High-Performance Graphics 2021
N. Binder and T. Ritschel
(Guest Editors)

Volume 40 (2021), Number 8

ReSTIR GI: Path Resampling for Real-Time Path Tracing

Y. Ouyang[1], S. Liu[1], M. Kettunen[1], M. Pharr[1], J. Pantaleoni[1]

[1]NVIDIA Corporation, Santa Clara, CA, USA

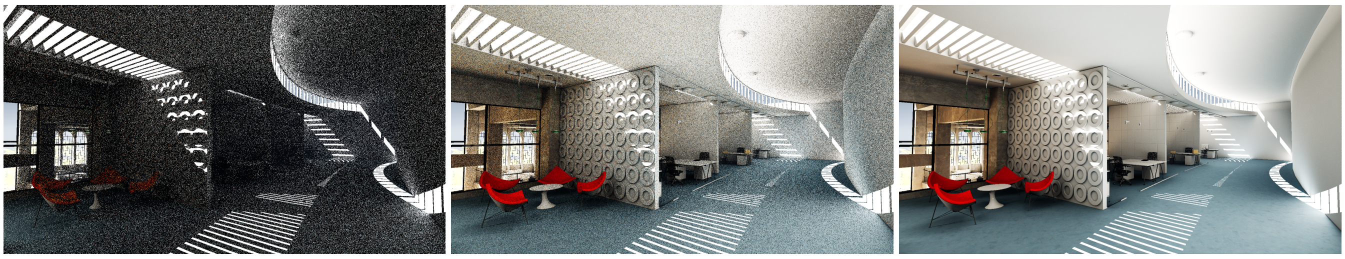

Figure 1: *Equal-time comparison of two-bounce path tracing with our approach. Images are rendered at 1080p resolution with an NVIDIA 3090 RTX GPU without denoising. (Left) Path tracing with one sample per pixel in 8.0 ms. (Middle) ReSTIR GI using spatial and temporal resampling and one sample per pixel in 8.9 ms. Mean squared error is improved by a factor of* 15.1. *(Right) Path traced reference image. This is a challenging scene for path tracing, as direct lighting is concentrated in small regions, making it difficult to find indirect lighting paths. ReSTIR GI is much more effective thanks to sample reuse in both space and time.*

Abstract

*Even with the advent of hardware-accelerated ray tracing in modern GPUs, only a small number of rays can be traced at each pixel in real-time applications. This presents a significant challenge for path tracing, even when augmented with state-of-the art denoising algorithms. While the recently-developed ReSTIR algorithm [BWP*20] enables high-quality renderings of scenes with millions of light sources using just a few shadow rays at each pixel, there remains a need for effective algorithms to sample indirect illumination.*

We introduce an effective path sampling algorithm for indirect lighting that is suitable to highly parallel GPU architectures. Building on the screen-space spatio-temporal resampling principles of ReSTIR, our approach resamples multi-bounce indirect lighting paths obtained by path tracing. Doing so allows sharing information about important paths that contribute to lighting both across time and pixels in the image. The resulting algorithm achieves a substantial error reduction compared to path tracing: at a single sample per pixel every frame, our algorithm achieves MSE improvements ranging from $9.3\times$ *to* $166\times$ *in our test scenes. In conjunction with a denoiser, it leads to high-quality path traced global illumination at real-time frame rates on modern GPUs.*

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ ***Rendering; Ray tracing;***

1. Introduction

The flexibility and generality offered by path tracing [Kaj86] is highly desirable for real-time rendering, offering the promise of a single unified algorithm that renders photorealistic imagery of scenes with complex lighting, materials, and geometry. However, path tracing has long seemed out of reach for real-time applications due to its substantial computational requirements; even with the advent of hardware-accelerated ray tracing [McC13, NVI18], at best a few tens of rays can currently be traced at each pixel at real-time framerates. While high quality denoising algorithms (e.g. Scheid et al.'s SVGF algorithm [SKW*17, SPD18] or neural approaches such as Munkberg and Hasselgren's [MH20]) can improve the appearance of noisy imagery, it is still important to sample rays effectively so they provide as much information as possible about the scene lighting.

Our work focuses on maximizing the quality of path traced images with multi-bounce global illumination (GI), before denoising. We aim to improve the efficiency of sampling indirect lighting to make path traced global illumination possible in real-time. To do so, we employ the combination of resampled importance sampling (RIS) [TCE05] and reservoir resampling [Vit85, Cha82], that was introduced in the form of the ReSTIR algorithm by Bitterli et al. for sampling direct illumination [BWP*20].

In contrast to ReSTIR, which places initial samples in a global light space, our algorithm places initial samples in the space of the local sphere of directions around shading points. Tracing the corresponding rays gives points on surfaces in the scene; the amount of light that they scatter back toward the ray origin determines their RIS weights. Resampling these points both in space and time allows us to generate weighted samples from a distribution approximating the indirect illumination in the scene, leading to substantial error reduction.

In our test scenes, we see improvements from 9.3$\times$ to 166$\times$ in mean squared error (MSE) compared to path tracing thanks to the massive resampling possible due to reservoirs. Because the variance of an unbiased Monte Carlo estimator decreases linearly with sample count, these results imply that path tracing requires 9.3 to 166 times more paths to achieve the same MSE.

One notable property of our approach is that all data structures required for storing reservoirs and samples are simple screen-space buffers that are independent of the spatial extent of the scene. Unlike for example, path guiding algorithms, which generally require maintaining complex world-space data structures, our approach uses a fixed amount of memory and is easily updated in parallel; each pixel only modifies its own reservoir, and it is not difficult to ensure that no reservoirs are being modified when reservoirs at nearby pixels are accessed. Thus, performance is high in a GPU implementation; our ReSTIR GI implementation in *Unreal Engine 4* adds from 8 ms to 18 ms per frame when rendering at 1080p resolution with an NVIDIA 3090 RTX GPU.

2. Previous Work

There is a wide plethora of existing techniques targeting real-time simulation of indirect diffuse global illumination (e.g. [MMSM21, HKL16]). Since most of them are heavily biased in a not so easily quantifiable manner, while ours can be made either unbiased or very low bias, here we will focus only on the techniques which are most closely related.

Building on the resampled importance sampling technique developed by Talbot et al. [TCE05], Bitterli et al. presented the ReSTIR algorithm, which applies screen-space and temporal resampling to light source samples for direct illumination [BWP*20]. Their approach maintains a small reservoir of one or more light samples at each pixel and then applies reservoir sampling [Vit85, Cha82] to generate samples from a distribution that approximates the product of BSDF, light source, and a binary visibility term. They showed both biased and unbiased variants of their algorithm, both of which gave substantial reductions in error compared to previous state-of-the-art light sampling algorithms, thanks to sample reuse and sharing information among pixels.

Indirect lighting poses more challenges than direct lighting, as the integration domain is intrinsically much more complex and higher-dimensional. In standard path tracing, a widely used approach is to importance sample directions at each vertex along a path, according to a distribution that well matches the local BSDF. Many such BSDF sampling algorithms have been developed; see Pharr et al. [PJH16] for more information. While BSDF sampling works well if the indirect light is slowly varying, it is not effective in the presence of strong, off-peak indirect illumination.

In the presence of non-uniform indirect lighting, error can be significantly reduced by using *path guiding* algorithms that attempt to sample according to either the incident indirect lighting, or the product of the BSDF and the indirect lighting. Early work in this area includes that of Lafortune and Willems [LW95], who built a 5D spatio-directional tree based on a path tracing preprocess, and Jensen's use of photon maps [Jen96] for path guiding [Jen95], which Hey and Purgathofer made a number of improvements to [HP02]. These approaches are two-pass methods, in which a preprocess builds a data structure that is then used during rendering. While these data structures are read-only during rendering, and thus suitable for highly-parallel architectures like GPUs, their parallel construction is more time-consuming and they deliver inferior error reduction compared to more recent approaches.

More recently, Vorba et al. [VKŠ*14] and Herholz et al. [HEV*16] applied Gaussian mixture models to learn a representation of incident illumination during rendering. Müller and colleagues developed a widely-used approach based on an adaptive spatial decomposition of the scene into an octree and an adaptive directional decomposition based on a quadtree [MGN17, VHH*19]. This approach has been extended to account for the product of illumination and the BSDF by Diolatzis et al. [DGJ*20]. Ruppert et al. further accounted for the effect of parallax within cells of the spatial decomposition [RHL20] and Deng et al. applied path guiding to rendering participating media [DWWH20]. While these approaches can provide substantial error reduction, constructing these structures in parallel with thousands of threads on a GPU incurs a significant amount of overhead, that does not seem suitable for real-time applications [Pan20]. Dittebrandt et al. recently presented cheaper and more scalable methods for path guiding [DHD20].

Deep learning has also been applied to path guiding, including work by Müller et al. [MMR*19], Zheng and Zwicker [ZZ19], and Bako et al. [BMDS19]. These approaches demonstrated substantial reduction in error due to more effective path sampling, though their performance is insufficient for real-time applications, both for training and inference.

Other forms of online learning have also been applied to path guiding, including Dahm and Keller's use of reinforcement learning [DK16]. Pantaleoni built on this work to show online learning can successfully learn high-dimensional control variates and radiance caches in real-time using spatio-directional hashing [Pan20].

Instead of directly guiding paths, our work reuses sample paths across time (frames) and space (pixels). Early forms of path reuse in global illumination are based on Virtual Point Lights (VPLs) [Kel97, DKH*14]. These methods sample light subpaths from the sources and reuse their vertices as virtual light sources to illuminate the points visible from the camera. This approach was later extended by employing more sophisticated many-light methods [HPB07] and full bidirectional path tracing variants [PRDD15, CBH*18, TH19].

A more general, seminal approach based on reusing subpaths starting from the camera was described by Bekaert et al. [BSH02]. Our work draws many ideas from this approach, employing virtually the same vertex reconnection strategy, while adapting the ReSTIR reservoir resampling and merging algorithms to select and weight reused subpaths without using complex auxiliary data structures. The result is a much more flexible algorithm: whereas Bekaert et al.'s original algorithm required an $O(M^2)$ multiple importance sampling (MIS) evaluation in the number M of reused samples, that could only be reduced to $O(M)$ by using a fixed tiled reuse pattern (amortizing some of the costs shared among all pixels), our ReSTIR based algorithm requires $O(M)$ computations for arbitrary reuse patterns, and allows temporal as well as spatial reuse.

More recently, Bauszat et al. have improved the efficiency of path reuse by applying ideas from gradient domain rendering [BPE17] and West et al. [WGGH20] have shown that Continuous MIS can be applied to to reduce the bias of path space filtering algorithms [KDB14]. Unlike West et al.'s algorithm, ours employs a more general spatio-temporal reuse, can be made fully unbiased, and explicitly targets real-time rendering and GPU acceleration.

3. Background

The fundamental problem in real-time rendering is to solve the rendering equation [Kaj86], which gives the outgoing radiance at a point x in direction ω_o for a purely reflective surface as

$$L(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega L_i(x,\omega_i)\, f(\omega_o,\omega_i)\langle\cos\theta_i\rangle\, d\omega_i, \quad (1)$$

where L is the outgoing radiance, L_e is the emitted radiance, Ω is the hemisphere of directions around the surface normal, L_i is the incoming radiance, f is the BSDF, $\langle\cos\theta_i\rangle$ is the cosine of the angle between the direction ω_i and the surface normal with negative values clamped to zero, and $d\omega$ is the solid angle measure.

Under the assumption that there is no participating media, the incident radiance L_i can be written in terms of the outgoing radiance at the first visible surface along a ray from x in the direction ω_i:

$$L_i(x,\omega_i) = L_o(\text{TRACE}(x,\omega_i), -\omega_i), \quad (2)$$

where the TRACE function returns the point on the closest surface from x in direction ω_i. The traditional Monte Carlo method uses the estimator

$$\hat{L} = L_e(x,\omega_o) + \frac{1}{N}\sum_{j=1}^{N} \frac{L_i(x,\omega_j)\, f(\omega_o,\omega_j)\cos\omega_j}{p(\omega_j)}, \quad (3)$$

where N independent samples are taken and $p(\omega_j)$ is the probability density function (PDF) from which the samples were drawn. As long as $p(\omega) > 0$ whenever the integrand is non-zero, the estimator gives an unbiased estimate of the integral. (See e.g. Pharr et al. [PJH16] for more information about the Monte Carlo method and its application to rendering.)

The more closely that the PDF p matches the integrand, the lower the error in the Monte Carlo estimator. Resampled importance sampling [TCE05] is an effective approach for sampling from complex distributions that cannot be sampled directly. It uses a two-pass algorithm to generate samples. First, M candidate samples $\mathbf{y} = y_1, \ldots, y_M$ are sampled from a source distribution $p(y)$. Then a *target PDF* $\hat{p}$ is used to resample one sample z from $\mathbf{y}$ with probability

$$p(z|\mathbf{y}) = \frac{w(z)}{\sum_{j=1}^{M} w(y_j)}, \quad (4)$$

where

$$w(y) = \frac{\hat{p}(y)}{p(y)} \quad (5)$$

is the sample's relative weight. As M increases, the distribution of samples z more closely matches $\hat{p}$. For the purposes of resampling, $\hat{p}$ can be replaced with an unnormalized target function that is only proportional to the target PDF. In the following, we will take advantage of this and will also use $\hat{p}$ to denote target functions.

Given such a z resampled from $\mathbf{y}$, then as long as $\hat{p} > 0$ where the integrand is non-zero, an unbiased estimate of the integral $\int f(x)\,dx$ is given by the RIS estimator:

$$\hat{L} = \frac{f(z)}{\hat{p}(z)} \frac{1}{M} \sum_{j=1}^{M} \frac{\hat{p}(y_j)}{p(y_j)}. \quad (6)$$

If the target PDF is a better match to the integrand than p then RIS reduces error.

As shown by Bitterli et al. [BWP*20], weighted reservoir sampling (WRS) [Vit85, Cha82] leads to an efficient GPU implementation of RIS. For reference, the WRS algorithm is shown in Algorithm 1, including both a function to update the reservoir with a single sample and to merge another reservoir, yielding a sample drawn from the candidate samples considered by both reservoirs. Following Bitterli et al., our reservoir also stores a weight W for the sample z that is stored in the reservoir and is given by

$$W(z) = \frac{1}{\hat{p}(z)M} \sum_{j=1}^{M} \frac{\hat{p}(y_j)}{p(y_j)}. \quad (7)$$

Thus, the RIS estimator is easily evaluated by

$$\hat{L} = f(z)W(z). \quad (8)$$

4. ReSTIR GI

The original ReSTIR algorithm [BWP*20] places initial samples using light sampling where the source PDF $p(x)$ samples uniformly on the surfaces of lights that are themselves sampled according to their emitted power. The target function $\hat{p}(x)$ is then given by the unshadowed reflected radiance due to the light sample, which is given by the product of emitted radiance, the BSDF, and the geometric coupling term.

To apply ReSTIR to sample indirect illumination, we must represent directions that contribute to indirect illumination. Because this representation must support both spatial and temporal reuse at different points in space, unit vectors on the local hemisphere of

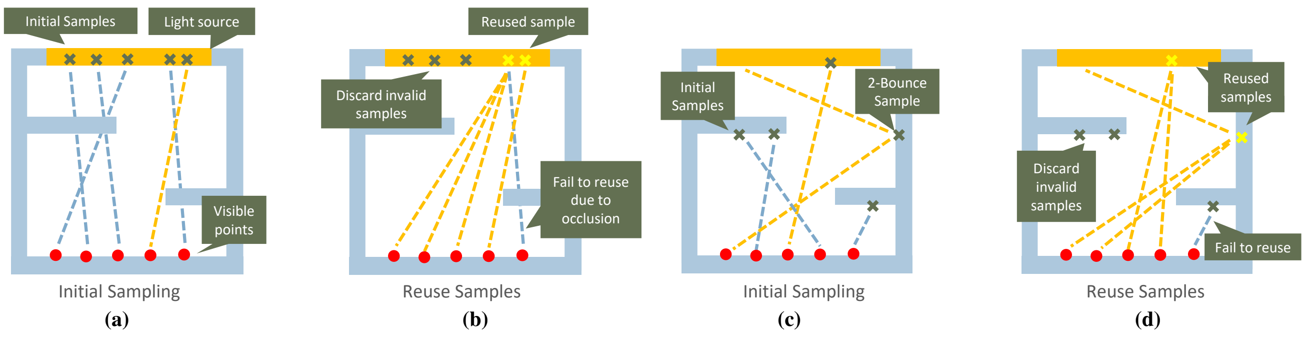

Figure 2: *Original ReSTIR and ReSTIR GI. (a) The original ReSTIR algorithm [BWP*20] starts by generating random samples on the lights in the scene. (b) After resampling, the original samples with no contribution are discarded; the useful samples are shared spatially and temporally and are used with probability based on their contribution. (c) Our approach generates initial samples by sampling random directions and tracing rays to find the closest intersections. Reflected radiance is computed at these intersections with path tracing. (d) Spatial and temporal resampling is applied in a similar manner. Doing so makes it possible to find directions that give meaningful indirect illumination, which is not handled by ReSTIR.*

Algorithm 1 Weighted Reservoir Sampling

```
 1: class RESERVOIR
 2:     SAMPLE $z$
 3:     $w \leftarrow 0$
 4:     $M \leftarrow 0$
 5:     $W \leftarrow 0$                                  ▷ Equation 7
 6:     procedure UPDATE(SAMPLE $S_{new}$, $w_{new}$)
 7:         $w \leftarrow w + w_{new}$
 8:         $M \leftarrow M + 1$
 9:         if $random() < w_{new}/w$ then
10:             $z \leftarrow S_{new}$
11:     procedure MERGE(RESERVOIR $r$, $\hat{p}$)
12:         $M_0 \leftarrow M$
13:         UPDATE($r.z$, $\hat{p} \cdot r.W \cdot r.M$)
14:         $M \leftarrow M_0 + r.M$
```

directions are an inconvenient representation. We therefore associate points on surfaces with the radiance they scatter back along an incident ray.

We will say that the *visible points* are the positions on surfaces in the scene that are visible from the camera at each pixel. At each visible point, a direction is randomly sampled and a ray is traced to find the closest surface intersection; these intersections are called *sample points*. Sample generation is described in more detail in Section 4.1. After sample points have been generated, resampling is performed and shaded values are computed at each visible point (Section 4.2). Figure 2 compares ReSTIR for direct lighting to ReSTIR GI and Figure 4 summarizes the algorithm.

The algorithm maintains three image-sized buffers that store the following values at each pixel:

- Initial sample buffer: a buffer of initial samples of type SAMPLE (Figure 3).
- Temporal reservoir buffer: a buffer of RESERVOIRS that accept samples from applying WRS to the previous samples generated in the pixel.
- Spatial reservoir buffer: a buffer of RESERVOIRS that accept samples from applying WRS to samples from nearby pixels.

```
struct SAMPLE
    float3 $x_v$, $\vec{n}_v$          ▷ Visible point and surface normal
    float3 $x_s$, $\vec{n}_s$          ▷ Sample point and surface normal
    float3 $\hat{L}_o$     ▷ Outgoing radiance at sample point in RGB
    float3 $Random$             ▷ Random numbers used for path
```

Figure 3: *Sample Representation. Each* SAMPLE *stores both the local geometry and outgoing radiance at the sample as well as the local geometry at the original visible point that generated the sample. The visible point geometry and the random numbers used for path tracing are used for the sample validation algorithm that is described in Section 4.3.*

4.1. Sample Generation

The first phase of our algorithm generates a new sample point for each visible point. Our implementation takes as input a G-buffer with the visible point's position and surface normal in each pixel, though it could also easily be used with ray-traced primary visibility.

For each pixel q with corresponding visible point x_v, we sample a direction ω_i using the source PDF $p_q(\omega_i)$ and trace a ray to obtain the sample point x_s. The source PDF may be a uniform distribution, a cosine-weighted distribution, or a distribution based on the BSDF at the visible point. (Section 6 has comparisons among them.) See Algorithm 2 for pseudo-code.

At each sample point, we need to compute the outgoing radiance $L_o(x_s, \omega_o)$, where ω_o is the normalized direction to the visible point. This radiance value can be computed in a variety of ways, though we apply Monte Carlo path tracing using next event esti-

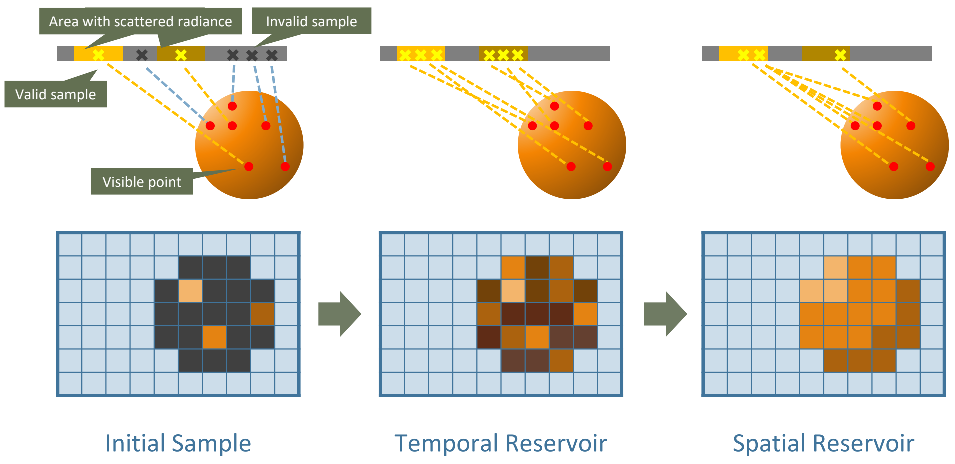

Figure 4: *Algorithm workflow. In each frame, we perform the following steps for each pixel:* Initial sampling*: we trace a ray from each visible point (red dots) with a random direction and record the closest intersection in a screen-space initial sample buffer. The position, normal and radiance of the intersection, the random numbers used in next event estimation, as well as the position and normal of the pixel, are recorded.* Temporal reuse*: we use the sample from the initial sample buffer to update temporal reservoir buffer by randomly choosing between the one created in current frame and the existing one in the buffer. Temporal reprojection is applied to find the corresponding temporal reservoir from the last frame.* Spatial reuse*: we use randomly-chosen temporal reservoirs from neighborhood pixels to update spatial reservoir. To suppress bias, we choose neighborhood pixels with similar geometric features by comparing their depth and normal with the current pixel's.*

Algorithm 2 Initial Sampling

```
1: for each pixel $q$ do
2:     Retrieve visible point $x_\mathrm{v}$ and normal $\vec{n}_\mathrm{v}$ from GBuffer
3:     Sample random ray direction $\omega_\mathrm{i}$ from source PDF $p_q$
4:     Trace ray to find sample point $x_\mathrm{s}$ and normal $\vec{n}_\mathrm{s}$
5:     Estimate outgoing radiance $\hat{L}_\mathrm{o}$ at $x_\mathrm{s}$
6:     $InitialSampleBuffer[q] \leftarrow \textsc{Sample}(x_\mathrm{v}, \vec{n}_\mathrm{v}, x_\mathrm{s}, \vec{n}_\mathrm{s}, \hat{L}_\mathrm{o})$
```

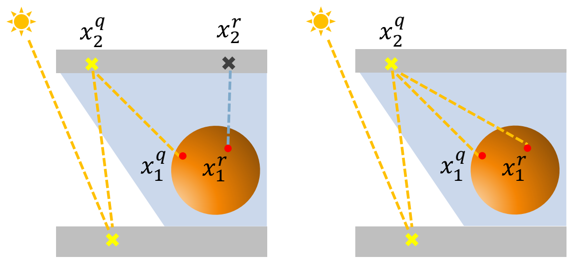

Figure 5: *Multi-bounce GI. At each sample point x_2^*, we estimate the radiance scattered to the corresponding visible point using path tracing. By connecting to the last path vertex, other visible points are able to reuse the contribution from the entire path.*

mation (NEE) and multiple importance sampling at each vertex. If only direct lighting is included in the radiance estimate, then our algorithm computes one-bounce global illumination. In general, following n path-traced bounces corresponds to $n+1$ bounce global illumination; see Figure 5.

4.2. Resampling and Shading

After the fresh initial sample is taken, spatial and temporal resampling is applied. The target function

$$\hat{p} = L_\mathrm{i}(x_\mathrm{v}, \omega_\mathrm{i})\, f(\omega_\mathrm{o}, \omega_\mathrm{i})\, \langle \cos\theta_\mathrm{i} \rangle = L_\mathrm{o}(x_\mathrm{s}, -\omega_\mathrm{i})\, f(\omega_\mathrm{o}, \omega_\mathrm{i})\, \langle \cos\theta_\mathrm{i} \rangle \tag{9}$$

includes the effect of the BSDF and cosine factor at the visible point, though we have also found that the simple target function

$$\hat{p} = L_\mathrm{o}(x_\mathrm{s}, -\omega_\mathrm{i}) \tag{10}$$

works well. While it is a suboptimal target function for a single pixel, we have found that it is helpful for spatial resampling in that it preserves samples that may be effective at pixels other than the one that initially generated it.

After initial samples are generated, temporal resampling is applied. In this stage, for each pixel, we read the sample from the initial sample buffer, and use it to randomly update temporal reservoir, computing the RIS weight following Equation 5 with the source PDF as the PDF for the sampled direction $p_q(\omega_\mathrm{i})$ and $\hat{p}$ as defined in Equation 10. The pseudo-code for temporal resampling is shown in Algorithm 3.

Algorithm 3 Temporal Resampling

```
1: for each pixel $q$ do
2:     $S \leftarrow InitialSampleBuffer[q]$
3:     $R \leftarrow TemporalReservoirBuffer[q]$
4:     $w \leftarrow \hat{p}_q(S)/p_q(S)$                ▷ Equations 5 and 9 or 10
5:     $R.\textsc{Update}(S, w)$
6:     $R.W \leftarrow R.w/(R.M \cdot \hat{p}(R.z))$           ▷ Equation 7
7:     $TemporalReservoirBuffer[q] \leftarrow R$
```

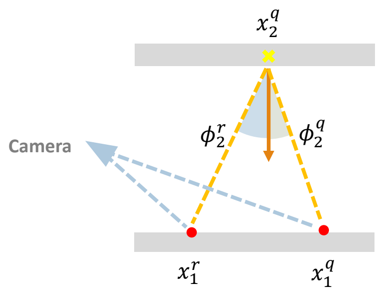

Figure 6: *If a visible point x_1^q generates a sample point x_2^q that is reused at another visible point x_1^r, then the Jacobian determinant in Equation 11 accounts for the fact that x_q^r would have itself generated the sample point x_2^q with a different probability.*

After temporal use, spatial reuse is applied. Samples are taken from the temporal reservoirs at nearby pixels, and resampled into a separate spatial reservoir. (See Algorithm 4 for pseudo-code.) With spatial reuse, it is necessary to account for differences in the source PDF between pixels that are due to the fact that our sampling scheme is based on the visible point's position and surface normal. (Such a correction was not necessary in the original ReSTIR algorithm since it sampled lights directly without considering each pixel's local geometry.) Therefore, when we reuse a sample from a pixel q at a pixel r, we must transform its solid angle PDF to current pixel's solid angle space by dividing it by the Jacobian determinant of the corresponding transformation [KMA*15, Equation 13]:

$$|J_{q\to r}| = \frac{|\cos(\phi_2^r)|}{|\cos(\phi_2^q)|} \cdot \frac{\|x_1^q - x_2^q\|^2}{\|x_1^r - x_2^q\|^2} \tag{11}$$

where x_1^q and x_2^q are the first and second vertex of the reused path, x_1^r is the visible point from the destination pixel, and ϕ_2^q and ϕ_2^r are the angles formed by the vectors $x_1^q - x_2^q$ and $x_1^r - x_2^q$ with the normal at x_2^q (Figure 6). Figure 7 shows the importance of this factor.

Pseudocode for our spatial resampling algorithm is shown in Algorithm 4. It includes a geometric similarity test following Bitterli et al.'s [BWP*20], requiring that the surface normals be within 25° and the two normalized depths to be within 0.05.

After both stages of reuse, the final scattered radiance at a visible point x_v due to indirect illumination is given by the RIS estimator, Equation 6, where the spatial reservoir's W weight gives all of the factors other than $f(y)$, which is evaluated as the product of the BSDF, cosine factor, and the reservoir sample's outgoing radiance.

4.3. Bias

Similar to the original ReSTIR algorithm, the ReSTIR GI algorithm has both biased and unbiased forms. Some sources of bias are easily corrected, while others may require more work (e.g. tracing a ray). Depending on performance requirements, biased variants may be desirable in order to trade off bias for improved performance. Bias can be introduced from both spatial and temporal sample reuse; we will consider each in turn.

As with ReSTIR for direct lighting, spatial resampling can introduce bias due to the fact that different source PDFs are used at

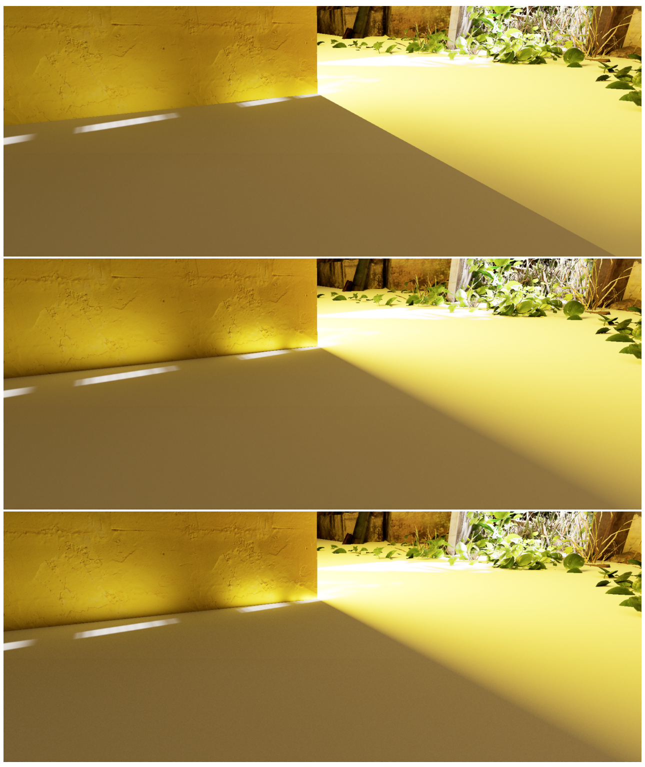

Figure 7: *Effect of the Jacobian determinant, Equation 11, in spatial resampling. The wall receives sunlight and indirectly illuminates the floor. Top: ignoring the Jacobian results in lighting discontinuities on the floor and overestimated lighting at the base of the wall. Middle: including the Jacobian corrects these artifacts. Bottom: path traced ground truth.*

Algorithm 4 Spatial Resampling

```
1:  for each pixel $q$ do
2:      $R_s \leftarrow SpatialReservoirBuffer[q]$
3:      $Q \leftarrow q$
4:      for s=1 to maxIterations do
5:          Randomly choose a neighbor pixel $q_n$
6:          Calculate geometric similarity between $q$ and $q_n$
7:          if similarity is lower than the given threshold then
8:              continue
9:          $R_n \leftarrow TemporalReservoirBuffer[q_n]$
10:         Calculate $|J_{q_n \to q}|$                         ▷ Equation 11
11:         $\hat{p}'_q \leftarrow \hat{p}_q(R_n.z)/|J_{q_n \to q}|$
12:         if $R_n$'s sample point is not visible to $x_v$ at $q$ then
13:             $\hat{p}'_q \leftarrow 0$
14:         $R_s.\textsc{Merge}(R_n, \hat{p}'_q)$
15:         $Q \leftarrow Q \cap q_n$
16:     $Z \leftarrow 0$
17:     for each $q_n$ in $Q$ do
18:         if $\hat{p}_{q_n}(R_s.z) > 0$ then
19:             $Z \leftarrow Z + R_n.M$                          ▷ Bias correction
20:     $R_s.W \leftarrow R_s.w/(Z \cdot \hat{p}_q(R_s.z))$           ▷ Equation 7
21:     $SpatialReservoirBuffer[q] \leftarrow R_s$
```

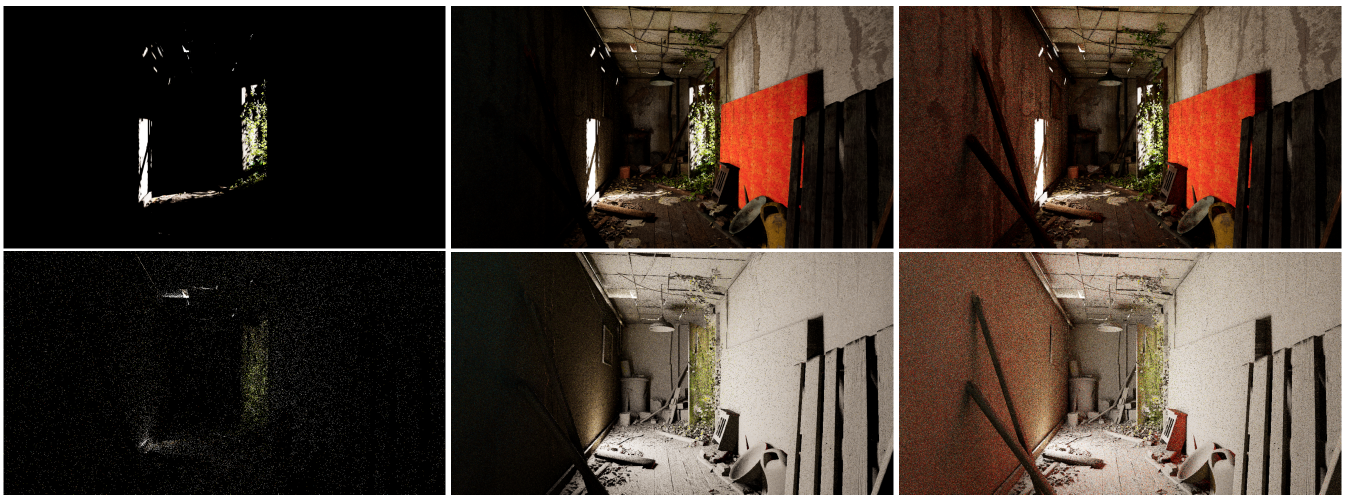

Figure 8: *Comparison between direct light, 1-bounce and 2-bounce GI rendered with our algorithm. Top row, left to right: scene rendered with direct lighting, one-bounce, and two-bounce global illumination. Bottom row: the indirect illumination alone, left to right: two-bounce path tracing, one-bounce, and two-bounce ReSTIR GI result. Rendered at* 1920×1050 *resolution on an NVIDIA RTX 3080 GPU, for ReSTIR GI, initial sampling takes 3.2 ms for one bounce and 4.2 ms for two. Sample reuse uses 4.6 ms in both cases. The 2-bounce path tracing takes 6.9 ms.*

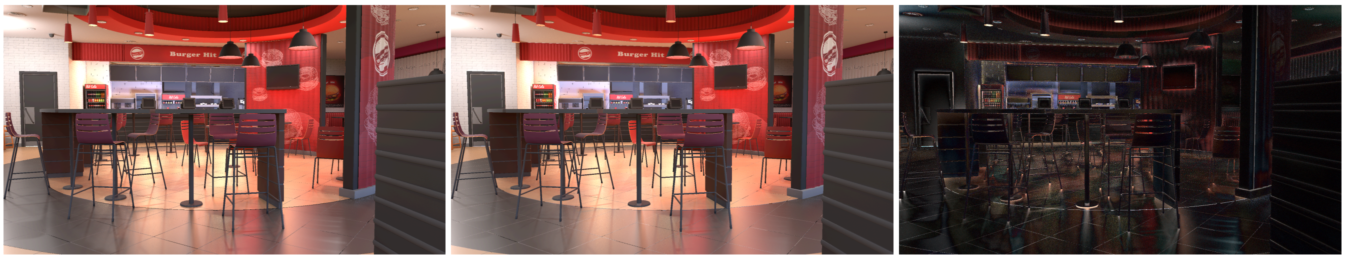

Figure 9: *Bias caused by spatial reuse. Left: Unbiased result. Middle: Biased result. Right:* $10\times$ *difference. Note that bias is mainly in shadow areas due to visibility changes. The glossy floor worsens bias around the chair legs because the target function includes glossy reflection and tends to place more samples around the specular reflection direction.*

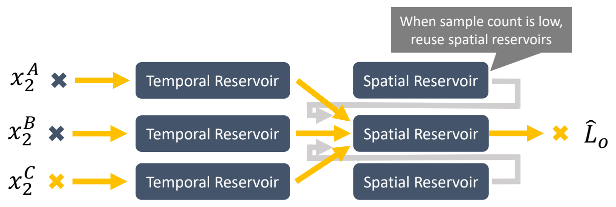

Figure 10: *Data flow of ReSTIR GI. In each frame, each pixel's temporal reservoir accepts newly generated samples. Instead of always reusing other spatial reservoirs, each spatial reservoir only reuses temporal reservoirs from neighboring pixels in order to suppress bias. When sample count is low, each spatial reservoir reuses other spatial reservoirs in order to boost convergence.*

different pixels. If samples are reused from another pixel's reservoir where that pixel's source PDF does not cover the domain of the current pixel, bias occurs in the estimate results. This bias can be fully corrected by tracing a visibility ray to check which source distributions can sample the final chosen sample and then weight the result accordingly [WP21]. (This test is in line 18 of Algorithm 4.) Alternatively, it can be reduced without tracing a ray by including a geometric similarity test between pixels.

Figure 9 shows the comparison between biased and unbiased sample reuse. In this figure, we set the target function to be the outgoing light radiance, with the BSDF term including both diffuse and specular components. Bias is mainly in shadow area due to visibility change. Note that the glossy floor makes bias worse because the target function with glossy component tends to place more samples around the reflection direction, where unfortunately a complex visibility change occurs.

Another way to reduce the bias from spatial reuse is for spatial reservoirs to only operate on temporal reservoirs from nearby pixels and not spatial reservoirs, as was done in Algorithm 4. In this way, bias does not accumulate due to multiple passes of resampling biased samples. However, newly disoccluded pixels may not be able to collect enough samples if only temporal reservoirs are used, which results in visible noise. In this case, we allow spatial reservoirs with low input sample counts to reuse nearby spatial reservoirs to improve convergence speed. Figure 10 shows the whole dataflow.

As discussed in Section 4.2, ReSTIR GI can also suffer from bias because by using BSDF sampling, the source distribution depends on each pixel's local geometry. The same distribution with respect to area measure has different shapes in different pixel's solid angle space. Reusing samples from other pixels without considering this difference causes bias. This bias is easily corrected by evaluating Equation 11; there is no reason not to include this factor, given its minimal computational expense.

If lighting or the scene geometry is changing from frame to frame, then temporal reuse may cause bias if the outgoing radiance values stored at the sample points become inaccurate. This problem is exacerbated by ReSTIR's characteristic of tending to keep relatively bright samples in the reservoirs for many frames before they are replaced; the result may be a noticeable lag in updates to the indirect illumination. To mitigate this problem, we apply a sample validation mechanism inspired by the A-SVGF denoising filter [SPD18]. Every few frames we re-trace the rays to recompute the outgoing radiance for all reservoir samples and check if the resulting radiance is in a given tolerance range, clearing the reservoir if it is not. (In this stage it is important that the same random numbers are used for random sampling as were used when the samples were originally generated.) The frame interval for sample verification can be adjusted depending on how dynamic the scene is.

Another source of temporal bias in dynamic scenes can come from an occluder that blocks the ray between visible point and sample point after the sample point was first generated. This bias can be corrected by tracing a shadow ray to the sample point from the visible point during sample verification.

5. Implementation

We have implemented our algorithm in *Unreal Engine 4* and *Falcor* [BYC*20]. In *Unreal Engine 4* implementation, the initial G-buffer is generated using rasterization before a full-screen pass generates the new samples for each visible point. The *Falcor* implementation is similar, though it uses ray tracing to generate G-buffer. Both temporal and spatial resampling are handled in a subsequent full-screen pass. Temporal resampling uses reprojected pixels according to their motion vectors from the previous frame. If temporal reprojection fails, we reset both the spatial and temporal reservoirs before performing spatial resampling.

For efficiency, our implementation neglects the directional variation in scattered radiance at the sample point. Instead, the scattered radiance in the direction to the original visible point is used for all directions, corresponding to Lambertian scattering. Thus, when a sample point is connected to a visible point that is different than the visible point that originally generated it, error may be introduced if the sample point's BSDF is not purely diffuse. Note that this simplification does not require that the visible point's BSDF be Lambertian, nor does it require Lambertian scattering at subsequent vertices of the indirect path.

We use double-buffering for both the temporal and spatial reservoir buffers so that spatial resampling can access other pixels' temporal buffers without data races. This introduces a one-frame lag in the indirect lighting, which is not problematic at high frame rates.

For storage in off-chip memory, surface normals in the SAMPLE structure are compressed to four bytes and half-precision floating point is used for the outgoing radiance; each reservoir then requires 48 bytes of storage. With one initial reservoir and two for both spatial and temporal reservoirs due to double-buffering, the total memory requirements at 1080p resolution (excluding the G-buffer, which we assume is required for other uses) are 475 MB. Approximately 570 MB of bandwidth is used for reading data from reservoirs each frame and 285 MB for writing updated reservoirs to memory.

In practice, sampling long paths in every pixel of every frame may cause a significant performance impact. As a result, we only follow multi-bounce paths in 25% of the pixels; doing so gives good results with a much lower performance cost. Randomly choosing pixel for these paths can hurt performance because of low thread coherency. Therefore, we divide the screen into tiles of 64×32 pixels and apply Russian roulette at the tile level. Tiles that fail the Russian roulette test are rendered using single-bounce indirect paths, just computing direct illumination at the sample point. Those that pass follow multi-bounce paths, but are reweighted using the Russian roulette probability. In this way, the expected value of all paths is that of a multi-bounce path. In practice, the stochastic nature of temporal and spatial resampling hides the tile pattern well.

We clamp the value of M in the temporal reservoirs to 30 and in the spatial reservoirs to 500 so that the reservoirs do not become stuck with a particular sample and be unlikely to replace it as M grows large.

We validate all output samples from the spatial reservoir buffer in a given frame interval. We reuse the initial sampling pass to perform sample validation so that it does not have extra performance cost. In the sample validation frame, the initial sampling pass reads sample information and traces the same rays to validate radiance values. By default, we validate samples every 6 frames.

The *maxIterations* parameter of the spatial resampling algorithm has a significant effect on performance: higher values lead to faster convergence but also increase the algorithm's cost. In practice, reservoirs that have considered few samples generally exhibit higher levels of noise and may cause artifacts, so we set *maxIterations* according to the sample count in each pixel's spatial reservoir. When the sample count is below half of the maximum M value in spatial reservoir, we set *maxIterations* to 9 and otherwise, we set it to 3.

The search radius used for spatial resampling algorithm also affects the image quality. Setting its value too high may cause a low acceptance rate for scenes with high geometric complexity, while setting it too low can introduce low frequency noise. Because the optimal value largely depends on scene depth complexity, we use an adaptive search strategy. Specifically, we set the initial radius as 10% of the image resolution. During spatial reuse, the radius is left unchanged if another pixel's sample can be reused and is provided

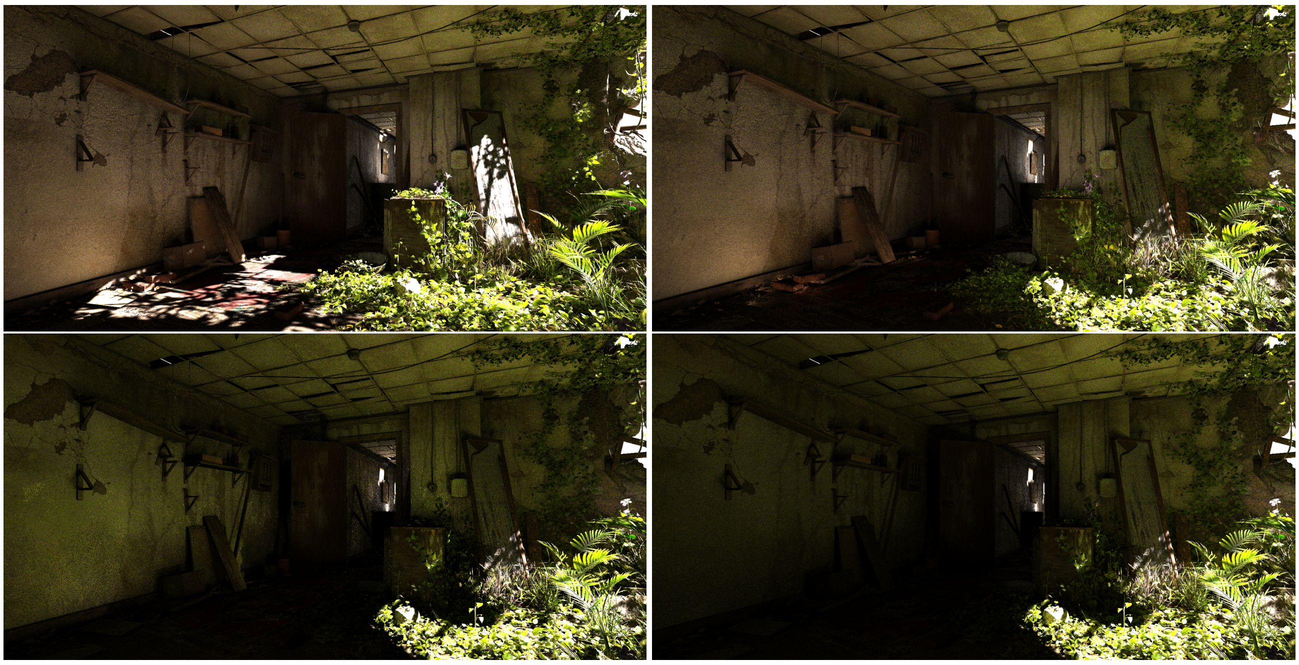

Figure 11: *Sample validation can effectively suppress bias caused by temporal reuse in dynamic scenes. Top left: original lighting. Top right: without sample validation, visible bias still remains after 12 frames of sudden change of sunlight—note the illumination on the wall on the left. Bottom left: with sample validation, the result rapidly adapts to changes in lighting. Bottom right: converged result.*

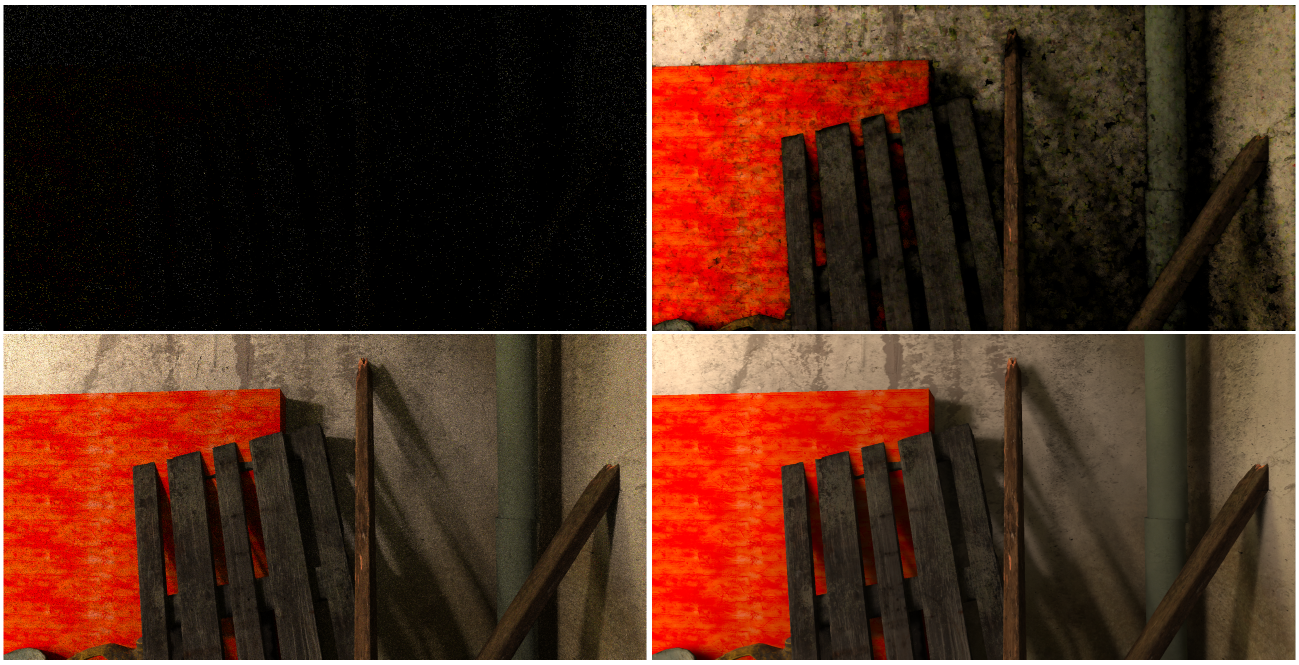

Figure 12: *ReSITR GI can produce superior results with popular spatio-temporal denoiser. The scene is a small room with sunlight only coming through a half-opened door on the left. 2-bounce GI is used in all shots and the Unreal Engine 4 default spatio-temporal denoiser is used for denoised results. Upper Left: Path traced result with 2 samples per pixel, rendered in 20.1 ms. Due to difficult lighting condition, the result is almost black with a small number of pixels containing super bright samples. Upper Right: Denoised path tracing result. Lower Left: Unbiased ReSTIR GI with 1 sample per pixel. The initial sampling pass uses 7.6 ms and spatio-temporal reuse pass uses 9.2 ms. Note that contact shadow details are well preserved. Lower Right: Denoised ReSTIR GI result.*

to the spatial reservoir. Otherwise, the radius is halved, down to a minimum search radius of 3 pixels, and kept at that value.

6. Results

We have evaluated ReSTIR GI with a variety of scenes to measure its effectiveness. All measurements were taken using an NVIDIA RTX 3090 GPU and none of the images have been denoised other than where explicitly noted. No post-processing effect except tone mapping has been applied. Timing measurements are reported as the time spent for the ReSTIR GI algorithm; other passes occupy less than 30% of the total time and G-buffer generation uses less than 1 ms per frame. The path tracer used for comparison is a standard path tracer that uses next event estimation for direct lighting and BSDF sampling to generate indirect rays. Reference images were computed using that path tracer with 8,192 samples per pixel.

Figure 8 shows the difference between single and multiple bounce global illumination. Multiple bounces can substantially improve lighting results for indoor scenes illuminated by strong sun light. Here multiple bounces are only computed in 25% of the tiles, following the approach described in Section 5. For this scene, which is rendered at 1920×1080 resolution, only considering 1-bounce global illumination uses 3.2 ms to trace rays and store initial samples, while the 2-bounce case costs 4.2 ms, an increased cost of 32%. Both cases use 4.6 ms for resampling. Total frame time including resampling is 14 ms.

Sample validation can effectively suppress temporal bias in scenes with dynamic lighting. In Figure 11, after a sudden change of sunlight, the algorithm without sample validation still keeps stale samples on the wall, causing incorrect lighting. With sample validation, all stale samples are discarded and the algorithm adapts to new lighting quickly. However, bias still exists due to overestimated lighting using new samples.

Figure 12 shows the effect of denoising with regular path tracing and ReSTIR GI using the default spatio-temporal denoiser in Unreal Engine 4.25, which performs temporal accumulation followed by spatial filtering and post filtering. The path traced image exhibits significant low-frequency noise even after denoising, while ReSTIR GI has much less noise and preserves contact shadow details.

Figure 13 compares the convergence of our algorithm to standard path tracing in roughly equal time for a variety of complex scenes that range from 1.8 million to 8.3 million triangles. Our approach successfully captures abundant lighting details, while standard path tracing gives noisy results. For ReSTIR GI, the images shown are captured after 32 frames of temporal resampling. ReSTIR GI provides a reduction in MSE ranging from $14.6\times$ to $141\times$ for the biased variant, and $9.3\times$ to $166\times$ for the unbiased version of it. The accompanying video demonstrates the improvements for sequences showing dynamic lights and moving cameras.

The bias of the ReSTIR GI algorithm is mainly caused by visibility changes in neighborhood pixels. Figure 9 shows the results from biased and unbiased spatial reuse. However, the spatial reuse result shows bias in shadow areas, which is caused by reusing reservoirs outside the shadow area. Comparing geometric similarity cannot discard these reservoirs. Besides, the glossy reflection component also worsens bias on the floor.

The choice of initial sampling method and target PDF also affects the result. We found that using uniform hemisphere sampling for initial samples causes lower variance than cosine-weighted sampling, especially for light from grazing angles. In that case, extra variance is caused by low sampling probability in light directions; see Figure 14. We further found that only using outgoing radiance (Equation 10) rather than the scattered radiance of Equation 9 gives a more stable result, though with higher variance.

6.1. Limitations

The cost of ReSTIR GI may still be too high for some real-time applications, especially with lower-end GPUs. For example, most video games expect >30 fps performance, which usually allows less than a 2 ms computational budget for global illumination. In such cases, the resolution of the reservoir buffers can be decreased in order to reduce computation. However, in this case, spatial reuse may be unstable when there are detailed normal maps. To improve stability while preserving lighting details, we have performed early experiments with computing geometric normals from the depth buffer, using spherical harmonics to record the lighting and finally recover details at full resolution using the original normals.

While using screen space buffers to store sample reservoirs gives a data structure that is easily sampled from and updated, this representation does have shortcomings. If the camera is moving quickly, newly visible pixels may be insufficiently sampled to give good results, which in turn leads to high noise. Furthermore, screen space may not be the best representation if there are perfectly specular objects in the scene. If a perfectly specular object is directly visible to the camera, the visible point should be at the first non-specular object that is visible along the ray path from the camera (similarly to as is commonly done in denoisers). In this case, spatial reuse is likely to be less effective as the visible points at nearby pixels may not be nearby in the scene.

Our approach is also not effective with multi-bounce global illumination with highly glossy surfaces. Not only is our assumption of Lambertian scattering at sampled points inaccurate in that case, but spatial sample reuse will be much less effective since a sample point that makes a large contribution at one pixel may be outside the peak specular lobe of another. If indirect lighting is concentrated in a small set of directions, it may be difficult to sample effectively, even with spatial and temporal reuse.

Finally, due to temporal and spatial reuse, the ReSTIR GI algorithm outputs correlated samples. That means that each frame's sample is to some extent similar to the previous frame's. However, many spatio-temporal denoisers assume their input to be independent. For example, the popular SVGF denoiser [SKW*17] calculates first and second moments for each pixel and then uses them to estimate variance, which is used to steer the bandwidth of spatial filter. Correlated input may cause artifacts due to an inexact estimate of variance and the resulting bandwidth.

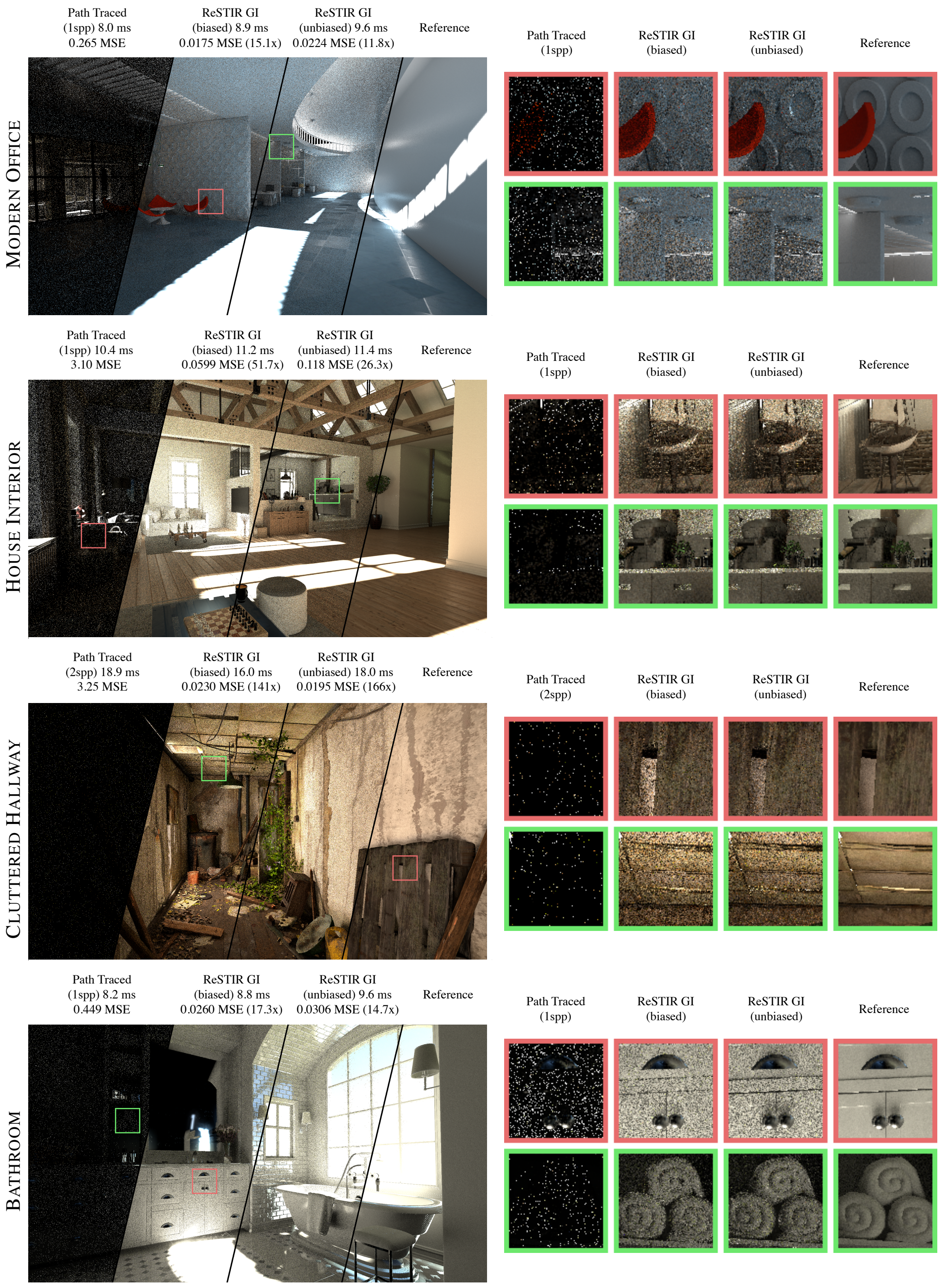
MODERN OFFICE
Path Traced (1spp) 8.0 ms 0.265 MSE
ReSTIR GI (biased) 8.9 ms 0.0175 MSE (15.1x)
ReSTIR GI (unbiased) 9.6 ms 0.0224 MSE (11.8x)
Reference
Path Traced (1spp)
ReSTIR GI (biased)
ReSTIR GI (unbiased)
Reference
HOUSE INTERIOR
Path Traced (1spp) 10.4 ms 3.10 MSE
ReSTIR GI (biased) 11.2 ms 0.0599 MSE (51.7x)
ReSTIR GI (unbiased) 11.4 ms 0.118 MSE (26.3x)
Reference
Path Traced (1spp)
ReSTIR GI (biased)
ReSTIR GI (unbiased)
Reference
CLUTTERED HALLWAY
Path Traced (2spp) 18.9 ms 3.25 MSE
ReSTIR GI (biased) 16.0 ms 0.0230 MSE (141x)
ReSTIR GI (unbiased) 18.0 ms 0.0195 MSE (166x)
Reference
Path Traced (2spp)
ReSTIR GI (biased)
ReSTIR GI (unbiased)
Reference
BATHROOM
Path Traced (1spp) 8.2 ms 0.449 MSE
ReSTIR GI (biased) 8.8 ms 0.0260 MSE (17.3x)
ReSTIR GI (unbiased) 9.6 ms 0.0306 MSE (14.7x)
Reference
Path Traced (1spp)
ReSTIR GI (biased)
ReSTIR GI (unbiased)
Reference

Figure 13: *Comparison of roughly equal time renderings of standard path tracing and ReSTIR GI, with two-bounce indirect illumination. Over a variety of scenes, ReSTIR GI exhibits a* $9.3\times$ *to* $166\times$ *improvement in mean squared error compared to path tracing at a similar computational cost.*

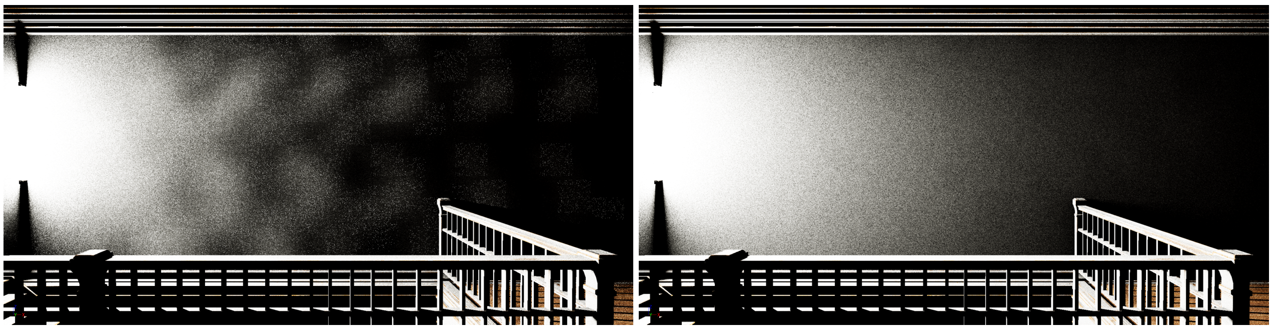

Figure 14: *Effect of the primary sampling technique. The scene is illuminated only by a light on the left. All samples are generated by BSDF sampling and no light sampling technique is used. (Left) Using cosine-weighted BSDF sampling leads to a noisy result due to a low sampling probability at grazing angles. (Right) Using uniform hemisphere sampling leads to a much better result.*

7. Conclusion and Future Work

We have shown that spatio- and temporal resampling using reservoirs leads to a highly effective algorithm for sampling indirect lighting. Our approach substantially reduces error for path traced indirect lighting, making it feasible for complex scenes at real-time frame rates on current GPUs. Compared to path tracing, we have shown reductions in MSE of up to 166× with our technique, with limited impact in frame time.

As discussed in Section 6.1, screen-space spatial resampling introduces a number of challenges. The first and most important to address is the ability to more effectively deal with non-Lambertian scattering events along light transport paths. Another interesting direction for future work is to consider other ways of generating sample points. Our approach finds them by randomly sampling directions from visible points, though another possibility could be to find them by tracing paths starting from light sources. For some scenes, this sampling approach may be more effective. Combining techniques such as those based on virtual point lights [DKH*14] with our resampling technique may lead to yet more ways of sampling indirect lighting.

Acknowledgements

Thanks to Aaron Lefohn, David Luebke and Calvin Lin for supporting this work. Unreal Engine 4 scenes in Figure 7, 8, 11, 12 and the third scene in Figure 13 are from *Quixel Megascans*, while the one in Figure 1 and other scenes in Figure 13 are from *Evermotion*. Unreal Engine 4 scene in Figure 14 is from *Matima Studio*. All of them can be found from *Unreal Marketplace*. The scene in Figure 9 is from [Ast16], rendered by *NVIDIA Falcor* [BYC*20].

References

[Ast16] ASTUFF: Burger restaurant, 03 2016. URL: https://www.turbosquid.com/3d-models/burger-restaurant-3d-model/1021436. 12

[BMDS19] BAKO S., MEYER M., DEROSE T., SEN P.: Offline deep importance sampling for Monte Carlo path tracing. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 527–542. 2

[BPE17] BAUSZAT P., PETITJEAN V., EISEMANN E.: Gradient-domain path reusing. *ACM Trans. Graph. 36*, 6 (Nov. 2017). URL: https://doi.org/10.1145/3130800.3130886, doi:10.1145/3130800.3130886. 3

[BSH02] BEKAERT P., SBERT M., HALTON J.: Accelerating path tracing by re-using paths. In *Proceedings of the 13th Eurographics Workshop on Rendering* (Goslar, DEU, 2002), EGRW '02, Eurographics Association, p. 125–134. 3

[BWP*20] BITTERLI B., WYMAN C., PHARR M., SHIRLEY P., LEFOHN A., JAROSZ W.: Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Trans. Graph. 39*, 4 (July 2020). URL: https://doi.org/10.1145/3386569.3392481, doi:10.1145/3386569.3392481. 1, 2, 3, 4, 6

[BYC*20] BENTY N., YAO K.-H., CLARBERG P., CHEN L., KALLWEIT S., FOLEY T., OAKES M., LAVELLE C., WYMAN C.: The Falcor rendering framework, 08 2020. URL: https://github.com/NVIDIAGameWorks/Falcor. 8, 12

[CBH*18] CHAITANYA C. R. A., BELCOUR L., HACHISUKA T., PREMOZE S., PANTALEONI J., NOWROUZEZAHRAI D.: Matrix bidirectional path tracing. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations* (2018), Jakob W., Hachisuka T., (Eds.), The Eurographics Association. doi:10.2312/sre.20181169. 3

[Cha82] CHAO M.-T.: A general purpose unequal probability sampling plan. *Biometrika 69*, 3 (1982), 653–656. 2, 3

[DGJ*20] DIOLATZIS S., GRUSON A., JAKOB W., NOWROUZEZAHRAI D., DRETTAKIS G.: Practical product path guiding using linearly transformed cosines. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 23–33. 2

[DHD20] DITTEBRANDT A., HANIKA J., DACHSBACHER C.: Temporal sample reuse for next event estimation and path guiding for real-time path tracing. In *Eurographics Symposium on Rendering* (2020), Dachsbacher C., Pharr M., (Eds.), The Eurographics Association. doi:10.2312/sr.20201135. 2

[DK16] DAHM K., KELLER A.: Learning light transport the reinforced way. In *International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing* (2016), Springer, pp. 181–195. 2

[DKH*14] DACHSBACHER C., KŘIVÁNEK J., HAŠAN M., ARBREE A., WALTER B., NOVÁK J.: Scalable realistic rendering with many-light methods. *Computer Graphics Forum 33*, 1 (2014), 88–104. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12256, doi:https://doi.org/10.1111/cgf.12256. 2, 12

[DWWH20] DENG H., WANG B., WANG R., HOLZSCHUCH N.: A practical path guiding method for participating media. *Computational Visual Media 6*, 1 (2020), 37–51. URL: https://doi.org/10.1007/s41095-020-0160-1, doi:10.1007/s41095-020-0160-1. 2

[HEV*16] Herholz S., Elek O., Vorba J., Lensch H., Křivánek J.: Product importance sampling for light transport path guiding. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 67–77. 2

[HKL16] Hedman P., Karras T., Lehtinen J.: Sequential Monte Carlo Instant Radiosity. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2016), ACM. 2

[HP02] Hey H., Purgathofer W.: Importance sampling with hemispherical particle footprints. In *Proceedings of the 18th spring conference on Computer Graphics* (2002), pp. 107–114. 2

[HPB07] Hašan M., Pellacini F., Bala K.: Matrix row-column sampling for the many-light problem. In *ACM SIGGRAPH 2007 Papers* (New York, NY, USA, 2007), SIGGRAPH '07, Association for Computing Machinery, p. 26–es. URL: https://doi.org/10.1145/1275808.1276410, doi:10.1145/1275808.1276410. 3

[Jen95] Jensen H. W.: Importance driven path tracing using the photon map. In *Eurographics Workshop on Rendering Techniques* (1995), Springer, pp. 326–335. 2

[Jen96] Jensen H. W.: Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96* (Berlin, Heidelberg, 1996), Springer-Verlag, p. 21–30. 2

[Kaj86] Kajiya J. T.: The rendering equation. *SIGGRAPH Comput. Graph. 20*, 4 (Aug. 1986), 143–150. URL: https://doi.org/10.1145/15886.15902, doi:10.1145/15886.15902. 1, 3

[KDB14] Keller A., Dahm K., Binder N.: Path space filtering. In *ACM SIGGRAPH 2014 Talks* (New York, NY, USA, 2014), SIGGRAPH '14, Association for Computing Machinery. URL: https://doi.org/10.1145/2614106.2614149, doi:10.1145/2614106.2614149. 3

[Kel97] Keller A.: Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 49–56. URL: https://doi.org/10.1145/258734.258769, doi:10.1145/258734.258769. 2

[KMA*15] Kettunen M., Manzi M., Aittala M., Lehtinen J., Durand F., Zwicker M.: Gradient-domain path tracing. *ACM Trans. Graph. 34*, 4 (July 2015). URL: https://doi.org/10.1145/2766997, doi:10.1145/2766997. 6

[LW95] Lafortune E. P., Willems Y. D.: A 5d tree to reduce the variance of Monte Carlo ray tracing. In *Eurographics Workshop on Rendering Techniques* (1995), Springer, pp. 11–20. 2

[McC13] McCombe J.: Low power consumption ray tracing. SIGGRAPH 2013 Course: Ray Tracing Is the Future and Ever Will Be, 2013. 1

[MGN17] Müller T., Gross M., Novák J.: Practical path guiding for efficient light-transport simulation. *Comput. Graph. Forum 36*, 4 (July 2017), 91–100. URL: https://doi.org/10.1111/cgf.13227, doi:10.1111/cgf.13227. 2

[MH20] Munkberg J., Hasselgren J.: Neural denoising with layer embeddings. *Computer Graphics Forum 39*, 4 (2020), 1–12. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14049, doi:https://doi.org/10.1111/cgf.14049. 1

[MMR*19] Müller T., Mcwilliams B., Rousselle F., Gross M., Novák J.: Neural importance sampling. *ACM Trans. Graph. 38*, 5 (Oct. 2019). URL: https://doi.org/10.1145/3341156, doi:10.1145/3341156. 2

[MMSM21] Majercik Z., Marrs A., Spjut J., McGuire M.: Scaling probe-based real-time dynamic global illumination for production. *Journal of Computer Graphics Techniques (JCGT) 10*, 2 (May 2021), 1–29. URL: http://jcgt.org/published/0010/02/01/. 2

[NVI18] NVIDIA I.: NVIDIA Turing GPU architecture. NVIDIA Whitepaper, 2018. 1

[Pan20] Pantaleoni J.: Online path sampling control with progressive spatio-temporal filtering. *SN Computer Science 1* (08 2020). doi:10.1007/s42979-020-00291-z. 2

[PJH16] Pharr M., Jakob W., Humphreys G.: *Physically Based Rendering: From Theory To Implementation*. Morgan Kaufmann, Burlington, Massachusetts, 2016. 2, 3

[PRDD15] Popov S., Ramamoorthi R., Durand F., Drettakis G.: Probabilistic connections for bidirectional path tracing. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering) 34*, 4 (2015). URL: http://www-sop.inria.fr/reves/Basilic/2015/PRDD15b. 3

[RHL20] Ruppert L., Herholz S., Lensch H. P.: Robust fitting of parallax-aware mixtures for path guiding. *ACM Transactions on Graphics (TOG) 39*, 4 (2020), 147–1. 2

[SKW*17] Schied C., Kaplanyan A., Wyman C., Patney A., Chaitanya C. R. A., Burgess J., Liu S., Dachsbacher C., Lefohn A., Salvi M.: Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics* (New York, NY, USA, 2017), HPG '17, Association for Computing Machinery. URL: https://doi.org/10.1145/3105762.3105770, doi:10.1145/3105762.3105770. 1, 10

[SPD18] Schied C., Peters C., Dachsbacher C.: Gradient estimation for real-time adaptive temporal filtering. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 2 (Aug. 2018). URL: https://doi.org/10.1145/3233301, doi:10.1145/3233301. 1, 8

[TCE05] Talbot J. F., Cline D., Egbert P.: Importance resampling for global illumination. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques* (Goslar, DEU, 2005), EGSR '05, Eurographics Association, p. 139–146. 2, 3

[TH19] Tokuyoshi Y., Harada T.: Hierarchical Russian roulette for vertex connections. *ACM Trans. Graph. 38*, 4 (July 2019). URL: https://doi.org/10.1145/3306346.3323018, doi:10.1145/3306346.3323018. 3

[VHH*19] Vorba J., Hanika J., Herholz S., Müller T., Křivánek J., Keller A.: Path guiding in production. In *ACM SIGGRAPH 2019 Courses* (New York, NY, USA, 2019), SIGGRAPH '19, Association for Computing Machinery. URL: https://doi.org/10.1145/3305366.3328091, doi:10.1145/3305366.3328091. 2

[Vit85] Vitter J. S.: Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS) 11*, 1 (1985), 37–57. 2, 3

[VKŠ*14] Vorba J., Karlík O., Šik M., Ritschel T., Křivánek J.: On-line learning of parametric mixture models for light transport simulation. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 1–11. 2

[WGGH20] West R., Georgiev I., Gruson A., Hachisuka T.: Continuous multiple importance sampling. *ACM Transactions on Graphics (TOG) 39*, 4 (July 2020). doi:10.1145/3386569.3392436. 3

[WP21] Wyman C., Panteleev A.: Rearchitecting spatiotemporal resampling for production. In *Proceedings of ACM/EG Symposium on High Performance Graphics* (2021), HPG '21. 7

[ZZ19] Zheng Q., Zwicker M.: Learning to importance sample in primary sample space. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 169–179. 2