Practical Stylized Nonlinear Monte Carlo Rendering

XIAOCHUN TONG, University of Waterloo, Canada
TOSHIYA HACHISUKA, University of Waterloo, Canada

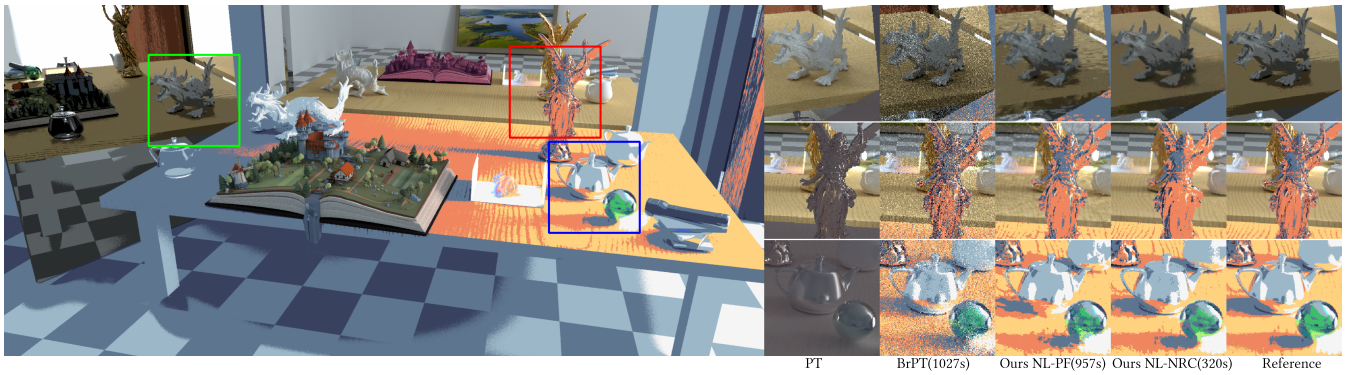

Fig. 1. We propose a practical approach for stylized nonlinear Monte Carlo rendering. Here, we show a modified Veach-Ajar scene with complex nonlinear stylizations applied to each bounce: Outside the mirrors, most objects have a color palette stylization applied at each bounce. Inside the middle mirror, objects retain their physically based appearance except for the book, where a different color palette stylization is applied. Inside the left mirror, every object is applied recursively with a Toon shader. The non-branching path tracer (PT) produces an obviously biased image, where nearly all stylizations are lost. The branching path tracer (BrPT) suffers from high variance, bias (in the form of reduced saturation), and exponential sampling cost. Both of our methods are able to render an image with reasonable accuracy much faster than baseline. The reference is rendered by applying four 32768 SPP final gathering passes to NL-NRC.

The recent formulation of stylized rendering equation (SRE) models stylization by applying nonlinear functions to reflected radiance recursively at each bounce, allowing seamless blend between stylized and physically based light transport. A naive estimator has to branch at each stylized surface, resulting in exponential computation and storage cost. We propose a practical approach for rendering scenes with SRE at a tractable cost. We first propose nonlinear path filtering (NL-PF) that caches the radiance evaluations at intermediate bounces, reducing the exponential sampling cost of the branching estimator of SRE to polynomial. Despite the effectiveness of NL-PF, its high memory cost makes it less scalable. To further improve efficiency, we propose nonlinear radiance caching (NL-NRC) where we apply a compact neural network to store radiance fields. Our NL-NRC has the same linear time sampling cost as a non-branching path tracer and can solve SRE with a high number of bounces and recursive stylization. Our key insight is that, by allowing the network to learn outgoing radiance prior to applying any nonlinear function, the network converges to the correct solution, even when we only have access to biased gradients due to nonlinearity. Our NL-NRC enables rendering scenes with arbitrary, highly nonlinear stylization while achieving significant speedup over branching estimators.

CCS Concepts: • **Computing methodologies** → **Non-photorealistic rendering**; **Ray tracing**.

SIGGRAPH Conference Papers '25, August 10–14, 2025, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1540-2/2025/08
https://doi.org/10.1145/3721238.3730686

Additional Key Words and Phrases: nonlinear Monte Carlo, stylized rendering, nonlinear path filtering, nonlinear neural radiance caching

ACM Reference Format:
Xiaochun Tong and Toshiya Hachisuka. 2025. Practical Stylized Nonlinear Monte Carlo Rendering. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25), August 10–14, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3721238.3730686

1 INTRODUCTION

Monte Carlo (MC) rendering builds upon the fact that the integral equation at each recursion of the rendering equation [Kajiya 1986] is linear, allowing us to expand the recursive integral into a series of high dimensional definite integrals, also known as the path space formulation [Veach 1997]. This formulation allows us to compute the contribution of each light path independently as a *sample* in MC rendering and accumulate them on the fly, yielding an unbiased estimate with constant storage cost and a computation time that scales linearly with the number of samples. However, this expansion no longer holds when the light transport operator becomes *nonlinear*. When an existing MC rendering method, such as path tracing, is used in this case, one generally obtains a biased estimate after applying a nonlinear function to a noisy radiance estimate [Fischer and Ritschel 2024; Rainforth et al. 2018]. This bias decreases as the variance of the radiance estimate decreases. Therefore, to ensure eventual convergence, we need to store all samples to obtain low-variance radiance estimates before applying each nonlinear function. A prominent example of nonlinear light transport is stylized rendering [Doi et al. 2021; West and Mukherjee 2024] where

path contributions will be transformed via an arbitrary, usually nonlinear, stylization function, allowing non-photorealistic effects to be blended seamlessly with physically based rendering. Existing methods either limit stylization application to the first hit only, as in classical stylization [Doi et al. 2021], or require the estimator to branch into an exponential number of light transport paths, resulting in exponential rendering and storage costs [West and Mukherjee 2024].

We propose a practical approach for rendering scenes with nonlinear stylization applied to all bounces. We first propose a nonlinear path filtering (NL-PF) algorithm that efficiently reuses samples to reduce the computation and storage cost from exponential [West and Mukherjee 2024] to polynomial. Secondly, to further improve upon the quadratic computation cost and linear storage scaling of NL-PF, we propose a novel nonlinear radiance caching method (NL-NRC) that reuses samples with a significantly reduced storage cost. NL-NRC further improves the sampling cost from the polynomial time in NL-PF to linear. We also devise a new scheme that allows us to train the neural model even when estimating gradients unbiasedly is challenging due to nonlinearities. Our method can render images of nonlinearly stylized scenes with reasonable accuracy in a few minutes, which would otherwise be impractical with the existing methods.

2 NONLINEAR MONTE CARLO INTEGRATION

Let us first discuss the implications of introducing nonlinearity into MC rendering. The rendering equation [Kajiya 1986] models light transport as a recursive integral equation of a radiance L as

$$L(\mathbf{x},\mathbf{y}) = L_e(\mathbf{x},\mathbf{y}) + \int_A f_s(\mathbf{x},\mathbf{y},\mathbf{z}) G(\mathbf{y},\mathbf{z}) L(\mathbf{y},\mathbf{z}) dz \tag{1}$$

where $L_e(\mathbf{x},\mathbf{y})$ is the radiance emitted from $\mathbf{y}$ to $\mathbf{x}$, f_s is the BSDF, and G is the geometry term. MC rendering in practice works by recursively expanding each $L(\mathbf{y},\mathbf{z})$ as another integral over A. Veach [1997] formalized this expansion as the path integral formulation, which defines the solution to the rendering equation as a definite integral over the product space of As. To derive the path integral formulation, one can first note that the rendering equation is a Fredholm equation of the second kind:

$$f(x) = h(x) + \int_\Omega k(x,y) f(y) dy = h(x) + \mathrm{T}[f] \tag{2}$$

where $h(x)$ and $k(x,y)$ are given functions that are not dependent on unknown $f(x)$, and T is a *linear* operator that defines an integral with $k(x,y)$. Neumann series expansion defines the solution f as

$$f(x) = h(x) + \mathrm{T}[f] \leftrightarrow f(x) = \frac{1}{\mathrm{I}-\mathrm{T}}[h] = \sum_{i=0}^{\infty} \mathrm{T}^{\mathrm{i}}[h] \tag{3}$$

where $\mathrm{I} = \mathrm{T}^0$ is the identity operator, and the solution to the rendering equation is defined as $L = \sum_{i=0}^{\infty} \mathrm{T}^{\mathrm{i}}[L_e]$. Since each T is an integral over the space Ω, we can simplify $\mathrm{T}^{\mathrm{i}}[h]$ as a multiple integral over the product space Ω^i.

$$\begin{aligned}\mathrm{T}^{\mathrm{i}}[h] &= \int_\Omega \left(\cdots \int_\Omega k(x_{i-1},x_i) h(x_i) dx_i \cdots \right) dx_1 \\ &= \int_{\Omega^i} k(x_0,x_1) \cdots k(x_{i-1},x_i) h(x_i) dx_i \cdots dx_1.\end{aligned} \tag{4}$$

This expansion corresponds exactly to the definite integral given by the path integral formulation, and each integral can be estimated using MC integration. For example, taking one sample for each x_i in each inner integral corresponds to the path tracing algorithm.

Let us now consider introducing a *nonlinear* function g that is applied to f, transforming Eq. (2) into a *nonlinear* Fredholm integral equations of the second kind:

$$f(x) = h(x) + \int_\Omega k(x,y) g\left(f(y)\right) dy = h + \mathrm{S}[f] \tag{5}$$

where S also becomes a *nonlinear* operator. While Neumann series expansion might not be applicable in such a case, for certain g that results in a unique solution f, we can still attempt to expand the recursion as repeated applications of S [Wazwaz 2011]:

$$f = \lim_{i\to\infty} h + \underbrace{\mathrm{S}[h + \mathrm{S}[h + \mathrm{S}[h + \cdots]]]}_{\mathrm{S}^i[h]}. \tag{6}$$

With a slight abuse of notation, we used $\mathrm{S}^i[h]$ as a shorthand of repeated applications of S for i times. The nonlinearity in g does not allow us to simplify $\mathrm{S}^{\mathrm{i}}[h]$ any further to a definite integral, and we are left to solve this form

$$\mathrm{S}^{\mathrm{i}}[h] = \int_\Omega k(x_1,x_2) g\left(\cdots \left(\int_\Omega k(x_{i-1},x_i) g\left(h(x_i)\right) dx_i \right) dx_1 \cdots \right). \tag{7}$$

When g is nonlinear, MC integration of each integral with *any* finite number of samples N results in a biased estimate because

$$g\left(\int_\Omega f(x) dx \right) \approx g\left(\frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \right) = g(\mathbf{E}[\langle I_N \rangle]) \tag{8}$$

and due to Jensen's inequality, for arbitrary g, we have

$$g(\mathbf{E}[\langle I_N \rangle]) \neq \mathbf{E}[g(\langle I_N \rangle)], \tag{9}$$

where the equality is only satisfied when g is a linear function. We later demonstrate how a path tracing-like algorithm that takes $N = 1$ samples per integral introduces a significant amount of bias in estimating $\mathrm{S}^{\mathrm{i}}[h]$ to show how problematic it is in practice.

Care has to be taken to ensure that it is not merely a matter of removing bias at each integral. For example, even if g has certain analytic forms such as $1/x$ or e^x, or even if debiasing techniques are applied [Misso et al. 2022] to make $g(\mathbf{E}[\langle I_N \rangle]) = \mathbf{E}[g(\langle I_N \rangle)]$, this does not achieve an unbiased estimate of $\mathrm{S}^{\mathrm{i}}[h]$. Even an unbiased, but *noisy*, estimate of an integral would be subject to multiple applications of g to estimate $\mathrm{S}^{\mathrm{i}}[h]$, ultimately resulting in a biased estimate of $\mathrm{S}^{\mathrm{i}}[h]$. Instead, debiasing needs to be performed on the entire $\mathrm{S}^{\mathrm{i}}[h]$; we later demonstrate that even then, it tends to introduce a significant amount of variance.

While many problems fit this nonlinear recursive integral form (e.g., nonlinear optics, particle physics), the latest example in rendering is the stylized rendering equation (SRE) [West and Mukherjee 2024] that modifies the outgoing radiance using a nonlinear function $g_\theta(\cdot)$ under some parameter θ:

$$L_i(\mathbf{x},\mathbf{y}) = g_\theta\Bigg(\underbrace{L_e(\mathbf{x},\mathbf{y}) + \int_A \overbrace{f_s(\mathbf{x},\mathbf{y},\mathbf{z})G(\mathbf{y},\mathbf{z})}^{C(\mathbf{x},\mathbf{y},\mathbf{z})} L_{i+1}(\mathbf{y},\mathbf{z})d\mathbf{z}}_{U_i(\mathbf{x},\mathbf{y})}\Bigg), \quad (10)$$

where $U_i(\mathbf{x},\mathbf{y})$ represents the pre-stylized outgoing radiance at i-th bounce (or recursion level), emitted from $\mathbf{y}$ to $\mathbf{x}$, $C(\mathbf{x},\mathbf{y},\mathbf{z})$ is a shorthand for the path contribution. Here, θ is a parameter that depends not only on the current vertex $\mathbf{y}$ but also on other information such as the recursive level i and potentially all the vertices in the light path up to $\mathbf{y}$. Due to the presence of the nonlinear function g_θ, path tracing estimation of the pre-stylized radiance $U_i(\mathbf{y},\mathbf{z})$ yields a biased estimate of $L_i(\mathbf{x},\mathbf{y})$. The branching path tracer (BrPT) [West and Mukherjee 2024] uses a multi-sample estimator for the pre-stylized radiance estimate at each recursion level:

$$L_i(\mathbf{x},\mathbf{y}) \approx g_\theta\Bigg(L_e(\mathbf{x},\mathbf{y}) + \sum_{k=1}^{N_i} \frac{C(\mathbf{x},\mathbf{y},\mathbf{z}_k)L_{i+1}(\mathbf{y},\mathbf{z}_k)}{p(\mathbf{z}_k|\mathbf{x},\mathbf{y})}\Bigg), \quad (11)$$

where N_i is the number of *inner* samples at level i. In total, $N = \prod_{i=0}^{D} N_i$ samples are required to estimate the radiance L_0 at the eye vertex, for a maximum recursion level D. Rainforth et al. [2018] showed that Eq. (11) is guaranteed to converge to the correct solution as each N_i approaches infinity.

2.1 Problem Statement

The introduction of nonlinearity in MC rendering poses two significant challenges that make it intractable in practice.

2.1.1 Exponentially costly sampling and slower convergence. Branching path tracing in Eq. (11) requires an exponential number of samples as the recursion level increases, leading to exponential computation and storage costs, since an entire path tree has to be sampled and stored in memory (e.g., stack for recursive calls). Rainforth et al. [2018] have shown that, if g_θ is continuously differentiable, Eq. (11) converges at a rate of $O(\frac{1}{N_0} + (\sum_{i=1}^{D} \frac{1}{N_k})^2)$. If only Lipschitz continuity of g_θ holds, it converges at an even slower rate of $O(\sum_{i=0}^{D} \frac{1}{N_k})$, exponentially slower than linear MC. Even worse, these continuity assumptions may not hold for many stylized functions of interest. For example, the step function used in Toon shading [Barla et al. 2006] is neither differentiable nor Lipschitz continuous. Despite the eventual convergence of Eq. (11), the exact asymptotic convergence rate under an arbitrary g is still an open question.

2.1.2 Convergence with infinite storage. One major difference between linear and nonlinear MC rendering is that, in general, nonlinear estimators require both infinite computation *and* sample storage to achieve consistency, whereas linear estimators require infinite computation but only finite storage (e.g., keeping one sample, accumulating, and discarding). While convergence to the exact solution with arbitrary precision is not required in practice, and consistency is mostly of theoretical interest, the storage cost to achieve a solution with reasonable accuracy can still be intractable.

```
class Vertex:
    x, ω_o     # vertex position, outgoing light direction
    S          # stylization configuration index
               # to receive radiance from
    incoming   # incoming radiance
    outgoing   # outgoing radiance
    level      # recursion level (path depth)
    prev,next  # pointer to previous/next vertex in path

def trace_path(px, ray):
    v, ray = camera.origin, camera.generate_ray(px)
    S = S_0
    path = [v]
    for i in 0..=D:
        v = intersect(ray)
        S = update_stylization_index(S, v)
        v.S = S
        path.push(v)
        ray = sample BSDF
    return path
```

Listing 1. Path vertex data structure and a modified path tracing function used in our methods. Each vertex $\mathbf{x}_i$ stores $\mathcal{S}_{i+1}$ in its $\mathcal{S}$ attribute.

```
def nonlinear_path_filtering():
    # sample one path per pixel
    paths = [trace_path(px) for each pixel]
    for i in D..0:
        for each path p:
            v = i-th vertex in p
            U_i = 0
            # vertices with the same level and
            # stylization configuration index
            for u in K_i(v):
                if u.S == v.S:
                    # Eq. (13)
                    U_i += filter_radiance(v, u)
            v.outgoing = g_{θ,v.prev.S}(U_i)
            # propagate to parent vertices
            v.prev.incoming = v.outgoing
    for each path p:
        splat p[0].outgoing to film
```

Listing 2. Pseudocode of nonlinear path filtering. The vertex data structure and `trace_path` are shown in Listing 1.

3 PRACTICAL ESTIMATOR VIA REUSE AND CACHING

We propose a practical method to address these challenges by extending the idea of path reuse and caching to nonlinear problems.

3.1 Nonlinear Path Filtering (NL-PF)

We have identified that BrPT is inefficient because it samples N_i *independent* inner samples to estimate U_i at each path vertex $\mathbf{x}_i$ at each recursion level. It is possible to amortize this cost by reusing the inner samples across each recursion level. This reuse scheme bears a conceptual similarity to path filtering methods [Keller et al. 2014; West et al. 2020], where each path reuses neighboring paths potentially generated through other pixels; thus, we call our method *nonlinear path filtering* (NL-PF). Listing 2 shows the pseudocode, together with the data structure shown in Listing 1.

Our reuse process starts from the vertices at the highest recursion level and proceeds to the lowest, maintaining and propagating

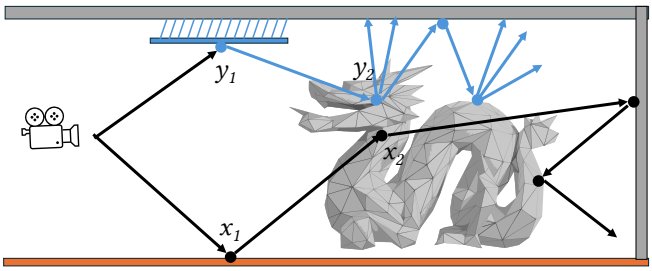

Fig. 2. Scene with multiple stylization configurations. In this example, stylization is enabled only when a path hit the blue mirror (compare $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$).

Fig. 3. A scene with a similar setup as Fig. 2. If we filter all vertices of same level in NL-PF, the right dragon appears pinkish as it incorrectly receives stylized radiance. Filtering vertices of same level and stylization configuration index makes sure that each vertex only receives radiance from desired stylization configuration and makes the rendering consistent with the reference. The reference is rendered by NL-NRC while the others are rendered by NL-PF.

outgoing radiance after stylization. We compute outgoing radiance L_D (which is non-zero only on emitters) and propagate L_D to its predecessor, $\mathbf{x}_{D-1}$. Once all vertices at level D are processed, we move on to level $D-1$. For each vertex $\mathbf{x}_{D-1}$, we find other vertices $\mathbf{y}_{D-1}$ of the same level within a small neighborhood $\mathcal{K}_{D-1}(\mathbf{x}_{D-1})$ and reuse their inner samples to estimate U_{D-1} for $\mathbf{x}_{D-1}$ by connecting $\mathbf{x}_{D-1}$ to the path suffix (in this case, just $\mathbf{y}_D$). When all vertices at level $D-1$ are processed, we move on to level $D-2$, repeating this process down to the camera vertex at level $D=0$.

Unlike path filtering, not all neighboring vertices can be grouped and filtered together in nonlinear setting. Fig. 2 illustrates this constraint. In this scene, once a ray hits the blue mirror, all subsequent levels will have stylization applied ($\overline{\mathbf{y}}$ path). If a ray does not hit the mirror, no stylization should be applied ($\overline{\mathbf{x}}$ path). In this case, the stylization parameter θ depends not only on the current vertex but also on the entire trajectory of a path [West and Mukherjee 2024]. Filtering $\mathbf{x}_2$ and $\mathbf{y}_2$ mixes paths with different stylizations.

To properly handle such cases in NL-PF, we explicitly parameterize the outgoing radiance by an additional parameter, $\mathcal{S}$, which we refer to as a *stylization configuration index*. This index, $\mathcal{S}$, is an integer value that identifies each stylization configuration. The stylization index enables multiple *stylization configurations* in the scene, where each stylization configuration is a mapping between objects and stylization functions. Thus, each object could have multiple stylization functions, selectively enabled by $\mathcal{S}$. Fig. 2, for example, has two stylization configurations: null stylization (i.e., no stylization) and a nonlinear stylization enabled only when a path hits the blue mirror. As a simple extension, it is also possible to have uncountably many stylization configurations (e.g., parametric stylization configuration depending on location) in a scene, but we limit ourselves to countably many configurations for simplicity.

The introduction of the stylization index leads to a slight modification to the notation of the stylized rendering equation:

$$L_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathcal{S}_i) = \\ g_{\theta,\mathcal{S}_i}\left(L_e(\mathbf{x}_{i-1}, \mathbf{x}_i) + \int_A C(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}) L_{i+1}(\mathbf{x}_i, \mathbf{x}_{i+1}, \mathcal{S}_{i+1}) d\mathbf{x}_{i+1}\right), \tag{12}$$

Each vertex $\mathbf{x}_i$ is associated with two stylization configuration indices, where $\mathcal{S}_i$ determines the stylization parameter θ, and $\mathcal{S}_{i+1}$ determines the stylization configuration from which $\mathbf{y}$ receives radiance. In Listing 1, we only store $\mathcal{S}_{i+1}$ for each vertex $\mathbf{x}_i$, since $\mathcal{S}_{i+1}$ is defined even if $\mathbf{x}_{i+1}$ does not exist (e.g., a ray misses the scene). The index $\mathcal{S}_{i+1}$ depends on $\mathcal{S}_i$, the current vertex $\mathbf{x}_i$, and the path prefix. The update rule from $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$ is controlled by the material/stylization assigned at $\mathbf{x}_i$. For example, in Fig. 2 and Fig. 3, let $\mathcal{S}_{null}$ and $\mathcal{S}_{sty}$ be the two stylization configuration indices. The mirror sets $\mathcal{S}_{i+1}$ to $\mathcal{S}_{sty}$ regardless of $\mathcal{S}_i$, while the other surfaces set $\mathcal{S}_{i+1} = \mathcal{S}_i$. This leads to the following NL-PF estimator:

$$U'_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathcal{S}_{i+1}) = L_e(\mathbf{x}_{i-1}, \mathbf{x}_i) + \\ \frac{1}{|\mathcal{N}|} \sum_{\mathbf{y}_i \in \mathcal{N}} \frac{C(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{y}_{i+1}) L'_{i+1}(\mathbf{x}_i, \mathbf{y}_{i+1}, \mathcal{S}_{i+1})}{p(\mathbf{y}_{i+1})} \\ L'_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathcal{S}_i) = g_{\theta,\mathcal{S}_i}(U'_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathcal{S}_{i+1})), \tag{13}$$

where $\mathcal{N} = |\mathcal{K}_i(\mathbf{x}_i)|$, $\mathcal{K}_i$ is the filter kernel of the i-th level, and U'_i and L'_i denote the filtered pre-stylized and stylized radiance at $\mathbf{x}_i$. The reciprocal probability $1/p(\mathbf{y}_{i+1})$ can be exact (if tractable) or approximated using MIS or another weighting scheme.

3.1.1 Implementation. A naive implementation of NL-PF incurs an $O(N^3)$ runtime cost and an $O(N^2)$ storage cost for N vertices due to the following reasons. The number of vertices within the filter kernel grows linearly with N, so each vertex $\mathbf{x}_i$ receives $O(N)$ incoming radiance estimates. Evaluating the incoming radiance from the successor $\mathbf{y}_{i+1}$ of one of its neighbors, $\mathbf{y}_i$, requires re-evaluating the BSDF for $O(N)$ incoming directions at $\mathbf{y}_{i+1}$. We need to perform this computation for all $O(N)$ outgoing directions, leading to a total $O(N^3)$ computation cost and $O(N^2)$ storage cost to track incoming radiance at all vertices from all other vertices.

While this polynomial scaling is an improvement over the exponential computation and storage cost of BrPT, the $O(N^3)$ scaling is still less than ideal. Some optimizations found in path filtering techniques (e.g., CMIS weights with clustering [Deng et al. 2021; West et al. 2022]) can improve this scaling to an $O(N^2)$ computation cost and an $O(N)$ storage cost. We also reuse the BSDF evaluations by keeping the incoming direction at $\mathbf{y}_i$ instead of computing new incoming directions between $\mathbf{x}_i$ and $\mathbf{y}_{i+1}$. Visibility tests between reused vertices are also omitted as an approximation.

To fit as many samples as possible within a limited amount of GPU memory, we implemented NL-PF in an out-of-core (OOC) style:

Fig. 4. Directly learning stylized radiance L_i causes NL-NRC to converge to an incorrect solution due to biased and inconsistent gradient estimates. Learning pre-stylized radiance U_i allows the network to converge to the correct solution as the bias in the gradient estimates diminishes.

```
def train_nl_nrc(model):
    # sample one path per pixel for a
    # fraction of all the pixels in the image
    paths = [trace_path(px) for a fraction of pixels]
    for each vertex in each path:
        i = v.depth
        ω_i = v.next.ω_o
        L_{i+1} = g_{θ,v.S}(M_π(i+1, v.S, v.next.x, ω_i, F))
        # Eq. (16)
        U_i = L_e + L_{i+1} · |ω_i · n| * bsdf(v) / v.next.pdf
        L = ||M_π(i, v.S, v.x, v.ω_o, F) - detach(U_i)||^2
        optimize π to minimize L
```

Listing 3. Overview of the training process of our NL-NRC model.

During the initial path tracing phase, we keep only the last vertex of each active path in GPU memory, while others are off-loaded to CPU memory. During path filtering, we keep only the vertices currently being filtered in GPU memory. Despite these optimizations, while its computation is faster than BrPT, NL-PF is still memory-intensive, as we will show later.

3.1.2 Differences from Regular Path Filtering. Let us highlight two key differences between our nonlinear path filtering and existing path filtering methods. The first difference is that regular path filtering [Keller et al. 2014; West et al. 2020] assumes each path within the path space can be decomposed into a prefix of finite length and a suffix of infinite length. This assumption allows regular path filtering to filter paths of different lengths simultaneously, even in a fixed-point iteration manner [Deng et al. 2021]. In our nonlinear extension, filtering should occur in an ordered manner, starting from the highest levels to the lowest, while filtering only vertices with the same level and stylization configuration index to correctly support flexible level-dependent and path-prefix-dependent stylization.

The second difference is the convergence behavior. Path filtering works well with a relatively low number of samples, and averaging *multiple runs* with a shrinking filtering radius can lead to convergence to the unbiased solution. Because of the presence of nonlinearity, our path filtering converges only when the number of samples within a *single* run approaches infinity. Not only averaging over multiple runs does not reduce bias, but also a shrinking radius makes convergence even less likely because each run will have fewer samples.

3.2 Nonlinear Neural Radiance Caching (NL-NRC)

Even after employing various optimizations and approximations, NL-PF has an impractical storage cost. NL-PF is also less effective than regular PF because paths can only be filtered when they share the same stylization index. We propose an alternative approach, *nonlinear neural radiance caching* (NL-NRC), which extends the neural radiance cache to enable radiance estimate reuse in nonlinear MC rendering. This approach achieves similar estimation accuracy to NL-PF with significantly less storage cost. Listing 3 presents the pseudocode for our NL-NRC method.

Because caching is typically performed in the space of positions and directions in 3D space, we first rewrite the SRE to be parameterized by the surface point $\mathbf{x}$ and view direction ω_o:

$$L_i(\mathbf{x}_i, \omega_o, \mathcal{S}_i) = g_{\theta,\mathcal{S}_i}\left(L_e(\mathbf{x}_i, \omega_o) + \int_\Omega f_s(\mathbf{x}_i, \omega_o, \omega_i) L_{i+1}(\mathbf{x}_{i+1}, \omega_i, \mathcal{S}_{i+1}) d\omega_i^\perp\right), \quad (14)$$

where ω_i represents the incoming ray direction from $\mathbf{x}_{i+1}$ to $\mathbf{x}_i$. Our model, $\mathcal{M}_\pi(i, \mathcal{S}, \mathbf{x}, \omega_o, \mathcal{F})$, takes the query position $\mathbf{x}$, view direction ω_o, path depth i, and stylization configuration index $\mathcal{S}$ as input, along with auxiliary features $\mathcal{F}$ such as albedo, shading normal, and roughness. Here, π represents the model parameters. The core idea is to have the model $\mathcal{M}$ learn the outgoing radiance at surface point $\mathbf{x}$ and view direction ω_o. It might be tempting to have the model learn the outgoing radiance directly by minimizing the following loss:

$$\mathcal{L}_\pi(i, \mathcal{S}_i, \mathbf{x}_i, \omega_o) = ||\mathcal{M}_\pi(i, \mathcal{S}_i, \mathbf{x}_i, \omega_o, \mathcal{F}) - L_i(\mathbf{x}_i, \omega_o, \mathcal{S}_i)||^2 \quad (15)$$

This approach faces an immediate problem: we do not have an unbiased estimator for the gradient since we have only a biased estimator for $L_i(\mathbf{x}, \omega_o, \mathcal{S}_i)$. Unfortunately, stochastic gradient descent is not guaranteed to converge to the correct local minima with biased gradient estimates [Ajalloeian and Stich 2021]. Unless we increase the sample count while computing the loss, the optimization generally does not converge to the correct solution, even if the network has unlimited capacity. Fig. 4 empirically confirms this problem.

We propose a solution to this problem that enables network convergence even when only biased gradient estimates are available. Instead of directly learning the outgoing radiance, we decouple the model to learn the pre-stylization radiance U_i, which can be estimated unbiasedly if L_{i+1} is unbiased. When computing the loss, we query the model $\mathcal{M}$ to obtain an approximation of L_{i+1};

$$U_i'(\mathbf{x}_i, \omega_o, \mathcal{S}_{i+1}) = L_e(\mathbf{x}_i, \omega_o) + \int_\Omega f_s(\mathbf{x}_i, \omega_o, \omega_i) g_{\theta,\mathcal{S}_{i+1}}(\mathcal{M}_\pi(i+1, \mathcal{S}_{i+1}, \mathbf{x}_{i+1}, \omega_i, \mathcal{F})) d\omega_i^\perp,$$
$$\text{and when } i = D, \quad U_i'(\mathbf{x}_i, \omega_o, \mathcal{S}_{i+1}) = L_e(\mathbf{x}_i, \omega_o)$$
$$\mathcal{L}_\pi(i, \mathcal{S}_{i+1}, \mathbf{x}_i, \omega_o) = ||\mathcal{M}_\pi(i, \mathcal{S}_{i+1}, \mathbf{x}_i, \omega_o, \mathcal{F}) - \mathit{detach}(U_i'(\mathbf{x}_i, \omega_o, \mathcal{S}_{i+1}))||^2, \quad (16)$$

where $\mathit{detach}(x)$ means that we do not backpropagate the gradient to x. In other words, we are decoupling nonlinear stylization from the data to be encoded by the network, while still coupling them via the loss function.

3.2.1 Asymptotically Unbiased Gradient Estimation via Propagation. The idea of the above approach is that we can compute asymptotically unbiased gradients as training progresses by propagating the lower-bias estimates from higher recursion levels to lower ones. At the beginning of training, one can estimate the loss unbiasedly only at the highest depth D based on the emission term. As training progresses and $\mathcal{M}_\pi(D,\cdot)$ approximates U_D, the loss estimate at the previous depth $D-1$ will exhibit a decreasing bias. If the network has sufficient capacity and learns U_D exactly, this approach should enable us to estimate $\mathcal{L}_\pi(D-1,\cdot)$ unbiasedly, allowing $\mathcal{M}_\pi(D-1,\cdot)$ to gradually converge to U_{D-1}. Subsequently, if $\mathcal{M}_\pi(D-1,\cdot)$ perfectly approximates U_{D-1}, we will receive an unbiased gradient estimate at level $D-2$, and the model will converge to U_{D-2}, propagating the unbiased estimate further down the levels and eventually reaching U_0. When training lower recursion levels, we treat the model output from higher levels as constant to prevent errors from lower levels from influencing the more accurate high levels.

While this process implies that we need to train NL-NRC starting from the highest levels to the lowest, as in NL-PF, we have empirically found that it is also possible to optimize the model for all levels simultaneously. We conjecture that this is related to the fact that SGD also converges with *consistent* gradients [Chen and Luss 2019]. Training all levels simultaneously also makes training progressive and allows us to take advantage of the correlation of radiance between levels to accelerate convergence. We defer a rigorous analysis of the convergence properties of NL-NRC to future work.

3.2.2 Training Details. For each iteration, we generate 16 batches of training records, where each batch consists of 65536 pairs of training data. We found that a large batch size is essential for faster convergence, presumably due to the high bias in gradient estimates during the initial phase. We disable Russian roulette when tracing training paths to reduce the variance. The initial learning rate is chosen between $0.001 \sim 0.002$ and is reduced using an exponential decay scheduler with a factor of 0.995 every 50 iterations. The training automatically terminates when the learning rate becomes sufficiently small. We also apply an EMA with a decay weight of 0.95 when visualizing the cache [Müller et al. 2021].

3.2.3 Final Gathering. While our NL-NRC can generally learn the pre-stylized radiance effectively, it may still produce minor artifacts if visualized directly. We can employ a final gathering step to remove those remaining artifacts. We trace one path per pixel until the first non-specular vertex is found, where we sample additional rays starting from that vertex and query the NL-NRC at ray hits. Path splitting may be required if smooth dielectric objects are encountered. The additional computational cost of this step is manageable. For example, our final gathering runs at a speed around 100 SPP/sec on a 1024^2 image, and a rendering with a final gathering pass of 32768 SPP would take approximately 350 sec.

4 DISCUSSION

4.1 Assumptions

Our methods are built upon two mild assumptions to make the problem more tractable. These assumptions are generally satisfied in most cases, but we explain them here for completeness. Most path tracer implementations use next-event estimation (NEE) to estimate the emission term. In nonlinear rendering, the outgoing radiance depends on the sum of the emission term and *reflected* indirect radiance in a nonlinear manner, thus both of them should be always evaluated to compute the outgoing radiance. A common implementation of NEE evaluates only the emission term and therefore leads to a minor bias in nonlinear rendering. We thus assume that emissive surfaces are either non-reflective, or their reflected radiance is of negligible magnitude compared to their emission. The stylization function g_θ generally contains discontinuities. We assume that there is no surface patch whose pre-stylized outgoing radiance lies exactly at the discontinuities of g_θ, because the stylized radiance becomes either undefined or extremely difficult to approximate. This assumption is satisfied for most stylizations, such as Toon or Cel shading [Barla et al. 2006]. Their discontinuities usually occur only along curves on the surface.

4.2 Reusing and Caching outside Rendering

Reusing and caching have been applied to many MC problems besides rendering, such as grid-free PDE solvers [Bakbouk and Peers 2023; Li et al. 2023] and MC fluid simulation [Rioux-Lavoie et al. 2022; Sugimoto et al. 2024]. Outside computer graphics, regression is a widely used method. Longstaff and Schwartz [2001] used least-squares regression for option pricing, and Broadie et al. [2015] used basis functions to approximate the inner-level integral for risk estimation. These methods mainly target nonlinear problems with only a few recursion levels. Feng and Li [2022] applied sample reuse in nonlinear problems in finance. Their estimator uses resampling to share samples between different simulation paths. Our work focuses on solving highly nested nonlinear MC integration problems with arbitrary nonlinear functions in the context of stylized rendering. In particular, pre-stylized radiance learning in NL-NRC utilizes domain-specific knowledge of stylized rendering to provide a practical algorithm.

4.3 Other Potential Approaches

4.3.1 Debasing and Multilevel Monte Carlo. Limited forms of nonlinear problems have appeared in computer graphics in the context of volumetric rendering [Georgiev et al. 2019; Kettunen et al. 2021], reciprocal estimation [Jhang and Chang 2022; Qin et al. 2015; Zeltner et al. 2020], and differentiable rendering [Bangaru et al. 2020]. Power series debiasing [Blanchet and Glynn 2015; Dauchet et al. 2018; Georgiev et al. 2019; Kettunen et al. 2021; Lee et al. 2019; Misso et al. 2022] expands a smooth nonlinear function using a series. By stochastically evaluating a prefix sum of the series, one can estimate the recursive nonlinear integral unbiasedly. However, not all stylization functions can be trivially expressed or approximated using a series. We instead aim to solve nonlinear light transport with *arbitrary* stylization. Telescoping series debiasing [Misso et al. 2022; Tong and Hachisuka 2024; West and Mukherjee 2024] and multi-level Monte Carlo (MLMC) [Giles 2008; Heinrich 2001] are other popular methods for transforming a biased yet consistent estimator into an unbiased one. The performance of telescoping series debiasing heavily depends on the convergence rate of the estimator sequence. We found that this method usually produces a

significant amount of variance (*e.g.,* Fig. 11), especially with discontinuous stylization functions. Note also that this method still suffers from the same exponential sampling cost as BrPT, as we need at least two independent samples to estimate the telescoping series, resulting again in branching at each recursion. Instead of unbiasedly estimating stylization at the cost of increased variance, we aim for a low-variance and low-bias solution to the stylized rendering equation that is consistent as storage size increases.

4.3.2 Fixed-point Iterations. Outside computer graphics, Beck et al. [2020] proposed a polynomial time solver based on fixed point iterations. Given this work, it might be tempting to solve Eq. (10) using fixed-point iteration as well. For example, neural radiosity [Hadadan et al. 2021; Su et al. 2024] essentially solves the rendering equation through a fixed-point iteration where the radiance field is stored in a neural network. Path graphs [Deng et al. 2021] also employ fixed-point iteration to refine those estimates. We, however, identified one major issue in employing fixed-point iterations in our problem: level-dependent stylization. Fixed-point iterations for integral equations are based on the assumption that the nonlinear function is independent of the recursion level. As West and Mukherjee [2024] pointed out, it is crucial to admit level-dependent stylization. As explained already, our work does not have any issue handling such stylization through the use of stylization configuration indices.

5 RESULTS

We implemented our method in a CPU/GPU rendering framework using the Rust frontend [Tong et al. 2023] of LuisaCompute [Zheng et al. 2022]. All results were measured on a workstation equipped with an Intel i9-13900KF 24C/32T CPU, 128GB of DDR5 RAM, and an NVIDIA RTX 4070 Ti SUPER GPU with 16GB VRAM. The baseline method is a wavefront-style branching path tracer (BrPT) that runs on a GPU and samples a full path tree at once, computing vertices of the same depth in parallel. Our NL-PF implementation utilizes CMIS weights with range-based clustering [West et al. 2022]. The filtering kernel size is set to be between 0.05% and 0.3% of the scene's bounding box. We implemented our NL-NRC using the fully fused MLPs of the tiny-cuda-nn [Müller 2021] framework. The position input is encoded using a multi-resolution hash grid encoding [Müller et al. 2022], and the view direction is encoded using one-blob encoding [Müller et al. 2019]. Our NL-NRC model has 16 million parameters in fp16 precision and requires 32MB of memory, with the majority of the memory usage attributed to the hash grid encoding. We employed an MLP with 5 hidden layers of width 128 for all scenes to achieve a balance between inference/training cost and reconstruction quality. Fig. 10 presents a brief analysis of the impact of network size.

5.1 Validation

To verify the correctness of our method, we compared it against brute-force BrPT on scenes with a recursive continuous stylization applied to all surfaces. The continuous stylization also allows us to render a low-bias BrPT reference solution in reasonable time. For this experiment, we disabled the pixel filter because it adds another dimension of integration that significantly extends the rendering time for BrPT references. It is important to note that all the methods compared here support pixel filters in practice.

First, we tested our method against BrPT on a scene with two-bounce continuous stylization, as shown in Fig. 5. We show independent runs of BrPT and NL-PF at increasing sample counts, and an NL-NRC training sequence with increasing computation time. The stylization used in this scene is the saturation adjustment function described in West and Mukherjee [2024]. Both our NL-PF and NL-NRC are able to converge to a solution similar to BrPT as computation time increases. We report the memory consumption for NL-PF, excluding the memory used for scene data. NL-PF quickly runs out of memory after one minute of rendering, preventing further improvements, whereas NL-NRC continues to improve with more samples.

We then compare our methods on the same scene, now with 12 bounces of recursive stylization, as shown in Fig. 6. Despite the high recursion level, a low-bias BrPT reference solution is still possible because of the relatively simple lighting in this scene. The reference BrPT image uses 1024, 32, 8, and 8 branches at the first four bounces, with no branching at higher levels. In this case, BrPT converges more slowly than in the two-bounce setting because of the increased sampling cost. Our NL-PF converges faster than BrPT but incurs a significant memory cost, and fails to produce a converged image before running out of memory. In contrast, our NL-NRC quickly produces a noise-free solution, and an additional final gathering pass can further improve image quality.

5.2 More Challenging Lighting and Stylization

We further evaluated our methods on scenes with a high number of recursion levels and discontinuous stylization in Fig. 7, Fig. 8, and Fig. 9. The discontinuous stylization we used maps radiance values to a predefined color palette: for a given pre-stylized radiance estimate, we replace it with the closest color in the palette while rescaling the value to approximately maintain its luminance. The user can also boost indirect lighting by a custom factor. In these scenes, it is essentially intractable for BrPT to render a reference solution. In fact, there is no readily available option for rendering an accurate solution in this scene. As a result, we do not report error metrics in these figures. In Fig. 7 and Fig. 8, even after hours of rendering, BrPT still produces visible bias and noise compared to our NL-NRC, which produces cleaned-up images orders of magnitude faster.

We also compared NL-PF and NL-NRC in challenging scenes in Fig. 1, Fig. 3, and Fig. 9, with "reference images" rendered by NL-NRC with 32768 SPP final gathering. Fig. 1 and Fig. 3 demonstrate that our NL-PF is consistent with NL-NRC on scenes with multiple stylization configurations and glossy light transport. Fig. 9 highlights another advantage of NL-NRC over NL-PF. While both methods solve nonlinear rendering faster than the BrPT baseline, NL-PF produces some visible bias because the bias introduced by optimizations during path filtering is amplified by the discontinuous stylization. Our NL-NRC is free from such issues. For example, the top-right wall is lit up in both the NL-NRC and BrPT solutions. Fig. 1 features a complex scene with three stylization configurations, detailed geometry, specular reflections, and both continuous

(the right dragon) and discontinuous recursive stylization (other objects). Both our NL-PF and NL-NRC produce similar solutions that are more accurate and faster than their baselines.

6 CONCLUSION AND FUTURE WORK

We present a practical approach for stylized nonlinear MC rendering. Our NL-PF reduces sampling and storage costs from exponential to polynomial, offering a predictable method for solving SRE that is guaranteed to converge as the sample count increases. Our NL-NRC further reduces the memory cost, resulting in a practical method. We believe NL-NRC is currently the only viable and practical method for handling arbitrary stylizations in nonlinear MC rendering.

We focused primarily on stylized rendering, but nested nonlinear integral equations also arise in other problems. For example, MC fluid solvers [Sugimoto et al. 2024] require nested integration due to their nonlinear advection step. Similarly, nonlinear optics [Bloembergen 1996] allows objects to scatter radiance nonlinearly depending on light field strength. Using NL-NRC to accelerate these problems could be a fruitful avenue for future research. Beyond the stylization demonstrated in this work, a vast range of potential nonlinear stylizations exists. Given our ability to render scenes with multiple-bounce, recursive stylization within a few minutes, we hope our work will inspire further efforts in designing novel nonlinear stylizations or extending NL-NRC to support even more interesting effects.

ACKNOWLEDGMENTS

We would like to thank all anonymous reviewers for their constructive feedback; Weijie Zhou, Wenyou Wang, and Shaokun Zheng for providing feedback on paper drafts. The book model in Fig. 1 is shared by Pixel from Sketchfab under CC-BY-4.0. The flashlight model in Fig. 1 is shared by Brandon Baldwin from Sketchfab under CC-BY-4.0. This research was funded by NSERC Discovery Grants (RGPIN-2020-03918).

REFERENCES

Ahmad Ajalloeian and Sebastian U. Stich. 2021. On the Convergence of SGD with Biased Gradients. arXiv:2008.00051 [cs.LG] https://arxiv.org/abs/2008.00051

Ghada Bakbouk and Pieter Peers. 2023. Mean Value Caching for Walk on Spheres. In *Eurographics Symposium on Rendering*, Tobias Ritschel and Andrea Weidlich (Eds.). The Eurographics Association. https://doi.org/10.2312/sr.20231120

Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased warped-area sampling for differentiable rendering. *ACM Trans. Graph.* 39, 6 (2020), 245:1–245:18. https://doi.org/10.1145/3414685.3417833

Pascal Barla, Joëlle Thollot, and Lee Markosian. 2006. X-toon: an extended toon shader. In *4th International Symposium on Non-Photorealistic Animation and Rendering, NPAR 2006, Annecy, France, June 5-7, 2006, Proceedings*, Douglas DeCarlo and Lee Markosian (Eds.). ACM, 127–132. https://doi.org/10.1145/1124728.1124749

Christian Beck, Arnulf Jentzen, and Thomas Kruse. 2020. Nonlinear Monte Carlo methods with polynomial runtime for high-dimensional iterated nested expectations. arXiv:2009.13989 [math.PR] https://arxiv.org/abs/2009.13989

Jose H. Blanchet and Peter W. Glynn. 2015. Unbiased Monte Carlo for optimization and functions of expectations via multi-level randomization. In *2015 Winter Simulation Conference (WSC)*. 3656–3667. https://doi.org/10.1109/WSC.2015.7408524

N. Bloembergen. 2002 - 1996. *Nonlinear optics* (4th ed. ed.). World Scientific, Singapore.

Mark Broadie, Yiping Du, and Ciamac C. Moallemi. 2015. Risk Estimation via Regression. *Operations Research* 63, 5 (2015), 1077–1097. http://www.jstor.org/stable/24540434

Jie Chen and Ronny Luss. 2019. Stochastic Gradient Descent with Biased but Consistent Gradient Estimators. arXiv:1807.11880 [cs.LG] https://arxiv.org/abs/1807.11880

Jérémi Dauchet, Jean-Jacques Bezian, Stéphane Blanco, Cyril Caliot, Julien Charon, Christophe Coustet, Mouna El Hafi, Vincent Eymet, Olivier Farges, Vincent Forest, Richard Fournier, Mathieu Galtier, Jacques Gautrais, Anaïs Khuong, Lionel Pelissier, Benjamin Piaud, Maxime Roger, Guillaume Terrée, and Sebastian Weitz. 2018. Addressing nonlinearities in Monte Carlo. *Scientific Reports* 8, 1 (05 Sep 2018), 13302. https://doi.org/10.1038/s41598-018-31574-4

Xi Deng, Milos Hasan, Nathan Carr, Zexiang Xu, and Steve Marschner. 2021. Path graphs: iterative path space filtering. *ACM Trans. Graph.* 40, 6 (2021), 276:1–276:15. https://doi.org/10.1145/3478513.3480547

Kohei Doi, Yuki Morimoto, and Reiji Tsuruno. 2021. Global Illumination-Aware Stylised Shading. *Comput. Graph. Forum* 40, 7 (2021), 11–20. https://doi.org/10.1111/CGF.14397

Runhuan Feng and Peng Li. 2022. Sample recycling method – a new approach to efficient nested Monte Carlo simulations. *Insurance: Mathematics and Economics* 105 (2022), 336–359. https://doi.org/10.1016/j.insmatheco.2022.04.012

Michael Fischer and Tobias Ritschel. 2024. ZeroGrads: Learning Local Surrogates for Non-Differentiable Graphics. *ACM Trans. Graph.* 43, 4 (2024), 49:1–49:15. https://doi.org/10.1145/3658173

Iliyan Georgiev, Zackary Misso, Toshiya Hachisuka, Derek Nowrouzezahrai, Jaroslav Křivánek, and Wojciech Jarosz. 2019. Integral formulations of volumetric transmittance. *ACM Trans. Graph.* 38, 6 (2019), 154:1–154:17. https://doi.org/10.1145/3355089.3356559

Michael B. Giles. 2008. Multilevel Monte Carlo Path Simulation. *Oper. Res.* 56, 3 (2008), 607–617. https://doi.org/10.1287/OPRE.1070.0496

Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. 2021. Neural radiosity. *ACM Trans. Graph.* 40, 6 (2021), 236:1–236:11. https://doi.org/10.1145/3478513.3480569

Stefan Heinrich. 2001. Multilevel Monte Carlo Methods. In *Large-Scale Scientific Computing, Third International Conference, LSSC 2001, Sozopol, Bulgaria, June 6-10, 2001, Revised Papers (Lecture Notes in Computer Science, Vol. 2179)*, Svetozar Margenov, Jerzy Wasniewski, and Plamen Y. Yalamov (Eds.). Springer, 58–67. https://doi.org/10.1007/3-540-45346-6_5

Jia-Wun Jhang and Chun-Fa Chang. 2022. Specular Manifold Bisection Sampling for Caustics Rendering. *Comput. Graph. Forum* 41, 7 (2022), 247–254. https://doi.org/10.1111/CGF.14673

James T. Kajiya. 1986. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986, Dallas, Texas, USA, August 18-22, 1986*, David C. Evans and Russell J. Athay (Eds.). ACM, 143–150. https://doi.org/10.1145/15922.15902

Alexander Keller, Ken Dahm, and Nikolaus Binder. 2014. Path Space Filtering. In *Monte Carlo and Quasi-Monte Carlo Methods, MCQMC 2014, Leuven, Belgium, April 2014 (Springer Proceedings in Mathematics and Statistics, Vol. 163)*, Ronald Cools and Dirk Nuyens (Eds.). Springer, 423–436. https://doi.org/10.1007/978-3-319-33507-0_21

Markus Kettunen, Eugene d'Eon, Jacopo Pantaleoni, and Jan Novák. 2021. An unbiased ray-marching transmittance estimator. *ACM Trans. Graph.* 40, 4 (2021), 137:1–137:20. https://doi.org/10.1145/3450626.3459937

A. Lee, S. Tiberi, and G. Zanella. 2019. Unbiased approximations of products of expectations. *Biometrika* 106, 3 (2019), pp. 708–715. https://www.jstor.org/stable/48637485

Zilu Li, Guandao Yang, Xi Deng, Christopher De Sa, Bharath Hariharan, and Steve Marschner. 2023. Neural Caches for Monte Carlo Partial Differential Equation Solvers. In *SIGGRAPH Asia 2023 Conference Papers, SA 2023, Sydney, NSW, Australia, December 12-15, 2023*, June Kim, Ming C. Lin, and Bernd Bickel (Eds.). ACM, 34:1–34:10. https://doi.org/10.1145/3610548.3618141

Francis Longstaff and Eduardo Schwartz. 2001. Valuing American Options by Simulation: A Simple Least-Squares Approach. *Review of Financial Studies* 14 (02 2001), 113–47. https://doi.org/10.1093/rfs/14.1.113

Zackary Misso, Benedikt Bitterli, Iliyan Georgiev, and Wojciech Jarosz. 2022. Unbiased and consistent rendering using biased estimators. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 41, 4 (July 2022). https://doi.org/10/gqjn66

Thomas Müller. 2021. *tiny-cuda-nn*. https://github.com/NVlabs/tiny-cuda-nn

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.* 41, 4 (2022), 102:1–102:15. https://doi.org/10.1145/3528223.3530127

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. *ACM Trans. Graph.* 38, 5 (2019), 145:1–145:19. https://doi.org/10.1145/3341156

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.* 40, 4 (2021), 36:1–36:16. https://doi.org/10.1145/3450626.3459812

Hao Qin, Xin Sun, Qiming Hou, Baining Guo, and Kun Zhou. 2015. Unbiased photon gathering for light transport simulation. *ACM Trans. Graph.* 34, 6 (2015), 208:1–208:14. https://doi.org/10.1145/2816795.2818119

Tom Rainforth, Robert Cornish, Hongseok Yang, and Andrew Warrington. 2018. On Nesting Monte Carlo Estimators. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 4264–4273. http://proceedings.mlr.press/v80/rainforth18a.html

Damien Rioux-Lavoie, Ryusuke Sugimoto, Tümay Özdemir, Naoharu H. Shimada, Christopher Batty, Derek Nowrouzezahrai, and Toshiya Hachisuka. 2022. A Monte

Fig. 5. We test BrPT, NL-PF, and NL-NRC on a scene with two-level recursive continuous stylization to verify that both our methods converge to a similar solution as BrPT. Our NL-NRC converges faster than BrPT and continues to improve as training time increases, thanks to its constant memory cost.

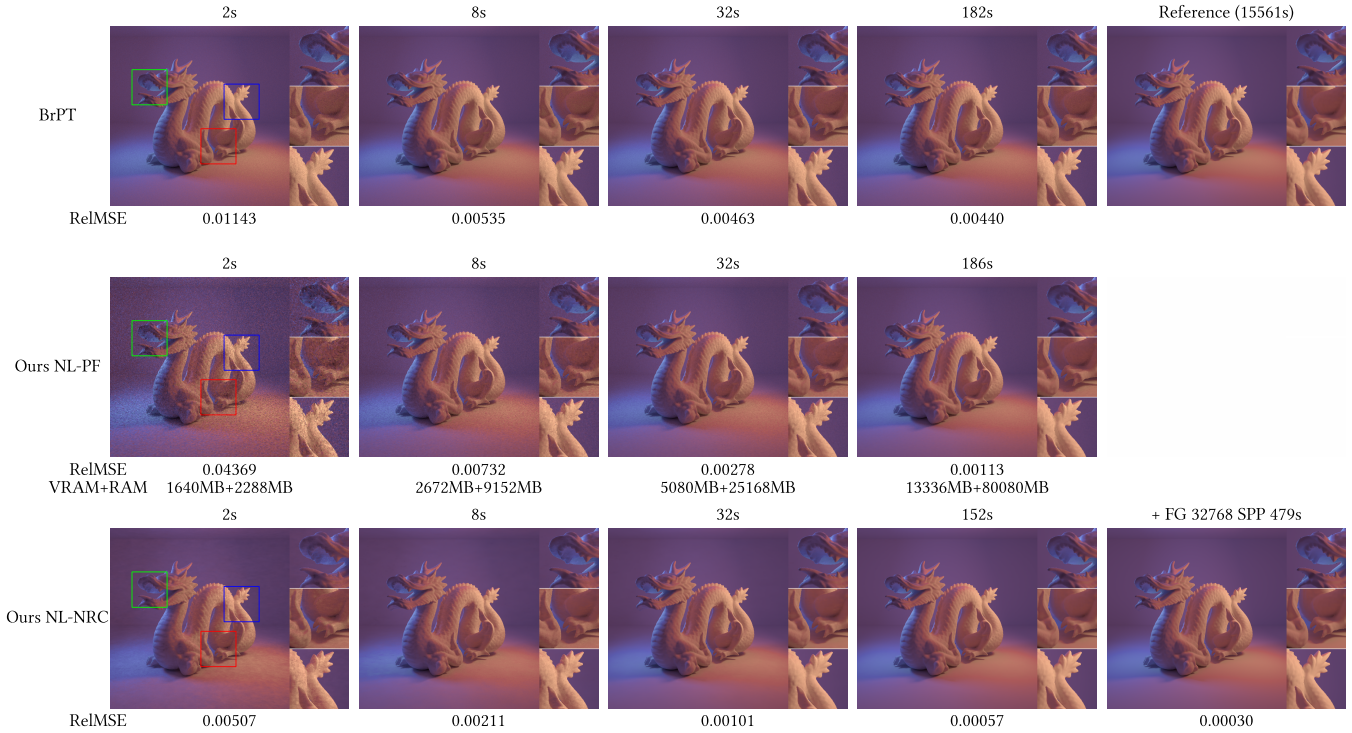

Fig. 6. A scene with 12-level recursive continuous stylization is used to verify that both our methods converge to a similar solution as BrPT. The reference solution by BrPT has relatively low bias due to low variance light transport in this scene. Both our NL-PF and NL-NRC converge faster than BrPT. NL-NRC training finishes at 152s with 3000 iterations. We can further improve its rendering quality by applying a post-training final gathering pass.

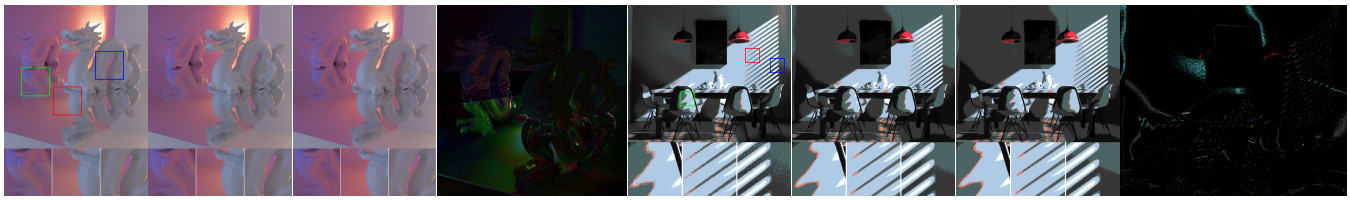

Fig. 7. On more challenging scenes, our NL-NRC produces reasonable results with low variance in a few minutes, while BrPT takes a long time to compute and still produces visible bias. Left: A scene with 12-bounce recursive continuous stylization and two stylization configurations. Even after 5 hours of rendering, the BrPT solution still contains some visible bias (in the form of reduced saturation). Right: A scene with 2-bounce recursive *discontinuous* stylization. The BrPT rendering still produces some noisy edges even after a very long rendering session. We use a 32768 SPP final gathering pass for NL-NRC in both scenes. The BrPT error image is computed as the relative L2 color difference between the NL-NRC image and the BrPT image with longer rendering time. The right halves of the images have their exposure adjusted for better visualization.

Fig. 8. A scene with 12-bounce recursive discontinuous stylization. BrPT produces highly visible noise and bias after hours of rendering, while NL-NRC produces a sharp image in a few minutes.

Fig. 9. We compare NL-PF and NL-NRC the same scene as Fig. 8. We run NL-PF with as many samples as physically possible. While NL-PF produces an image that is, in many places, similar to the NL-NRC solution, visible bias in the image can still be observed because the discontinuous stylization amplifies the bias introduced during path filtering. For example, the wall at top-right appears too dark.

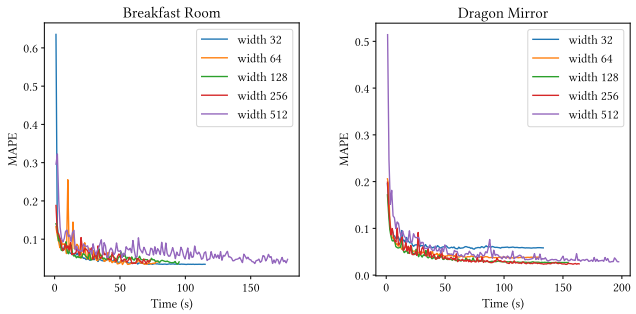

Fig. 10. We show the impact of the width of hidden layer in NL-NRC. For scenes with mostly diffuse illumination (left), even small networks, such as those 32 units wide, produce good results. On scenes with specular materials (right), a larger network is required for an accurate approximation.

Fig. 11. Non-branching path tracer produces obviously biased result. Applying telescoping series debiasing removes the bias in the limit, but it is prone to having a significant amount of variance, especially for discontinuous stylization. Exposure=+1.0 for better visualization.

Carlo Method for Fluid Simulation. *ACM Trans. Graph.* 41, 6 (2022), 240:1–240:16. https://doi.org/10.1145/3550454.3555450

Rui Su, Honghao Dong, Jierui Ren, Haojie Jin, Yisong Chen, Guoping Wang, and Sheng Li. 2024. Dynamic Neural Radiosity with Multi-grid Decomposition. In *SIGGRAPH Asia 2024 Conference Papers, SA 2024, Tokyo, Japan, December 3-6, 2024*, Takeo Igarashi, Ariel Shamir, and Hao (Richard) Zhang (Eds.). ACM, 23:1–23:12. https://doi.org/10.1145/3680528.3687685

Ryusuke Sugimoto, Christopher Batty, and Toshiya Hachisuka. 2024. Velocity-Based Monte Carlo Fluids. In *ACM SIGGRAPH 2024 Conference Papers, SIGGRAPH 2024, Denver, CO, USA, 27 July 2024- 1 August 2024*, Andres Burbano, Denis Zorin, and Wojciech Jarosz (Eds.). ACM, 8. https://doi.org/10.1145/3641519.3657405

Xiaochun Tong and Toshiya Hachisuka. 2024. Efficient Image-Space Shape Splatting for Monte Carlo Rendering. *ACM Trans. Graph.* 43, 6 (2024), 233:1–233:11. https://doi.org/10.1145/3687943

Xiaochun Tong, Hsueh-Ti Derek Liu, Yotam I. Gingold, and Alec Jacobson. 2023. Differentiable Heightfield Path Tracing with Accelerated Discontinuities. In *ACM SIGGRAPH 2023 Conference Proceedings, SIGGRAPH 2023, Los Angeles, CA, USA, August 6-10, 2023*, Erik Brunvand, Alla Sheffer, and Michael Wimmer (Eds.). ACM, 19:1–19:9. https://doi.org/10.1145/3588432.3591530

Eric Veach. 1997. *Robust Monte Carlo methods for light transport simulation.* Ph. D. Dissertation. Stanford University, USA. https://searchworks.stanford.edu/view/3911108

Abdul-Majid Wazwaz. 2011. *Linear and Nonlinear Integral Equations: Methods and Applications* (1st ed.). Springer Publishing Company, Incorporated.

Rex West, Iliyan Georgiev, Adrien Gruson, and Toshiya Hachisuka. 2020. Continuous multiple importance sampling. *ACM Trans. Graph.* 39, 4 (2020), 136. https://doi.org/10.1145/3386569.3392436

Rex West, Iliyan Georgiev, and Toshiya Hachisuka. 2022. Marginal Multiple Importance Sampling. In *SIGGRAPH Asia 2022 Conference Papers, SA 2022, Daegu, Republic of Korea, December 6-9, 2022*, Soon Ki Jung, Jehee Lee, and Adam W. Bargteil (Eds.). ACM, 42:1–42:8. https://doi.org/10.1145/3550469.3555388

Rex West and Sayan Mukherjee. 2024. Stylized Rendering as a Function of Expectation. *ACM Trans. Graph.* 43, 4 (2024), 96:1–96:19. https://doi.org/10.1145/3658161

Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. 2020. Specular manifold sampling for rendering high-frequency caustics and glints. *ACM Trans. Graph.* 39, 4 (2020), 149. https://doi.org/10.1145/3386569.3392408

Shaokun Zheng, Zhiqian Zhou, Xin Chen, Difei Yan, Chuyan Zhang, Yuefeng Geng, Yan Gu, and Kun Xu. 2022. LuisaRender: A High-Performance Rendering Framework with Layered and Unified Interfaces on Stream Architectures. *ACM Trans. Graph.* 41, 6 (2022), 232:1–232:19. https://doi.org/10.1145/3550454.3555463