless, even in high learning rate settings, our resampling algorithm still empirically accelerates inverse rendering optimization. We also note that the relationship between gradient estimation errors and convergence speed of optimization steps remains generally unclear in differentiable rendering. Indeed, we have observed that inverse rendering can reach adequate minima even with imprecise or noisy gradients.

Future work. Reuse across parameters, analogous to spatial reuse in ReSTIR, is possible with our parameter-space formulation. A potential challenge lies in efficiently selecting neighboring parameters to reuse. Reuse introduces correlation in sample estimates [Sawhney et al. 2022], and the exact effect of correlated gradients in inverse optimization is an interesting avenue to be investigated. While we have focused on differentiable and inverse rendering for BRDF textures under complex direct lighting, our theory and methods are general and can be extended to other rendering methods and scenarios involving general light transport, discontinuities, and other parameters, such as in volumetric or neural representations. Finally, our PGRIS estimator is immediately applicable to Monte Carlo integral estimation in contexts broader than rendering, where integrands can be both positive and negative.

ACKNOWLEDGMENTS

This work was funded in part by NSF grant 2105806, gifts from Adobe and Google, the Ronald L. Graham Chair, the UC San Diego Center for Visual Computing, an NSERC USRA, and a Waterloo Undergraduate Research Fellowship. We thank the anonymous reviewers for their valuable feedback and Yash Belhe for useful discussions during the project.

REFERENCES

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 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.

Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. 2020. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Trans. Graph. (Proc. SIGGRAPH)* 39, 4 (2020), 148.

Brent Burley. 2012. Physically-based shading at Disney. In *SIGGRAPH Course Notes. Practical physically-based shading in film and game production.*, Vol. 2012. ACM, 1–7.

M. T. Chao. 1982. A general purpose unequal probability sampling plan. *Biometrika* 69 (1982), 653–656.

Binh-Son Hua, Adrien Gruson, Victor Petitjean, Matthias Zwicker, Derek Nowrouzezahrai, Elmar Eisemann, and Toshiya Hachisuka. 2019. A Survey on Gradient-Domain Rendering. *Comput. Graph. Forum (Proc. Eurographics STAR)* 38, 2, 455–472.

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

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. 2022a. Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 41, 4 (2022).

James T. Kajiya. 1986. The Rendering Equation. *Comput. Graph. (Proc. SIGGRAPH)* 20, 4 (1986), 143–150.

Markus Kettunen, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. 2015. Gradient-domain Path Tracing. *ACM Trans. Graph. (Proc. SIGGRAPH)* 34, 4 (2015), 123:1–123:13.

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

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.

Daqi Lin, Markus Kettunen, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman. 2022. Generalized Resampled Importance Sampling: Foundations of ReSTIR. *ACM Trans. Graph. (Proc. SIGGRAPH)* 41, 4, Article 75 (2022), 23 pages.

Matthew M. Loper and Michael J. Black. 2014. OpenDR: An Approximate Differentiable Renderer. In *European Conference on Computer Vision*, Vol. 8695. ACM, 154–169.

Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38, 6 (2019), 228.

Marco Manzi, Markus Kettunen, Frédo Durand, Matthias Zwicker, and Jaakko Lehtinen. 2016. Temporal Gradient-Domain Path Tracing. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 35, 6 (2016).

Merlin Nimier-David, Zhao Dong, Wenzel Jakob, and Anton Kaplanyan. 2021. Material and Lighting Reconstruction for Complex Indoor Scenes with Texture-space Differentiable Rendering. In *Eurographics Symposium on Rendering - DL-only Track*, Adrien Bousseau and Morgan McGuire (Eds.).

Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. 2020. Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 39, 4 (2020).

Yaobin Ouyang, Shiqiu Liu, Markus Kettunen, Matt Pharr, and Jacopo Pantaleoni. 2021. ReSTIR GI: Path Resampling for Real-Time Path Tracing. 40, 8 (2021), 17–29.

Art Owen and Yi Zhou. 2000. Safe and effective importance sampling. *J. Amer. Statist. Assoc.* 95, 449 (2000), 135–143.

Art B. Owen. 2013. Monte Carlo theory, methods and examples.

Rohan Sawhney, Daqi Lin, Markus Kettunen, Benedikt Bitterli, Ravi Ramamoorthi, Chris Wyman, and Matt Pharr. 2022. Decorrelating ReSTIR Samplers via MCMC Mutations. *arXiv preprint arXiv:2211.00166* (2022).

Justin F Talbot. 2005. *Importance resampling for global illumination.* Brigham Young University.

Justin F. Talbot, David Cline, and Parris Egbert. 2005. Importance Resampling for Global Illumination. *Rendering Techniques (Proc. EGSR)* (2005), 139–146.

Eric Veach. 1998. *Robust Monte Carlo Methods for Light Transport Simulation.* Ph. D. Dissertation. Stanford University. Advisor(s) Guibas, Leonidas J.

Eric Veach and Leonidas J. Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *SIGGRAPH*. 419–428.

Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021. Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (2021), 108:1–108:14.

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 (2022), 191:1–191:16.

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. 2021. 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, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. 2019. A differential theory of radiative transfer. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38, 6 (2019), 227.

Shuang Zhao, Ioannis Gkioulekas, and Sai Bangaru. 2021. Physics-Based Differentiable Rendering. In *CVPR Tutorial*.

Shuang Zhao, Wenzel Jakob, and Tzu-Mao Li. 2020. Physics-Based Differentiable Rendering: A Comprehensive Introduction. In *SIGGRAPH Courses*. 14:1–14:30.

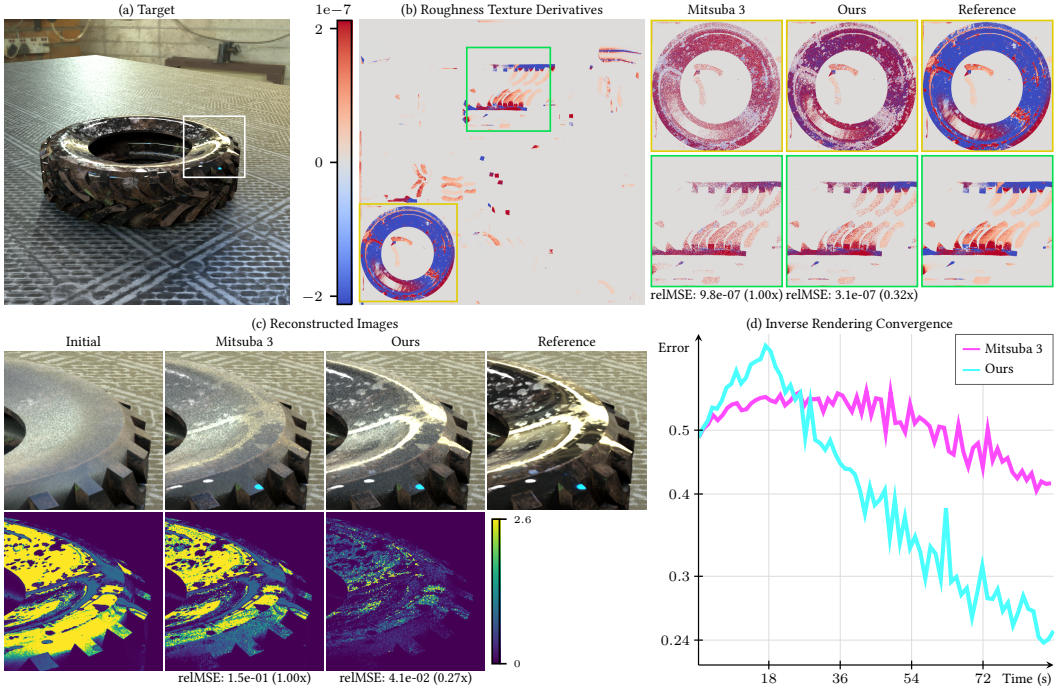

Figure 4: Tire. (a) Inverse rendering of a tire, optimizing for its roughness texture. Mitsuba 3 often computes derivatives (b) with the wrong sign compared to the reference, leading to slow convergence (d), and only our method recovers the glossy yellow highlights (c) by 90 seconds. Scene adapted from Dirty Truck Tire ©HorusZ, Fairy lights ©laha_pictures, and Workshop ©Dimitrios Savva and Jarod Guest.

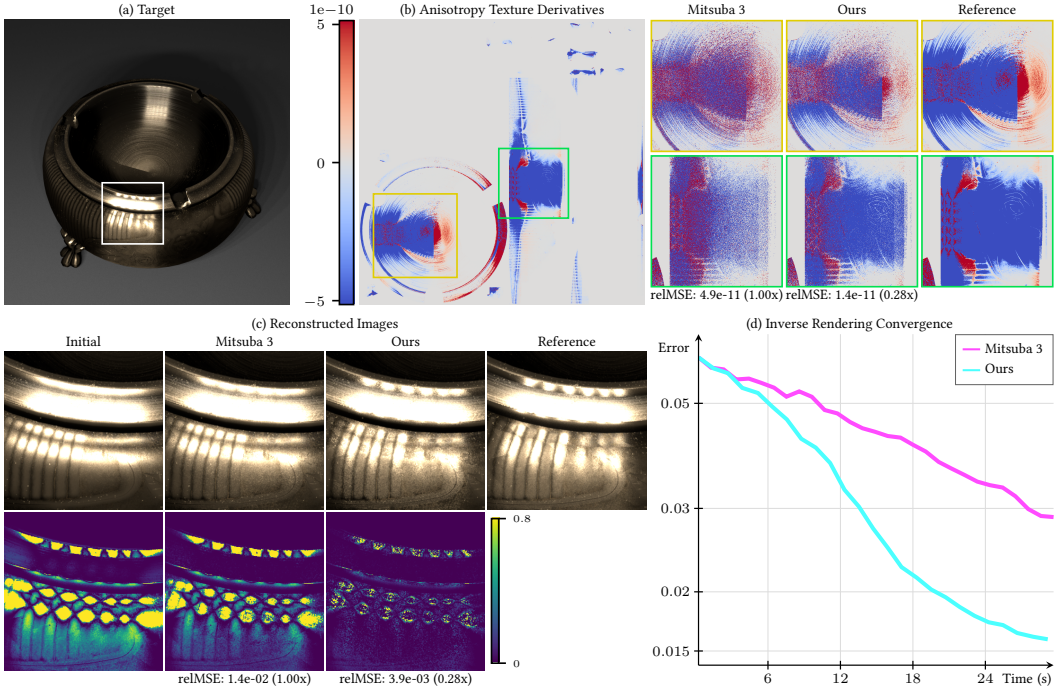

Figure 5: Ashtray. (a) Inverse rendering of an ashtray, optimizing for the anisotropy texture. As in Tire, Mitsuba 3 often computes derivatives (b) with the wrong sign, leading to slower convergence (c) compared to our method. Scene adapted from Vintage Ashtray ©Aartee and Fairy lights ©laha_pictures.

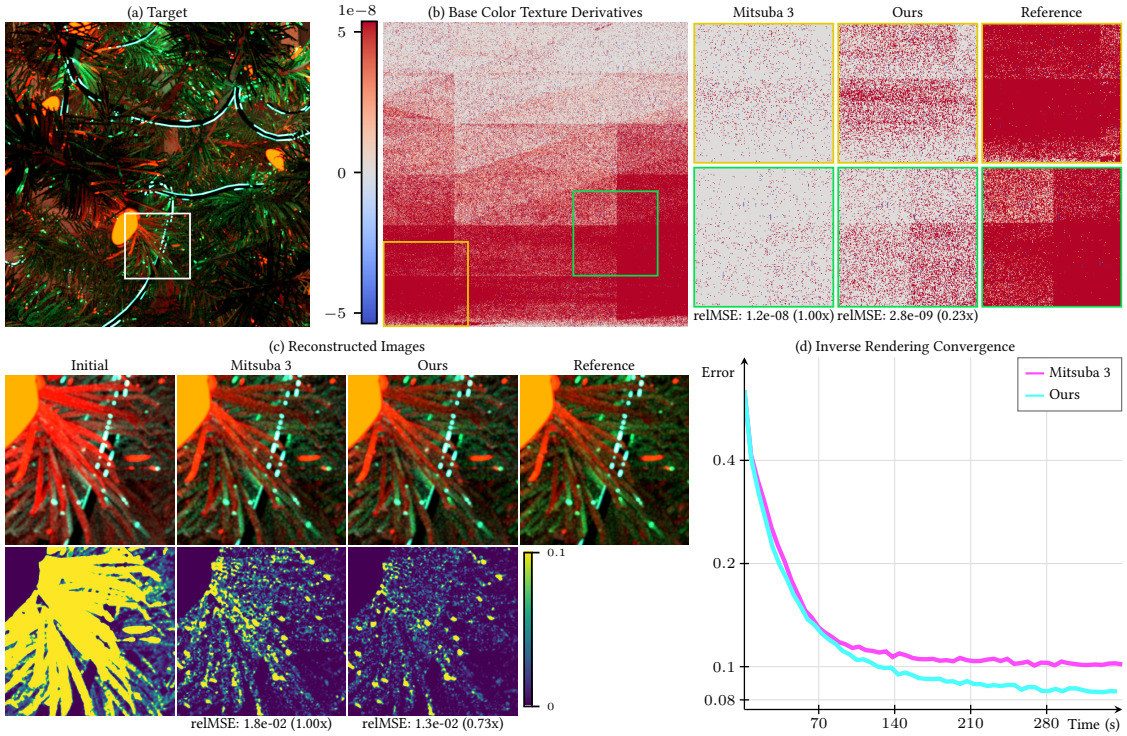

Figure 6: Christmas Tree. (a) Inverse rendering of a Christmas tree, optimizing for the base color of the pine needles. Due to the challenging visibility, Mitsuba 3 computes highly sparse derivatives (b) and therefore only our method (d) continues reducing the image loss after 200 seconds. Scene adapted from Christmas ©Jeremy Birn.

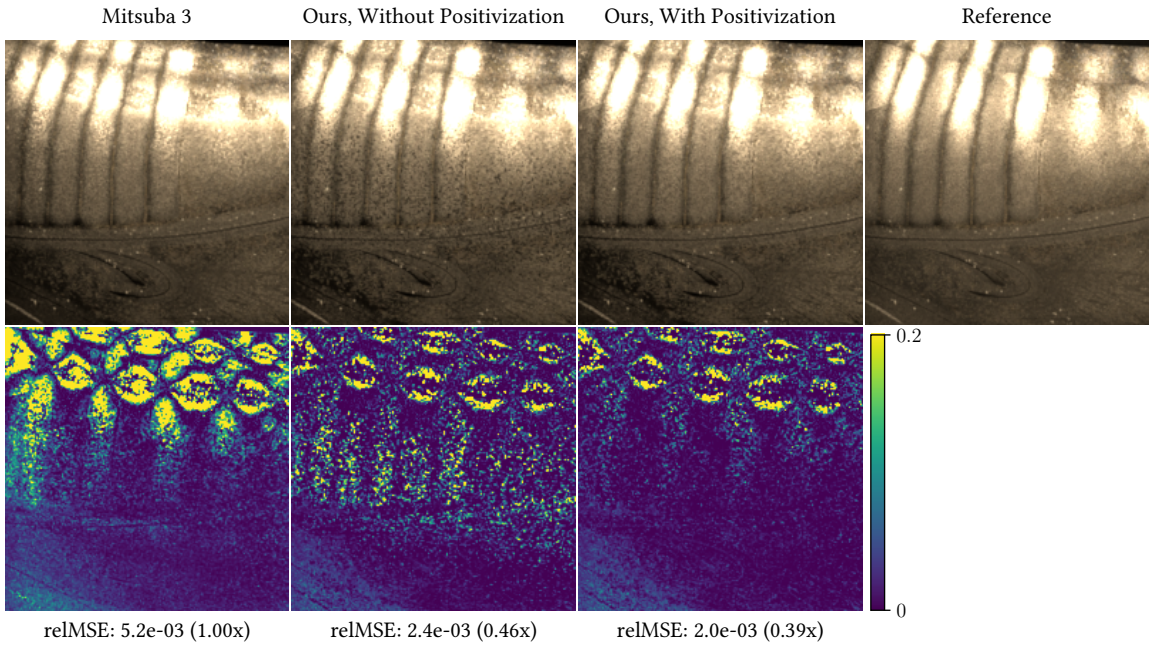

Figure 7: Positivization. Close-up of the ashtray from Figure 5. Not using positivization leads to noisier gradients that manifest as distracting noise in the otherwise smooth surface shown in the final reconstructed image. Scene adapted from Vintage Ashtray ©Aartee and Fairy lights ©laha_pictures.