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

Rearchitecting Spatiotemporal Resampling for Production

Chris Wyman[†] and Alexey Panteleev [‡]

NVIDIA, Santa Clara, California, USA

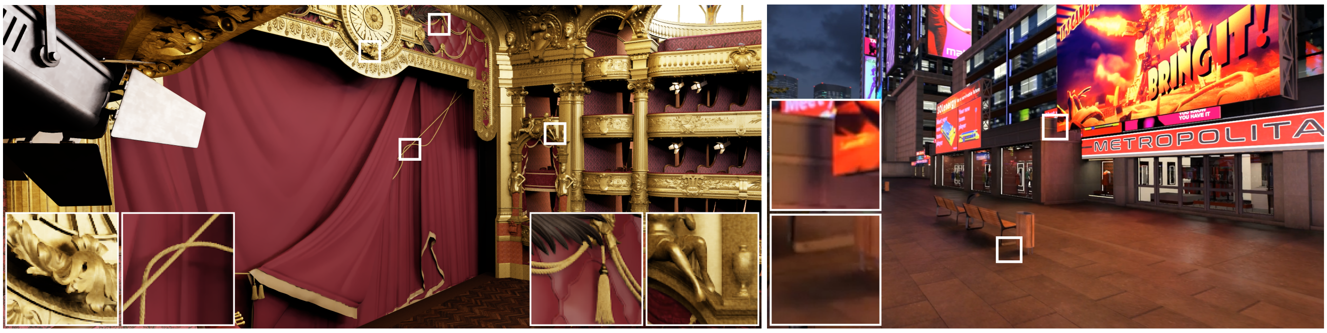


Figure 1: *Our rearchitected spatiotemporal resampling improves Bitterli et al.'s [BWP*20] ReSTIR with significantly lower cost and is shown here in two different renderers. (Left) Our Falcor [BYC*20] prototype shows the 150 million triangle Paris Opera House model (courtesy ©GoldSmooth from TurboSquid), with lighting at a cost of 11.8ms for* 3840×1440 *using two rays per pixel, on a RTX 3090. (Right) A dynamic night city scene, rendered in a driving simulator in 31 ms at* 2560×1440*, including 2001 analytic primitive lights and 453 emissive meshes (26,067 triangles). Both results have denoising applied [NVI20].*

Abstract

*Recent work by Bitterli et al. [BWP*20] introduced a real-time, many-light algorithm for rendering dynamic direct illumination from millions of lights by iteratively applying resampled importance sampling using weighted reservoir sampling. While enabling new levels of lighting complexity in real-time, the total cost remained beyond the budgets of even the most computationally demanding games. We introduce key algorithmic improvements developed while productizing this method that collectively reduce lighting costs by up to* $7\times$*, dramatically improve memory coherence, shrink the required ray budget, increase rendering quality, and expose parameters that enable trading quality for performance.*

CCS Concepts

• ***Computing methodologies*** → ***Rendering; Ray tracing;***

1. Introduction

Offline renderers have long used Monte Carlo ray and path tracing to stunning effect (e.g., [BAC*18, FHL*18]). With the introduction of hardware-accelerated ray tracing [Har20, KMSB18], developers can perform arbitrary visibility queries for real-time Monte Carlo integration. However, modern ray tracing hardware's throughput is limited, especially when targeting lower-end devices. Thus, developers often integrate ray tracing as optional eye candy for a subset of users. Combined with recent advances in fast, low-sample reconstruction (e.g., [SKW*17, SPD18]), this has significantly raised the image quality of games rendered at maximum graphics settings.

But many benefits are not attained without redesigning assets and rendering pipelines. For example, many-light rendering techniques can simplify artist workflow, replace multiple per-light shadow algorithms, and remove ambient occlusion passes. But improvements only accrue when new algorithms also work on lower-performance and legacy hardware. Without sufficient scaling, ray tracing's advantages become disadvantages: artists must tweak *another* asset variant and developers must maintain an *additional* code path.

We started productizing recent work [BWP*20] on many-light sampling. But their research prototype varied from 10 and 50 milliseconds per frame on high-end, prosumer GPUs; while an undeni-

[†] chris.wyman@acm.org; @_cwyman_

[‡] alpanteleev@nvidia.com; @more_fps

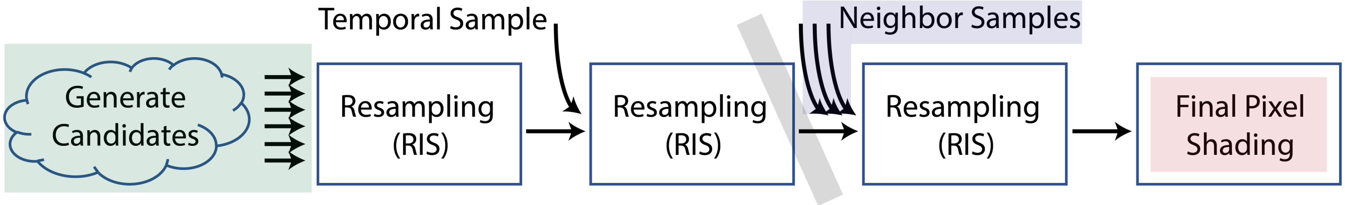


Figure 2: *An overview of the reservoir-based spatiotemporal importance resampling (ReSTIR) pipeline from Bitterli et al. [BWP*20]. We highlight above some specific performance bottlenecks addressed in this paper, including memory incoherence during initial candidate generation (Section 5), a global synchronization prior to spatial reuse (Section 6.2), determining optimal number of spatial neighbors (Section 6.3), ray count for shading (Section 6.1), and decoupling shading from reuse (Section 7).*

ably impressive price-to-performance ratio, even the minimal cost exceeds game budgets. But low-level performance analysis led to some interesting observations. First, while leveraging ray tracing, reservoir-based spatiotemporal importance resampling (ReSTIR) often spent less than $1/3$ of its time tracing rays. Second, the computational complexity of ReSTIR's non-ray computations is $O(1)$, but some component costs varied over 20× scene-to-scene.

To address these issues, we entirely rearchitected ReSTIR to improve both quality and performance. We achieved these gains by:

- Understanding the causes of bias and their relation to noise; this enables use of simple heuristics to reduce both bias and noise,
- Applying resampled importance sampling (RIS) [Tal05, TCE05] to *reshape* computations, extracting most of the incoherent memory fetches from the inner loop,
- Carefully analyzing parameter quality implications; this allows reducing the maximum ray budget from 5 to 2 rays per pixel,
- Removing a per-frame global barrier; selecting spatial samples from our temporal buffer removes intra-frame dependencies,
- Decoupling shading from sample reuse; this improves quality at a given cost by shading samples that resampling discards,
- Reusing visibility to trade quality for performance; in decoupled shading, one of our two rays per pixel *only* impacts shading. Selectively tracing that ray degrades quality, but reduces cost.

With traditional optimizations, such as data structure compression, reducing intermediate data, and deduping computations, we reduce cost up to 7× over Bitterli et al. [BWP*20]. At 1920×1080 with max settings, ReSTIR uses under 1.5 ms of compute plus two rays per pixel, for total cost of 1.9 to 5.1 ms (varying by scene) on an RTX 3090. By reducing settings, both costs decrease further.

2. Paper Overview

In this paper, we have multiple goals. First, Talbot et al. [TCE05] and Bitterli et al. [BWP*20] motivate use of importance resampling and spatiotemporal reuse for rendering, but today neither is widely used or discussed in textbooks (e.g., [DBB06, MS18, PJH16]). Section 3 walks through the underlying mathematics, covering the basics in one place to develop more of the underlying intuition.

Second, Bitterli et al. [BWP*20] derive how to unbiasedly reuse samples, but this remains non-intuitive. Section 4 restates the math, visually depicts bias, discusses how bias manifests, and reviews techniques that reduce and eliminate bias. This helps validate the soundness of spatiotemporal reuse to more skeptical readers.

Finally, we focus on improvements to drive higher efficiency (see Figure 2), including both theoretically- and empirically-motivated changes. Readers not interested in the underlying theoretical discussions can skip Sections 3 and 4.

Section 5 presents how we reshape computations to lift execution and data divergence from our inner loop. Section 6 outlines empirically motivated optimizations to our ray budget, sampling parameters, and data flow. And Section 7 traces the life of a sample, uncovering previously discarded work that we show drives improved quality and performance.

3. Background and Preliminaries

Before describing our rearchitected resampling, we review the math and highlight how ReSTIR benefits real-time rendering.

A fundamental challenge in real-time rendering is efficiently approximating the rendering equation [Kaj86], which has no analytic solution in most realistic scenarios:

$$L(\mathbf{x},\omega_o) = \int_\Omega \rho(\mathbf{x},\omega,\omega_o) L(\mathbf{x},\omega) \langle \vec{n}_\mathbf{x} \cdot \omega \rangle d\omega, \qquad (1)$$

This shades point $\mathbf{x}$ as viewed from direction ω_o, with surface normal $\vec{n}_\mathbf{x}$, BRDF ρ, and incident radiance $L(\mathbf{x},\omega)$. Instead, ray tracers typically approximate the rendering equation via importance sampled Monte Carlo integration:

$$F = \int_\Omega f(\omega) d\omega \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(\omega_i)}{p(\omega_i)}. \qquad (2)$$

In other words, we numerically approximate Equation 1 by sampling N directions ω_i per pixel, selecting ω_i with probability $p(\omega_i)$.

In real-time, minimizing ray count N is vital. Optimally, a *perfect* importance sampler reduces N to 1. This requires $p(\omega) \propto f(\omega)$, allowing cancellation in the summand from Equation 2. As probability density functions must integrate to unity, this means:

$$p(\omega) = \frac{f(\omega)}{\int f(\omega) d\omega}, \qquad (3)$$

but requires knowing F a priori. Introductory texts (e.g., [PJH16]) often show Equation 3 and conclude perfect sampling is unachievable. But can we *approximate* perfect importance sampling? Approximating may require $N > 1$ rays per pixel, but generally minimizes N. Suggesting *approximately-perfect* importance sampling seems odd, but it follows well-known statistical methods, e.g., see Devroye's [Dev86] section on *almost-exact* cdf inversion.

3.1. Resampled Importance Sampling

To approximate perfect importance sampling, we iteratively apply Monte Carlo integration. To sample according to the unnormalized *target* function $\hat{p}(\omega)$, we approximate the normalization:

$$p(\omega) = \frac{\hat{p}(\omega)}{\int \hat{p}(\omega)d\omega} \approx \frac{\hat{p}(\omega)}{\frac{1}{M}\sum_j \frac{\hat{p}(\omega_j)}{q(\omega_j)}}, \tag{4}$$

where $q(\omega)$ is the *source* pdf. Essentially, choose M samples from source pdf q to guide our N samples to best approximate target pdf $\hat{p}$. Merging back into Equation 2 and rearranging gives:

$$F = \int_\Omega f(\omega)d\omega \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{\hat{p}(\omega_i)}\frac{1}{M}\sum_{j=1}^{M}\frac{\hat{p}(\omega_{ij})}{q(\omega_{ij})}\right] \equiv \langle F\rangle_{ris}, \tag{5}$$

the resampled importance sampling (RIS) estimator, $\langle F\rangle_{ris}$, introduced to graphics by Talbot et al. [TCE05]. RIS reevaluates, or *resamples*, the M candidate samples using $\hat{p}$ and selects one proportional to weight $w_{ij} = \hat{p}(\omega_{ij})/q(\omega_{ij})$ as sample ω_i.

3.1.1. Why Is Resampled Importance Sampling Valid?

Astute readers may observe the right hand side of Equation 4 is not a pdf. Specifically, integrating over the hemisphere may not give unity. Remember, these are *approximate*, randomized distributions. Understanding them requires trusting their aggregate behavior, i.e., validating their expected values converge to our desired result. To do that, define a function $W(\omega,z)$:

$$W(\omega,z) = \frac{1}{\hat{p}(\omega_z)}\left[\frac{1}{M}\sum_{i=1}^{M} w_i(\omega_i)\right], \tag{6}$$

where ω_z explicitly represents a random sample from $\{\omega_1,\ldots,\omega_M\}$ selected proportional to weights $w_i = \hat{p}(\omega_i)/q(\omega_i)$, i.e., with probability $w_z/\sum w_i$. This allows rewriting Equation 5 as:

$$\langle F\rangle_{ris} = \frac{1}{N}\sum_{i=1}^{N} f(\omega_i)W(\omega,i), \tag{7}$$

and is unbiased as long as the expected value:

$$\mathbb{E}\left[W(\omega,z)\right] = \frac{1}{p(\omega_z)} = \frac{\int \hat{p}(\omega)d\omega}{\hat{p}(\omega_z)}. \tag{8}$$

This essentially requires $\mathbb{E}\left[\frac{1}{M}\sum \frac{\hat{p}(\omega_i)}{q(\omega_i)}\right] = \int \hat{p}(\omega)d\omega$, which is just Monte Carlo integration. But ensuring this expected value *continues* to converge to the desired result is a challenge as reuse becomes more complex, i.e., when iteratively resampling. This opens opportunities for bias, which we explore more deeply in Section 4.

3.1.2. Degenerate Resampled Importance Sampling

Examining RIS edge cases provides insights into its sampling characteristics and motivates some performance improvements we introduce later. One set of edge cases includes $M = 1$ and $M \to \infty$:

$$\begin{aligned} F &\approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{q(\omega_i)}\right] \quad \text{for } M=1, \text{ and} \\ F &\approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{\hat{p}(\omega_i)}\int_\Omega \hat{p}(\omega)d\omega\right] \quad \text{when } M\to\infty. \end{aligned} \tag{9}$$

For $M = 1$, RIS becomes Monte Carlo integration, sampling source pdf q. When $M = \infty$, RIS perfectly samples normalized target function $\hat{p}$. Between the edge cases, as M grows, RIS more and more closely samples according to target $\hat{p}$'s distribution, and (for good $\hat{p}$) the better overall importance sampling becomes.

Equation 9 provides insight into the value of using higher-quality source pdfs $q(\omega)$. For small M, the quality of q dramatically impacts the final estimate. As $M \to \infty$, the quality of q is irrelevant. For RIS and ReSTIR, in areas with large M, improving q has little impact; where M is small, improving q is worthwhile.

Another degeneracy arises when we define $\hat{p}(\omega) = q(\omega)$. In this case, Equation 5 becomes:

$$\begin{aligned} F &\approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{\hat{p}(\omega_i)}\frac{1}{M}\sum_{j=1}^{M}\frac{\hat{p}(\omega_{ij})}{q(\omega_{ij})}\right] \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{q(\omega_i)}\frac{1}{M}\sum_{j=1}^{M}1\right] = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{q(\omega_i)}\right]. \end{aligned} \tag{10}$$

This *also* gives a standard Monte Carlo estimator. But unlike Equation 9, which sets $M = 1$, this degenerate RIS *has two steps!* Algorithmically: choose an M-element subset from a universe of potential samples, then pick N of these elements for Monte Carlo integration. The key observation: any Monte Carlo estimator can be replaced with a two-step degenerate RIS estimator.

3.1.3. Resampled Importance Sampling as Stratification

A final edge case defines $\hat{p}(\omega) = f(\omega)$. Here, Equation 5 becomes:

$$F \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{\hat{p}(\omega_i)}\frac{1}{M}\sum_{j=1}^{M}\frac{\hat{p}(\omega_{ij})}{q(\omega_{ij})}\right] = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{M}\sum_{j=1}^{M}\frac{f(\omega_{ij})}{q(\omega_{ij})}\right]. \tag{11}$$

This shows RIS acting as a form of stratification over randomized subsets, rather than typical stratification on fixed sets. I.e., in each pixel, N sets of M random candidates ω_{ij} are generated:

$$\{\{\omega_{11},\ldots,\omega_{1M}\},\{\omega_{21},\ldots,\omega_{2M}\},\ldots,\{\omega_{N1},\ldots,\omega_{NM}\}\}, \tag{12}$$

and one random sample ω_i is selected from each of the N sets (proportional to weights w_{ij}).

Viewing Equation 5 as a stratified estimator allows us to also define an unstratified RIS estimator. This selects all N samples ω_i from the *same* set of M candidates, making the sums independent:

$$F = \int_\Omega f(\omega)d\omega \approx \left(\frac{1}{N}\sum_{i=1}^{N}\frac{f(\omega_i)}{\hat{p}(\omega_i)}\right)\left(\frac{1}{M}\sum_{j=1}^{M}\frac{\hat{p}(\omega_j)}{q(\omega_j)}\right). \tag{13}$$

The utility of viewing RIS as a form of random stratification may be unclear, but we revisit this observation in Section 5 when applying RIS to reshape our computations.

3.2. Weighted Reservoir Sampling

To minimize ray count, we aim to resample from the largest set M affordable. Best results can require $M > 1000$, but prior RIS methods [TCE05, TH16] retain candidates until resampling completes, requiring $O(M)$ storage that makes high sample counts infeasible.

Bitterli et al. [BWP*20] reformulates RIS using weighted reservoir sampling [Cha82, Vit85] to stream candidates, reducing storage to $O(N)$, i.e., only the selected samples ω_i remain in memory:

```
input:  stream $\mathbb{S}$ of samples $\omega_i$ on lights
output: reservoir $\mathcal{R} = \{\omega, w_{sum}, M\}$

$\mathcal{R} \leftarrow \{\emptyset, 0, 0\}$

for each $\omega_i$ in stream $\mathbb{S}$:
    $\mathcal{R}.w_{sum} \leftarrow \mathcal{R}.w_{sum}$ + weight( $\omega_i$ )
    $\mathcal{R}.M \leftarrow \mathcal{R}.M$ + 1
    if rand() < (weight( $\omega_i$ )/$\mathcal{R}.w_{sum}$) then
        $\mathcal{R}.\omega \leftarrow \omega_i$
```

Weighted reservoir sampling takes a straightforward inductive approach. Assuming a stream has been partially sampled with the desired distribution, to process new element ω_i we just need to determine whether to replace current sample $\mathcal{R}.\omega$ with ω_i. For RIS, our function weight(ω_i) is $w_i = \hat{p}(\omega_i)/q(\omega_i)$, and the probability to select a new sample is $w_i \;/\; (\sum_{j \leq i} w_j)$.

3.3. Spatiotemporal Importance Resampling

Weighted reservoir sampling enables memory-efficient RIS, but computational complexity remains $O(M)$. This is problematic as M must be large to closely approximate perfect importance sampling.

ReSTIR [BWP*20] proposes reusing neighbor and prior frame samples to substantially increase *effective* sample count, amortizing candidate sample costs over many pixels. This prefilters the pdfs, rather than traditional reconstruction and denoising methods, which postfilter color after rendering completes.

The key mathematical insight is that the approximately perfect importance sampling shown in Equation 4 need not use a traditional Monte Carlo estimator. It can use an RIS estimator:

$$p(\omega) = \frac{\hat{p}_0(\omega)}{\int \hat{p}_0(\omega) d\omega} \approx \frac{\hat{p}_0(\omega)}{\frac{1}{M_0} \sum_j \left[\frac{\hat{p}_0(\omega_j)}{\hat{p}_1(\omega_j)} \frac{1}{M_1} \sum_k \frac{\hat{p}_1(\omega_{jk})}{q(\omega_{jk})} \right]}. \tag{14}$$

Plugging this back into Equation 2 and rearranging gives us a two-iteration RIS estimator similar to Equation 5:

$$F \approx \frac{1}{N} \sum_{i=1}^{N} \left[\frac{f(\omega_i)}{\hat{p}_0(\omega_i)} \frac{1}{M_0} \sum_{j=1}^{M_0} \left[\frac{\hat{p}_0(\omega_{ij})}{\hat{p}_1(\omega_{ij})} \frac{1}{M_1} \sum_{k=1}^{M_1} \frac{\hat{p}_1(\omega_{ijk})}{q(\omega_{ijk})} \right] \right] \tag{15}$$

We can iterate an arbitrary number of times, i.e., using Equation 15 as the estimator in Equation 14, giving:

$$F \approx \frac{1}{N} \sum_{i=1}^{N} \left[\frac{f(\omega_i)}{\hat{p}_0(\omega_i)} \frac{1}{M_0} \sum_{j=1}^{M_0} \left[\frac{\hat{p}_0(\omega_{ij})}{\hat{p}_1(\omega_{ij})} \frac{1}{M_1} \sum_{k=1}^{M_1} \left[\frac{\hat{p}_1(\omega_{ijk})}{\hat{p}_2(\omega_{ijk})} \cdots \right] \right] \right] \tag{16}$$

Per-iteration target functions, $\hat{p}_i(\omega)$, are very flexible. They can estimate distributions in spatial or temporal neighbors, or the current pixel. They can include some, or all, of the rendering equation (e.g., visibility, incident lighting, BRDF, or cosine terms). Biased and ad hoc target functions can be used without biasing results, as long as $q(\omega) > 0$ and $\hat{p}_i(\omega) > 0$ whenever $f(\omega) > 0$.

Intuitively, read Equation 16 as: N need not be large, as we approximate perfect importance sampling with quality controlled by M_0; but M_0 is minimized by (again) importance sampling with quality driven by M_1; but due to good sampling, M_1 need not be large, driven by M_2; et cetera. We either need *one* term M_i or the product $\prod_i M_i$ to tend towards infinity. To make this cheap, we amortize costs. By reusing samples, we pay for computation once but each sample increases M_i values for thousands of neighbors.

3.4. Variance of Resampling Importance Sampling

Of course, this intuitive interpretation argues iterative resampling reduces variance. Talbot et al. [Tal05, TCE05] provide a theoretical variance analysis of RIS, allowing a detailed understanding. They show the variance of our RIS estimator $\langle F \rangle_{ris}$ can be written:

$$Var(\langle F \rangle_{ris}) = \frac{1}{N} \left[\frac{1}{M}(e_3 - e_2) + (e_2 - e_1) \right], \tag{17}$$

where

$$e_1 = \mathbb{E}\left[\frac{f}{q}\right]^2, \qquad e_2 = \mathbb{E}\left[\frac{f^2}{\hat{p}q}\right] \mathbb{E}\left[\frac{\hat{p}}{q}\right], \qquad e_3 = \mathbb{E}\left[\frac{f^2}{q^2}\right].$$

N and M both control variance from $e_3 - e_2$, whereas only N affects $e_2 - e_1$. Useful corner cases include: if $\hat{p} \propto q$, then $e_3 - e_2 = 0$; if $\hat{p} \propto f$, then $e_2 - e_1 = 0$.

When selecting $\hat{p} \propto f$ to ensure $e_2 - e_1 = 0$, iteratively applying Equation 17 indeed has variance proportion to $1/(N \prod_i M_i)$. But in practice $\hat{p} \propto f$ is not feasible, leading to residual variance proportional to $1/N$, $1/(NM_0)$, $1/(NM_0M_1)$, and other partial products.

In Bitterli et al. [BWP*20], N largely controls visibility variance, as $\hat{p}$ is proportional to f modulo visibility. During spatiotemporal reuse, the magnitude $|e_2 - e_1|$ depends on how well reused samples approximate values in our current pixel.

For small residuals, where a pixel and its neighbors are similar, variance is dominated by the $1/(N \prod_i M_i)$ term and the additional samples from reuse improve quality significantly. If neighbors differ significantly, generally near various discontinuities, $|e_2 - e_1|$ becomes large. Reusing from such neighbors may be detrimental.

3.5. Comments on Notation and Generalization

To improve clarity above, we avoided explicitly adding subscripts to all values that might vary during resampling. Specifically, M values can vary with i and source pdfs q need not be the same for every sample j. In other words, we can write Equation 5:

$$F \approx \frac{1}{N} \sum_{i=1}^{N} \left[\frac{f(\omega_i)}{\hat{p}(\omega_i)} \frac{1}{M_i} \sum_{j=1}^{M_i} \frac{\hat{p}(\omega_{ij})}{q_{ij}(\omega_{ij})} \right],$$

which selects our N samples from differently-sized random subsets (of size M_i), and each sample ω_{ij} may have a unique source pdf q_{ij}.

While somewhat reducing notational complexity (e.g., in Equation 16), variation frequently occurs. ReSTIR resamples between different pixels. Target functions $\hat{p}_1$, $\hat{p}_2$, etc., all vary depending on which pixel gets reused. And resampling begins anew from $M = 0$ at disocclusions, which happens at different time across the screen.

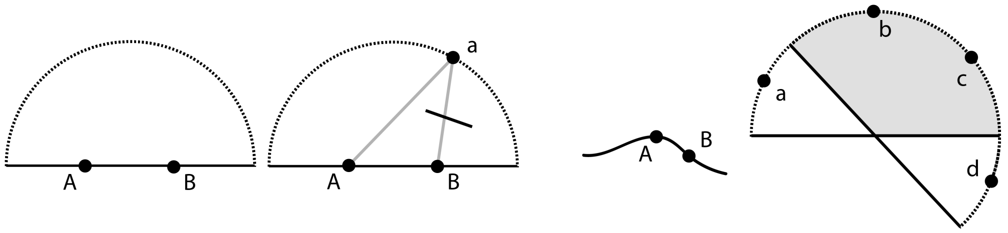


Figure 3: *Reusing samples between pixels A and B causes bias if target functions $\hat{p}$ have zeros in different locations. (Left) No bias when both points lie on a plane illuminated by an unoccluded emissive hemisphere. (Left center) Adding visibility into $\hat{p}$ causes bias when introducing occlusions that vary between pixels ($\hat{p}_A(a) > 0$ but $\hat{p}_B(a) = 0$). (Right center) Reuse between pixels with varied surface normals introduces bias, shown (right), where $\hat{p}_A(d) = 0$ and $\hat{p}_B(a) = 0$, causing bias when reusing B at A (or vice versa).*

4. On Bias in Spatiotemporal Reuse

Intuitively, RIS and ReSTIR provide an approximately-perfect importance sampler we can apply instead of truly perfect sampling. But remember, it is only *approximately*-perfect. As described in Section 3.1.1, we actually generate pdfs $p(\omega)$ stochastically via a randomized algorithm (e.g., see [MR95]). Not only do samples randomly vary between frames, so do the pdfs used to select them!

Section 3.1.1 argues the expected value $\mathbb{E}[W(\omega,z)]$ indeed converges to the desired value of $1/p(\omega_z)$. But this is *only* true when the domains of our samples are identical. In ReSTIR, we explicitly reuse pixels whose target functions $\hat{p}_i$ or source pdfs q_{ij} might vary from ours; this expands the diversity of our sample pool.

But this diverse sample reuse potentially introduces bias. Bitterli et al. [BWP*20] show that, for Equation 5, if source pdfs $q_{ij}(\omega)$ arbitrarily vary per sample, then:

$$\mathbb{E}[W(\omega,z)] = \frac{1}{p(\omega_z)}\frac{|Z(\omega_z)|}{M}, \tag{18}$$

where $|Z(\omega_z)|$ counts the source pdfs q_j where $q_j(\omega_z) > 0$, i.e.,

$$Z(\omega) = \{j \mid 1 \leq j \leq M \text{ and } q_j(\omega) > 0\}. \tag{19}$$

Basically, if not all pdfs *can generate* sample ω (i.e., $q_j(\omega) = 0$ for some j), selecting ω introduces bias. However, Equation 18 suggests debiasing is straightforward, as then:

$$\mathbb{E}\left[\frac{M}{|Z(\omega_z)|}W(\omega,z)\right] = \frac{1}{p(\omega_z)}, \tag{20}$$

leading to an unbiased variant of the RIS estimator in Equation 5:

$$F \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{\hat{p}(\omega_i)}\frac{1}{|Z(\omega_i)|}\sum_{j=1}^{M}\frac{\hat{p}(\omega_{ij})}{q_{ij}(\omega_{ij})}\right], \tag{21}$$

and an unbiased ReSTIR estimator updating Equation 16:

$$\frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{\hat{p}_0(\omega_i)}\frac{1}{|Z_i|}\sum_{j=1}^{M_0}\left[\frac{\hat{p}_0(\omega_{ij})}{\hat{p}_1(\omega_{ij})}\frac{1}{|Z_{ij}|}\sum_{k=1}^{M_1}\left[\frac{\hat{p}_1(\omega_{ijk})}{\hat{p}_2(\omega_{ijk})}\cdots\right]\right]\right], \tag{22}$$

where $|Z_i|$ counts the number of $\hat{p}_1$ pdfs that are non-zero at ω_i and $|Z_{ij}|$ counts how many $\hat{p}_2$ pdfs are non-zero at ω_{ij}, etc.

4.1. Intuitively Understanding Bias

While Equation 21 shows reuse can be unbiased, evaluating $|Z(\omega_i)|$ is costly. Knowing how bias arises helps determine when debiasing boosts fidelity enough to compensate for increased costs. Without costly $|Z|$ terms, the estimators guarantee unbiasedness, for:

- Importance sampling (Eq. 2), if $p(\omega) > 0$ when $f(\omega) > 0$.
- RIS (Eq. 5), if $\hat{p}(\omega) > 0$ and $q_j(\omega) > 0$ when $f(\omega) > 0$.
- ReSTIR (Eq. 16), if $\hat{p}_i(\omega) > 0$ and $q_j(\omega) > 0$ when $f(\omega) > 0$.

These simply restate conditions when all $|Z(\omega)|$ terms equal their corresponding M terms, cancelling the extra factor in Equation 18.

Figure 3 illustrates when these guarantees fail for RIS (and ReSTIR). Two points on a planar surface, illuminated by distant, unshadowed lights can reuse samples in an unbiased way. But introducing occlusions or sharing between different surface orientations lead to zero-valued probabilities. Specifically, bias arises if:

1. B's target function, $\hat{p}(\omega)$, has varying visibility (from A to B),
2. B's target function, $\hat{p}(\omega)$, does not cover the hemisphere at A, or
3. B's source pdf, $q(\omega)$, does not cover the hemisphere at A.

Enumerating our list is easy: consider where reflected light is non-zero at the current pixel but zero at reused neighbors. These introduce bias. Complex environments may add bias from other sources, e.g., directionally varying emission and BRDFs with zero-valued reflectance in the visible hemisphere.

All bias types are handled by the estimators in Equations 21 and 22, however the cost to compute $|Z(\omega)|$ can vary significantly. For instance, identifying zeros at samples below the horizon, caused by varying normals, requires a few dot products. But finding visibility changes requires shooting new rays.

Importantly, when debiasing via $|Z(\omega)|$ terms, we consider only one iteration at a time. While varied $\hat{p}_2$'s in Equation 22 can add bias when approximating $\hat{p}_1$, correcting gives an unbiased estimate for $\hat{p}_1$ at a particular pixel. These unbiased $\hat{p}_1$ pdfs can be (re)used to estimate $\hat{p}_0$. But by borrowing $\hat{p}_1$ from neighbors, we may introduce *new* differences to account for to get unbiased $\hat{p}_0$.

While debiasing considers only one iteration at a time, if we skip debiasing to reduce cost, it is important to understand that bias *iteratively compounds*. Care is required to avoid bias explosions.

4.2. Manifestation of Bias

Section 4.1 describes why bias occurs, but it may be unclear how bias manifests. Figure 4 shows a concrete example, reusing samples from pixels with varying normals. Bias arises in corner cases, when

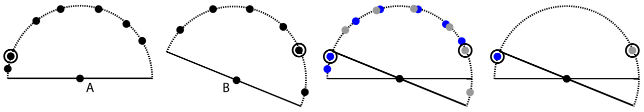


Figure 4: *(Left) Two adjacent pixels, A and B, have different normals. Both pixels use RIS, selecting one random candidate ($M_A=7$, $M_B=6$) to store for reuse. (Right center) When reusing B to improve sampling at A, we have $M_{A+B}=13$ effective candidates. (Right) But we only pick from the samples explicitly selected for reuse. Bias arises when incorrectly weighting the selected sample. If we pick the blue sample, it* could not *have been generated at B; we need to use an appropriate M when normalizing (use M_A if selecting the blue sample, M_{A+B} for the gray sample).*

selecting a sample that *cannot* be sampled by both pixels (e.g., the circled blue one). These cases require careful normalization.

Normalization essentially splits the integral into disjoint pieces. In Figure 4, the region covered by both hemispheres is integrated with M_{A+B} effective samples. The region covered by only A's hemisphere is integrated with only M_A effective samples. Intuitively, this is what the $|Z(\omega)|$ terms do in Equations 21 and 22.

Once we understand bias arises from the ignored $M/|Z(\omega)|$ term in Equation 20, it becomes easier to describe its appearance. Most commonly, bias gets introduced by always using all neighbor samples, without accounting for their target functions going to zero.

In Figure 4, such reuse assumes M_{A+B} effective samples, even in regions where only M_A contribute. For the RIS normalization term:

$$\frac{1}{M_i}\sum_{k=1}^{M_i}\left[\frac{\hat{p}_i(\omega_k)}{\hat{p}_{i+1}(\omega_k)}\cdots\right], \tag{23}$$

M_i ends up being too large (i.e., M_{A+B} rather than M_A). This reduces the normalization term, causing *darkening*. Darkening worsens when reusing more neighbors with partially relevant pdfs. Thus, bias increases nearing large depth, normal, or shadow boundaries.

4.3. Heuristics for Reducing Bias

Bitterli et al. [BWP*20] propose applying simple heuristics to help reduce bias without computing $|Z(\omega)|$. Mathematically, this introduces a heuristic function H to the RIS estimator from Equation 5:

$$F \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_i)}{\hat{p}(\omega_i)}\frac{1}{\mathbb{H}}\sum_{j=1}^{M}\left(H(i,j)\frac{\hat{p}(\omega_{ij})}{q(\omega_{ij})}\right)\right], \tag{24}$$

where $H(i,j) \in \{0,1\}$, based on the heuristic result. $\mathbb{H}$ replaces M, and counts samples where the heuristic passes, i.e., $\mathbb{H}=\sum H(i,j)$. Heuristics work like edge-stopping functions in filtering. They stop bias crossing boundaries, keeping per-neighbor bias below a threshold (controlled by any heuristic parameters).

Bitterli et al. [BWP*20] use standard heuristics, checking if normals at i and j point in similar directions (i.e., $\langle\vec{n}_i\cdot\vec{n}_j\rangle>\tau$) and if pixel depths are similar (i.e., $\text{abs}[(z_i-z_j)/z_i]<\varepsilon$).

Figure 5 shows how heuristics help. In spatiotemporal reuse, bias increases as target function overlap decreases. For smaller overlaps, reuse greatly increases bias but adds few new samples. Ignoring poorly matching neighbors reduces bias with little quality impact.

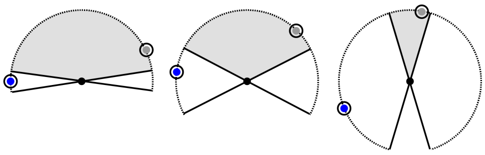


Figure 5: *When reusing from pixels with different normals, sample relevance, usefulness, and bias varies highly. (Left) Given similar normals, we have a low chance to select (blue) samples introducing bias. (Center) With more variation, we get a benefit from reuse (gray region) but add more bias. (Right) If normals point in opposite directions, reuse introduces a lot of bias for little improvement.*

While Figures 4 and 5 show reuse over varying normals, changing visibility also defines regions where neighbor target functions $\hat{p}$ go to zero. Reuse over large depth differences often changes sample visibility; a depth heuristic helps reject these samples.

Other heuristics are possible, but heuristics examining sample weights or pdfs $\hat{p}(\omega_{ij})$ or $q(\omega_{ij})$ generally add conditional probabilities into $H(i,j)$ in Equation 24. Our resampling theory does not yet handle such conditional probabilities, so these heuristics tend to overzealously discard neighbors. This shrinks M_i in Equation 23, causing a *lightening* bias.

4.4. Multiple Importance Sampling For Sample Reuse

Equation 21 correctly normalizes a sample based on which neighbors contributed to its selection. The heuristics in Section 4.3 discard categorically, using knowledge about samples likely to be beneficial. But both approaches make binary decisions: either a neighbor is reused or discarded. What if some neighbors provide better estimates than others? Could we weight neighbors differently?

Multiple importance sampling (MIS) [VG95] allows exactly this sort of reweighting. In resampling, each neighbor is a different estimator we use to sample our current pixel. Bitterli et al. [BWP*20] show a straightforward application of MIS by redefining the weight function $W(\omega,z)$ in Equation 6 with an arbitrary weight $m(\omega_z)$:

$$W(\omega,z)=\frac{1}{\hat{p}(\omega_z)}\left[m(\omega_z)\sum_{i=1}^{M}w_i(\omega_i)\right]. \tag{25}$$

Rederiving the expected value with this new $W(\omega,z)$ gives:

$$\mathbb{E}\left[W(\omega,z)\right]=\frac{1}{p(\omega_z)}\sum_{i\in Z(\omega_z)}m(\omega_i), \tag{26}$$

allowing unbiased reuse whenever we define $m(\omega_z)$ such that sum $\sum_{i\in Z(\omega)}m(\omega_i)=1$. This is trivially true for $m(\omega_z)=1/|Z(\omega_z)|$, as used in Equation 21. But we can replace this trivial $m(\omega_z)$ with a standard balance heuristic:

$$m(\omega_z)=\frac{q_z(\omega_z)}{\sum_{i=1}^{M}q_i(\omega_z)}, \tag{27}$$

which, for clarity, may also be written:

$$m(\omega_z)=\frac{q_z(\omega_z)}{\sum_{i\in Z(\omega_z)}q_i(\omega_z)}, \tag{28}$$

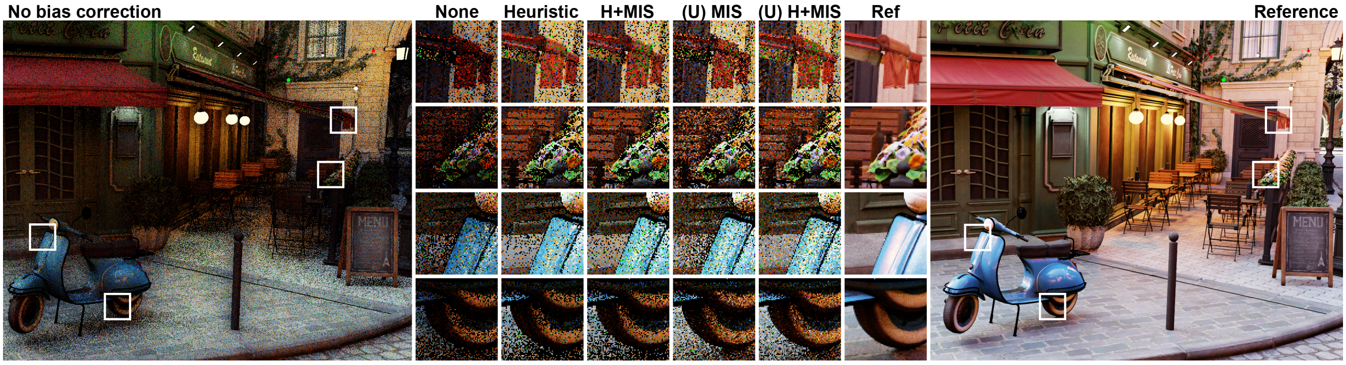


Figure 6: *Compare (left) a ReSTIR rendering without debiasing to a (right) converged, unbiased ground truth. We also compare various insets (center) using varying debiasing settings, left to right: no debiasing; still biased heuristic-based rejection of neighbors [BWP*20]; heuristics* plus *MIS to remove bias due to varying normals; unbiased using MIS [BWP*20]; unbiased using both MIS and heuristics to reject likely bad neighbors; and converged ground truth. H+MIS improves the highlight on the Vespa body and reduces noise on the flowers. (U) H+MIS helps eliminate noise at discontinuities that using MIS, alone, to debias tends to introduce.*

as pdfs $q_i(\omega_z)$ may be zero during spatiotemporal reuse. Modifying Equation 5 gives an unbiased, MIS variant of our RIS estimator:

$$F \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_{z_i})}{\hat{p}(\omega_{z_i})} m(\omega_{z_i}) \sum_{j=1}^{M}\frac{\hat{p}(\omega_{ij})}{q_j(\omega_{ij})}\right], \tag{29}$$

and inserting our balance heuristic from Equation 27:

$$F \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_{z_i})}{\hat{p}(\omega_{z_i})} \frac{q_{z_i}(\omega_{z_i})}{\sum_{k=1}^{M} q_k(\omega_{z_i})} \sum_{j=1}^{M}\frac{\hat{p}(\omega_{ij})}{q_j(\omega_{ij})}\right], \tag{30}$$

where, for clarity of notation, we've defined ω_{z_i} as the sample in the set $\{\omega_{i1},\ldots,\omega_{iM}\}$ selected for reuse by RIS and q_{z_i} is that sample's corresponding source pdf q_j.

4.5. Interesting MIS Observations

Bitterli et al. [BWP*20] derive that $m(\omega_i)$ summing to unity gives unbiasedness (in Equation 26), and then directly inferred we could reuse standard balance heuristics for MIS. However, we can make more mundane, but still useful, observations.

For instance, if $M = 4$ (i.e., reusing from four neighbors), we can define $m(\omega_z) = 0.25$. Or more usefully, if the four neighbors are in a 2×2 block, we can define $m(\omega_z)$ as a set of bilinear weights.

Interpolating reservoirs makes no sense, as they contain discrete samples without a well-defined interpolation operator. But with Equation 26, we can incorporate bilinear weights directly in function $m(\omega_z)$ to similar effect. During temporal reuse, such weights provide one mechanism to proportionately reuse from the four nearest temporal neighbors of a backprojected pixel.

4.6. Combining Heuristics and Multiple Importance Sampling

Two issues remain with multiple importance sampling in either Equation 30 or iterative variants (akin to Equation 22):

- MIS can increase variance when using poor pdfs, and
- One may want to reduce bias, without paying to remove it all.

Essentially, we want the heuristics and MIS in Sections 4.3 and 4.4 to not be mutually exclusive. Perhaps developers can afford extra dot products for MIS to remove bias from varying surface normals, but the extra rays to remove visibility bias are too costly.

And while MIS theoretically combines two extremely different estimators correctly (e.g., Figure 5, right), perhaps it is a bad idea practically. We can use heuristics to avoid passing known-bad estimators into MIS, reducing variance as a result.

Fortunately, the heuristic estimator in Equation 24 resembles the MIS estimator in Equation 29. In fact, heuristics can be thought of as a form of MIS weighting, i.e., we can define set $\mathcal{Z}(i,\omega)$:

$$\mathcal{Z}(i,\omega) = \{j \mid 1 \le j \le M \text{ and } q_j(\omega) > 0 \text{ and } H(i,j) = 1\}, \tag{31}$$

allowing use of both MIS and heuristics, akin to Equation 21:

$$F \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_{z_i})}{\hat{p}(\omega_{z_i})} \frac{1}{|\mathcal{Z}(i,\omega)|} \sum_{j\in\mathcal{Z}(i,\omega)}\frac{\hat{p}(\omega_{ij})}{q_j(\omega_{ij})}\right], \tag{32}$$

and to Equation 30:

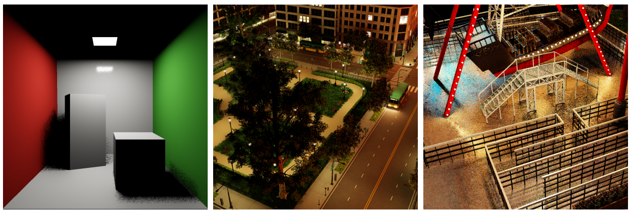
$$F \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{f(\omega_{z_i})}{\hat{p}(\omega_{z_i})} \frac{q_{z_i}(\omega_{z_i})}{\sum_{k\in\mathcal{Z}(i,\omega)} q_k(\omega_{z_i})} \sum_{j\in\mathcal{Z}(i,\omega)}\frac{\hat{p}(\omega_{ij})}{q_j(\omega_{ij})}\right]. \tag{33}$$

These complete our options, allowing us to apply RIS and ReSTIR in a biased form, with heuristics, with MIS to eliminate bias, or with both MIS and heuristics.

With Equation 33, the difference between our highest quality biased and unbiased modes is simply whether we shoot shadow rays in the MIS weight $q_{z_i}(\omega_{z_i})/\sum q_k(\omega_{z_i})$. Our unbiased variant shoots one ray per neighbor (i.e., for each q_k) while the biased variant skips tracing rays, assuming each query is unoccluded.

In both variants, MIS corrects bias due to reuse between neighbors with varied normals and heuristics discard neighbors likely to introduce both bias and noise. The only difference is whether our results have bias due to spatiotemporal visibility variations.

A key takeaway, illustrated by Figure 5, is neighbors introducing large biases also introduce significant noise. Heuristics help discard such neighbors, and help even when using MIS. Figure 6 shows the effect of various combinations of MIS and our heuristics.



(a) *Cornell Box*
0.5 ms / 0.5 ms
1.4 ms / 1.1 ms
2 emissive tris

(b) *Emerald Square*
1.8 ms / 0.8 ms
6.1 ms / 2.0 ms
89k emissive tris

(c) *Amusement Park*
18.3 ms / 0.8 ms
98.7 ms / 2.1 ms
3.4M emissive tris

Figure 7: *Initial light candidate generation costs for varying complexity scenes, using Bitterli et al. [BWP*20] and our presampling that pulls incoherent memory accesses out of the inner render loop. Times on (top) RTX 3090 and (bottom) laptop RTX 2080 Max-Q.*

5. Reshaping Resampling for Improved Memory Coherency

Given the above theoretical derivations of RIS and ReSTIR estimators and a deeper understanding of how bias occurs during resampling, we now shift to focus on practical problems like removing bottlenecks and redesigning the algorithm for optimal performance.

An appealing aspect of ReSTIR is a constant-time computational complexity. Each pixel selects a small set of initial candidates from source pdf $p(\omega)$ and combines them with a few reservoirs from the current and prior frame. Bitterli et al. [BWP*20] use 32 candidates, 1 temporal reservoir, and 2 rounds of spatial reuse with 5 neighbors each. Reservoirs store 4 samples, so pixels touch exactly 72 lights.

But testing ReSTIR, we saw *non-constant* performance. In fact, profiled component costs varied up to 20× between scenes. Obviously, fixed computation and memory usage (i.e., 72 lookups) with varying performance implies different caching behavior. This limits its adoption, as artists must manage cache-based performance cliffs that change between GPU vendors and generations.

This cache thrashing arises due to independent sampling of light candidates. In the Cornell Box (see Figure 7), pixels pick 32 samples on one light, using memory coherently. In the Amusement Park, pixels randomly pick 32 of the 3 million lights; this randomly walks memory, often going to DRAM due to cache misses.

Interestingly, stratified sampling helps improve performance. For example, with jittered sampling [Coo86] threads coherently walk through subsets of the light list, reducing potential for cache thrash. Still, this just reduces intra-warp incoherency; separate warps continue to compete for cache lines. But Section 3.1.3 shows RIS stratifies over random subsets. Can that improve coherence?

5.1. Reshaping Computation Using Degenerate RIS Steps

Remember degenerate RIS estimators offer two benefits. They provide randomized stratification and decompose sampling into two steps. A key observation: the decomposed sampling steps can occur at different times. The first step, creating M-sized light subsets, contains most incoherent memory accesses; removing it from the inner render loop improves performance significantly (see Figure 7).

Since random memory lookups cause incoherence, presampling or prerandomizing the light lists reduces incoherency if it allows using smaller light subsets that remain in cache during the inner render loop. Figure 8 compares pseudocode for the per-pixel sampling from Bitterli et al. [BWP*20] with two prerandomized variants.

The simplest factorization (Figure 8b) selects a subset S of the scene lights L once per frame and pixels sample from S to select their ($N = 32$) light candidates. If $S = L$, this reverts to independent per-pixel sampling. If $|S| = N$, pixels all use the same candidates. In any case, this gives *unstratified* RIS, per Equation 13, as neighbor pixels all select from the same candidate pool.

A more complex approach precomputes light subsets S_j for each pixel. When rendering, each pixel selects candidates from its corresponding subset. This gives fully stratified RIS, per Equation 5. But this fails to improve performance; it has the same incoherency, moved from the per-pixel sampling into a per frame preprocess.

The power of decomposed sampling is these methods are ends of a continuum; we need not use either one precomputed set per frame or one per pixel. If $|P|$ is the number of pixels, we can precompute $\mathbb{S}$ light subsets S_j, where $1 \ll \mathbb{S} \ll |P|$ (see Figure 8c). This leads to *partially* stratified RIS, mixing Equations 5 and 13. This reduces incoherency (as $\mathbb{S} \ll |P|$) and removes it from the inner loop.

5.2. Coherent Initial Candidate Generation

Given that principle, let us define an algorithm to select initial, per-pixel light candidates. Key parameters are the number of light subsets $\mathbb{S}$ generated each frame, each subset's size $|S_j|$, how a pixel selects a light subset S_j to sample, and the pdfs used when precomputing and sampling from S_j.

The pdfs fall out of Equation 10. When presampling lights S_j from global light list L, we sample L using source pdf $q(\omega)$, i.e., use the same pdf Bitterli et al. [BWP*20] used for initial candidate generation. When selecting per-pixel candidates from S_j, we select *uniformly*, as subsets S_j are already distributed according to $q(\omega)$.

To maximize coherence within warps, tiles of pixels all sample the same subset S_j. We found 8 × 8 pixel tiles maximize coherence without introducing noticeable correlations not masked by spatiotemporal reuse.

This gives our new algorithm for initial candidate selection:

```
input:  scene light list $L = \{l_i\}$
output: 32 light candidates $C$ per pixel
once per frame:
   for $j \in [1 \ldots \mathbb{S}]$:
      for $k \in [1 \ldots |S_j|]$
         $S_j$[k] = sample($L$, $q()$)

for each pixel $(x,y)$:
    $S$ = selectRandomSubsetForTile($x,y,\{S_j\}$)
    for $i \in [1 \ldots 32]$
       $C$[i] = sample($S$, uniformly)
```

We empirically found $\mathbb{S} = 128$ and $|S_j| = 1024$ minimizes cost without introducing artifacts, across a wide set of scenes. For smaller scenes, these values are overkill but launch overheads dominate the

```
input $L = \{l_i\}$
for each pixel:
  select N lights from L
```

(a) *Independent per-pixel samples*

```
input $L = \{l_i\}$
once per frame:
  select M lights from L
  S = {set of M lights}

for each pixel:
  select N lights from S
```

(b) *Single preselected subset*

```
input $L = \{l_i\}$
once per frame:
  for j $\in [1 \ldots \mathbb{S}]$:
    select M lights from L
    $S_j$ = {set of M lights}

for each pixel:
  select light set $S_j$
  select N lights from $S_j$
```

(c) *Multiple preselected subsets*

Figure 8: *Three light candidate selection methods. (a) Independently sample lights L per pixel, causing cache thrashing in many-light scenes. (b) Factor candidate selection via degenerate RIS, using one light subset S. This gives unstratified RIS as in Equation 13. (c) Factor candidate sampling, but use multiple light subsets S_j to allow (partially) stratified RIS as in Equation 5.*

cost of precomputation, so lower values have no benefit. For large scenes, the premise is the total working set, $\mathbb{S} \times |S_j|$, fits entirely in cache and each subset S_j fits in a (few) cache lines.

5.3. Other Implications of Presampling Lights

Beyond improved caching, prerandomizing lights in subsets S_j has other benefits. In particular, highly optimizing sampling performance is less important. With 32 initial candidates per pixel across a 1920×1080 image, Bitterli et al. [BWP*20] incoherently sample 64 million lights per frame; with $\mathbb{S} = 128$ and $|S_j| = 1024$, we only need 128,000 incoherent lookups, invariant of resolution.

This gives flexibility to employ different sampling algorithms. For example, prerandomization allows easy mixing of emissive primitives: triangles, spheres, capsules, patches, etc. Each has unique sampling code, so mixing them per-pixel adds both data and execution divergence. But degenerate RIS pulls this divergence out of the inner, per-pixel loop. Each frame starts by selecting new samples per primitive type: a set S_t for samples on triangles, S_s for spheres, S_c capsules, etc. Then we create $\mathbb{S}$ per-frame light sets S_j by sampling the combined set $\{S_t, S_s, S_c, \ldots\}$ with an appropriate pdf, e.g., proportional to total emission from each primitive type.

Additionally, Bitterli et al. [BWP*20] use alias tables [Wal77] to pick lights proportional to power. Alias tables allow cheap $O(1)$ sampling, which is vital with 64 million samples a frame. But optimal table builds cost $O(N)$ [Vos91] and do not parallelize trivially on GPUs. In moderate complexity scenes CPU builders take under 0.5 ms, but build costs for large environment maps exceed 200 ms.

For dynamic light probes, we can substitute alternative approaches. Binder and Keller [BK19] propose one option, quickly building a radix forest to sample via the cutpoint method [FM84]. The cutpoint method has $O(\log N)$ worst-case lookups, but searching via a radix forest has average case complexity of $O(1)$.

Instead, we perform cdf inversion [PJH16] hierarchically using a mipmap chain, leveraging existing GPU strengths. We create a texture storing luminance (times solid angle) for each environment map texel, and build a mip chain. When sampling, we start at the second coarsest mip level, loading all 4 texels. We create a 4-element cdf, and sample one (with probability p_{n-1}). We load our sample's four children, in the next mip level, and again sample from the new 4-element cdf (with probability p_i). On reaching the finest level, we have a random texel sampled with pdf $p_{n-1} p_{n-2} \cdots p_0$. Building our hierarchy costs around 60 μs for a 2048×1024 light probe. Lookups are $O(\log N)$, but occur outside the inner loop, thanks to presampling. This allows truly dynamic textured emissives, e.g., light probes streamed live from a 360° camera.

6. Improved Efficiency for Sampling

In Section 5, we addressed the aspect of ReSTIR that scaled poorly with increased complexity. But deeper investigation found other hidden inefficiencies and redundancies that we address, below, to reduce cost and increase flexibility.

6.1. Ideal Ray Budget

Bitterli et al. [BWP*20] propose using 5 rays per pixel. On an absolute scale, this is exceedingly cheap to dynamically shadow millions of lights. But developers only budget $\frac{1}{4}$, $\frac{1}{2}$, or 1 ray per pixel today, depending on quality settings. Thus, scaling ray counts is vital for wide adoption; asking 20× current budgets is a hard sell.

The variance analysis in Section 3.4 motivates this five ray budget; ReSTIR largely skips reusing visibility, meaning N, the per-pixel shade count, controls visibility variance. Talbot et al. [TCE05] propose balancing N and M via a complex cost analysis. On high-end GPUs, $N = 4$ shadow rays cost milliseconds and seem a small price for reduced visibility variance.

But reevaluating this is vital to scale down cost. To this end, we analyzed the impact of changing $N = 4$ to 1, with surprising results. More rays reduce noise. But the difference is fairly small and (arguably) not cost effective, as higher frame rates also improve perceptual quality due to flicker fusion in the visual system [ANS*19].

What limits the variance improvement from increased ray count? ReSTIR leverages correlations to improve neighbor sampling quality. But correlations become problematic if shading multiple samples; with 4 light samples per pixel, ReSTIR likely picks the same sample multiple times.

Figure 9 visualizes duplicate samples in the Amusement Park. Across scenes, when shading four samples per pixel, between $1/2$ and $2/3$ of pixels include duplicates. Perhaps 20% of pixels pick

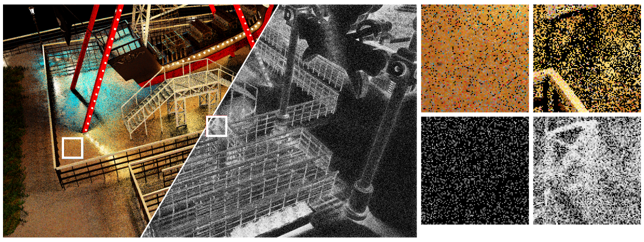


Figure 9: *ReSTIR improves quality via correlated sampling. But correlations may cause duplicate selections in per-pixel light samples. (Left) The Amusement Park rendered with ReSTIR. (Right) A heat map visualizing duplicate samples in per-pixel reservoirs with four elements (e.g., [BWP*20]). Black means no duplicates; white means all four light samples are identical. Duplicates occur most near shadows—exactly where we hoped higher sample counts would reduce variance.*

the same light sample four times. Worse, duplicates occur most frequently near shadows; this is exactly where we hoped to reduce variance by increasing sample count!

Fortunately, researchers and engineers frequently filter visibility with very few samples [JWPJ16, MML12, SKW*17, EHDR11]. Given N controls visibility noise, increasing N is costly and somewhat ineffective, and filtering this noise is known feasible with one sample per pixel, we decided $N = 4$ is excessive and wasteful.

Throughout the rest of this paper, we fix $N = 1$. This reduces the ray count for ReSTIR from five to two rays per pixel. Quality decreases somewhat in shadowed regions, especially in static images. But our other improvements help compensate for this quality loss.

6.2. Removing Per-frame Global Synchronization

Consider the pipeline in Figure 2. This structure relies on a global barrier; before spatial reuse, neighbors must select candidates and reuse temporally. This forces certain implementation choices, including the use of multiple kernels.

In theory, removing synchronization allows a single kernel variant of ReSTIR, containing all steps from Figure 2. This opens more opportunities for optimization, amortization, and reducing bandwidth for intermediate data. In practice, hardware constraints may make separate components more efficient (e.g., one kernel for ray tracing, another for other computations). Without a forced synchronization, testing these reformulations becomes significantly easier.

ReSTIR gets its *spatiotemporal* name by reusing both spatially and temporally. But we need not avoid true spatiotemporal reuse, i.e., reusing neighbors in the temporal buffer. In fact, replacing all spatial samples with spatiotemporal samples trivially removes our synchronization; all reused samples come from the last frame.

Generally, this change is imperceptible. However, fast-moving geometry causes large disocclusions where the prior frame poorly approximates the current frame. Here, using spatiotemporal samples essentially delays convergence by one frame—current candidates only get shared next frame. Where such lag is objectionable, synchronizing to reuse current samples may still be desirable.

6.3. Importance of Temporal versus Spatial Reuse

As ReSTIR reuses both spatial and temporal samples, understanding their relative importance helps us prioritize computations in a limited budget. Both types of reuse have advantages that compensate for issues in the other; but they are not equally important.

Figure 10 compares the Paris Opera House with various types of reuse. Basic RIS tests a few candidate lights per pixel but is woefully inadequate in complex environments. Spatial reuse multiplies the effective sample count by a constant (e.g., 5× given five taps); this noticeably improves quality but remains far from the reference. Temporal reuse accumulates from all prior frames, increasing effective sample count much more significantly.

But two problems emerge when only reusing temporally. Motion reveals disoccluded regions with no temporal history. And backprojection identifies nearest neighbor temporal samples, as reservoirs contain discrete samples without a well-defined interpolation operator. During motion, this adds noticeable correlation between adjacent pixels. Such artifacts are hard to see in images, e.g., Figure 10. Combining spatial and temporal samples helps address these issues.

Adding even one spatial sample eliminates issues due to discrete backprojection. Here, spatial reuse acts as a stochastic filter, choosing between the correlated temporal sample and a randomized spatial sample. Each pixel selects different neighbors, so spatial reuse stochastically dithers out temporal correlations.

To determine the ideal spatial sample count, the key is how spatial samples reduce noise from disocclusions. By definition, disocclusions mean no temporal reuse. Quality depends *solely* on spatial samples and improvement occurs linearly with more samples. Reaching quality comparable to temporal reuse requires hundreds of spatial samples, which is infeasible in constrained budgets.

6.4. Target Spatial Sample Count at Disocclusions

The key question is: at game framerates, how many spatial samples are needed for unnoticeable disocclusions? This is complicated—it depends on user taste, available budget, framerate, and the denoiser. At very low framerates, fast motion causes large disocclusions that remain for a sizable time. Bitterli et al.'s [BWP*20] prototype has similar characteristics. In such cases their parameters are quite reasonable: two spatial reuse passes with five spatial taps each.

But at 60 Hz, spatiotemporal reuse quickly fills disocclusions. Figure 11 shows quick horizontal strafing in a 5120×1440 render. With only temporal reuse, disocclusions persist for over 20 frames. With one spatial and one temporal sample, significant variance lasts for 3 or 4 frames, with a longer trail of elevated noise that a denoiser easily fixes. Using one temporal and five spatial samples increases noise for 2 frames, after which it largely reconverges to average variance across the frame.

This comes from the exponential growth in spatiotemporal reuse: combining one spatial and temporal tap over 4 frames borrows from 2^4 pixels; using five spatial taps reuses 6^4 pixels. In both cases, spatial reuse potentially finds neighbors with valid temporal histories, improving quality further.

Seeing disocclusion noise similar to Figure 11 across various

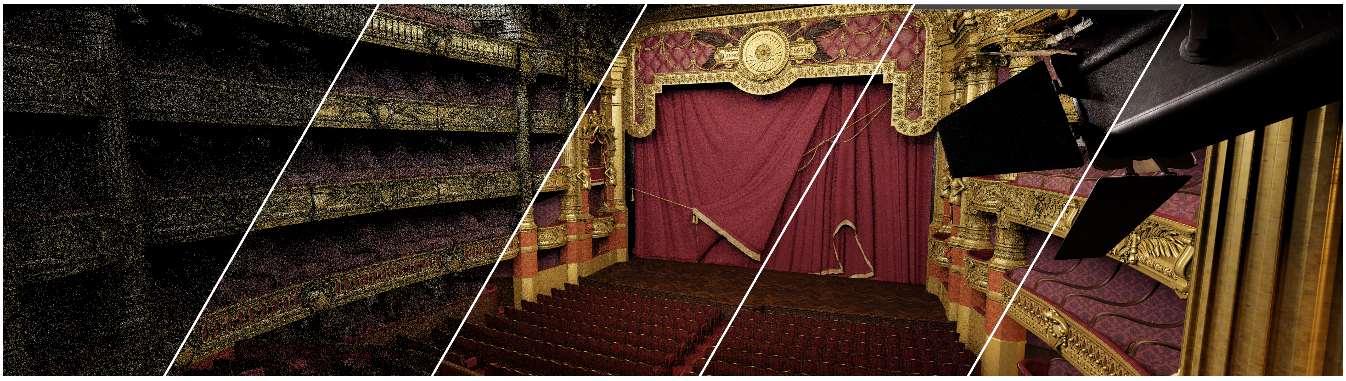


Figure 10: *The Paris Opera House with over 150 million polygons and 500,000 emissive triangles rendered at* 5120 × 1440*, using (a) naive RIS lighting in 8.8 ms, (b) spatial reuse in 12.4 ms, (c) temporal reuse in 13.4 ms, (d) spatiotemporal reuse in 13.0 ms, and (e) a reference. Times include initial sampling, reuse, two shadow rays, and final shading on a RTX 3090. Model courtesy TurboSquid and ©GoldSmooth.*

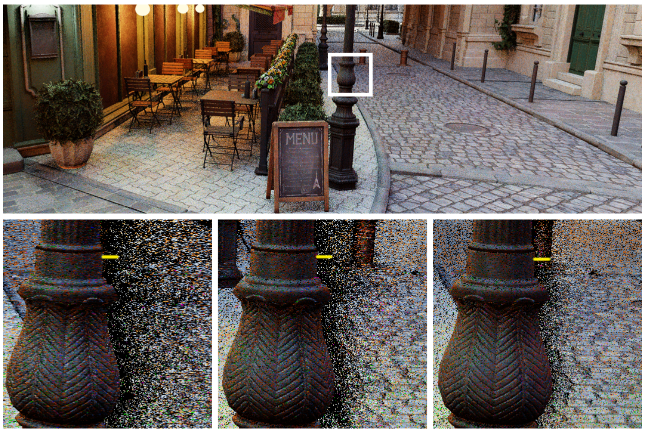


Figure 11: *The Amazon Bistro (top) with insets showing temporal disocclusions under horizontal motion, using three settings. (Left) Temporal reuse only, (center) temporal reuse plus one spatial sample, and (right) temporal reuse plus five spatial samples. The yellow lines show the width of the disocclusion in one frame.*

scenes, we found that with modern denoisers [NVI20] a single spatial tap is frequently sufficient for most disocclusions. Developers targeting higher quality or 30 Hz framerates may want 2 or 3 spatiotemporal samples. When using more than one, spatiotemporal samples should be adaptively used in just disoccluded regions; this targets the additional cost exactly where these computations actually impact quality.

7. Decoupling Shading and Reuse

Another observation on ReSTIR: it couples shading and reuse. The samples shaded in frame N are also reused when resampling frame $N+1$. However, the *goals* of shading and reuse are quite different.

For shading, we want best quality for immediate display. Resampling should forward samples to optimize quality in *future* frames. Both goals may not be optimized via the same samples. In fact, we found explicitly decoupling shading and reuse opens new opportunities to improve current frame render quality.

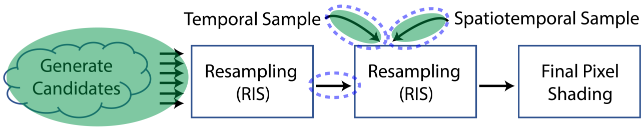


Figure 12: *Our pipeline after improvements from Section 6. Note each pixel examines $M+2$ samples (highlighted in green): M candidates, 1 temporal and 1 spatiotemporal sample. But just one gets shaded and reused. This seems wasteful, but we cannot afford $M+2$ shadow rays per pixel to shade them all. But RIS cuts our pool of samples. We know one of the three dashed samples gets selected for shading and reuse. Perhaps we could shade all three?*

How? Resampling selects one sample to reuse and shade from many, according to a more optimal distribution, i.e., resampling selectively *discards* samples (rejection sampling). But why discard samples *before* shading? Why not leverage them all to improve shading? Decoupling allows shading and resampling different subsets of the samples we examine each frame.

7.1. What Samples Are Examined?

Figure 12 shows the simpler ReSTIR pipeline after Section 6. With M candidates, pixels each touch exactly $M+2$ light samples. After multiple resampling steps, $M+1$ get discarded, with just one sample shaded and reused in future frames.

Why not shade all $M+2$ samples but only reuse one temporally? The problem is shadow rays. Shading $M+2$ samples requires $M+2$ shadow rays, and in Section 6.1 we assert even 5 rays is too many.

But we can group samples multiple ways, thanks to ReSTIR's iterative nature. Our per-pixel candidates undergo resampling, leaving three inputs to the second resampling. ReSTIR chooses one of these three samples for shading and reuse.

Why not shade all three? This requires more shadow rays—three

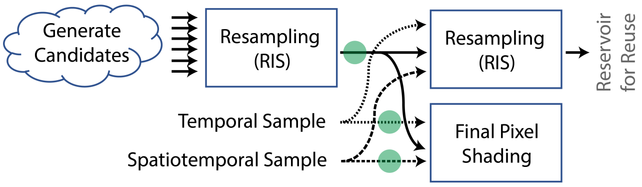


Figure 13: *Our decoupled pipeline separately evaluates shading and resampling for candidate, temporal, and spatiotemporal samples. We reuse only one next frame but shade all three this frame. Green dots represent shadow queries. We show a biased pipeline; unbiased reuse of temporal sample visibility requires additional rays for the MIS terms in Section 4.6.*

per pixel rather than two. But it also triples the shaded sample count, increasing image quality. That might be a good tradeoff.

Figure 13 shows our decoupled ReSTIR pipeline. Each of the three inputs to our shading pass traces a shadow ray. If using multiple spatiotemporal taps, we can first select just one to pass to shading and reuse (as in the additional RIS step used for candidates).

Note, just the candidate shadow ray gets fed forward, as reusing visibility for spatiotemporal samples either introduces compounding bias (Section 4.1) or requires additional shadow rays for multiple importance scattering (Section 4.6) to maintain unbiasedness.

During shading, we weigh our three samples' contributions with the probabilities used for resampling. If a sample has a 95% chance of being reused, it contributes 95% of pixel color. This avoids introducing too much noise from the new, low-probability candidates added each frame. While Figure 13 depicts shading and reuse as logically separate, we run them simultaneously in a single kernel.

7.2. Reusing Visibility in Resampling

In this decoupled shading pipeline, an important change is visibility rays are now tied to explicit samples, rather than to pipeline stages. This allows reasoning about the samples, and avoids any chance for duplicate shadow queries in a pixel.

With explicit bubbles to highlight shadow rays, Figure 13 helps clarify visibility reuse in ReSTIR. If (spatio-)temporal sample visibility gets piped into resampling, it introduces potential zeros in $\hat{p}$ for future frames (i.e., darkening bias from Section 4.2). This requires additional rays to remove, via the weights in Equation 33.

Figure 15 shows different types of biased visibility reuse. Without shooting more rays to compute MIS weights that remove visibility bias, we found the pipeline in Figure 13 optimal. It biasedly reuses visibility for selected per-pixel candidates, as this provides significant quality gains (e.g., Figure 15f versus 15b). Other shadow rays only impact shading, but these queries are important for local contact shadows (e.g., Figure 15f versus 15c).

7.3. Cheaper Visibility Reuse

Sadly, the pipeline in Figure 13 requires three rays per pixel, rather than the two rays proposed in Section 6.4. However, with visibility queries directly corresponding to specific samples, we can consider cheaper alternatives for determining visibility (see Figure 14).

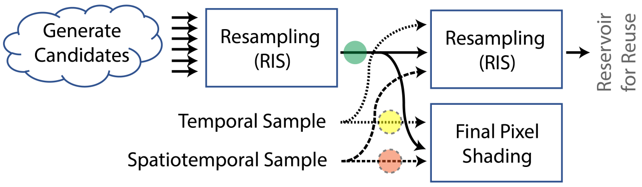


Figure 14: *As (spatio-)temporal visibility queries only affect shading in our decoupled pipeline, we can consider cheaper, alternate methods to compute them. For both (yellow) temporal and (orange) spatiotemporal samples, presumably we knew visibility last frame when shading. Can we reuse that visibility?*

For instance, frames come from a sequence of renderings. While we do not reuse visibility as part of resampling, we did previously *compute* visibility to shade temporal samples. Why not reuse this visibility to shade again?

For our temporal sample (yellow dot in Figure 14), modulo scene animation, visibility has not changed; reuse easily eliminates one shadow ray. Under animation, reusing this visibility adds a one-frame shadow lag in one of the three samples shaded per pixel. But we were unable to perceive this lag to show here.

Similar reuse for our spatiotemporal sample (orange dot in Figure 14) is less straightforward. Clearly, visibility may vary between the current pixel and its neighbor, so reuse adds additional bias. How does this bias manifest? For static images, hard-coding spatiotemporal visibility (orange dot) to one gives the result in Figure 15c. Tracing a ray gives the result in Figure 15f.

This shows that skipping the spatiotemporal visibility ray significantly reduces quality; but perhaps we can trace the ray less often? For instance, the further away a neighbor is from the current pixel, the more likely its visibility will vary. Perhaps we can reuse visibility for neighbors within some user-defined threshold.

Figure 16 shows the effect of this parameter. We reuse spatiotemporal visibility when resampling from nearby neighbors; we shoot visibility rays, as before, for more distant neighbors. This provides a parameter to trade quality for performance. As neighbor selection is randomized in ReSTIR, this parameter stochastically interpolates between endpoints (i.e., never and always reusing visibility).

To summarize, our decoupled shading pipeline provides flexibility for computing visibility queries in Figure 14. We always trace a candidate ray (green dot), always reuse visibility for our temporal sample (yellow dot), and parameterize reuse for our spatiotemporal sample (orange dot). This allows ReSTIR-based lighting by tracing between one and two rays per pixel, based on desired quality.

8. Results

Our experimental prototype uses Falcor [BYC*20], though our ReSTIR lighting and sampling has been integrated into various other renderers. Detailed performance numbers and comparisons come from our prototype, where we have control to carefully instrument,

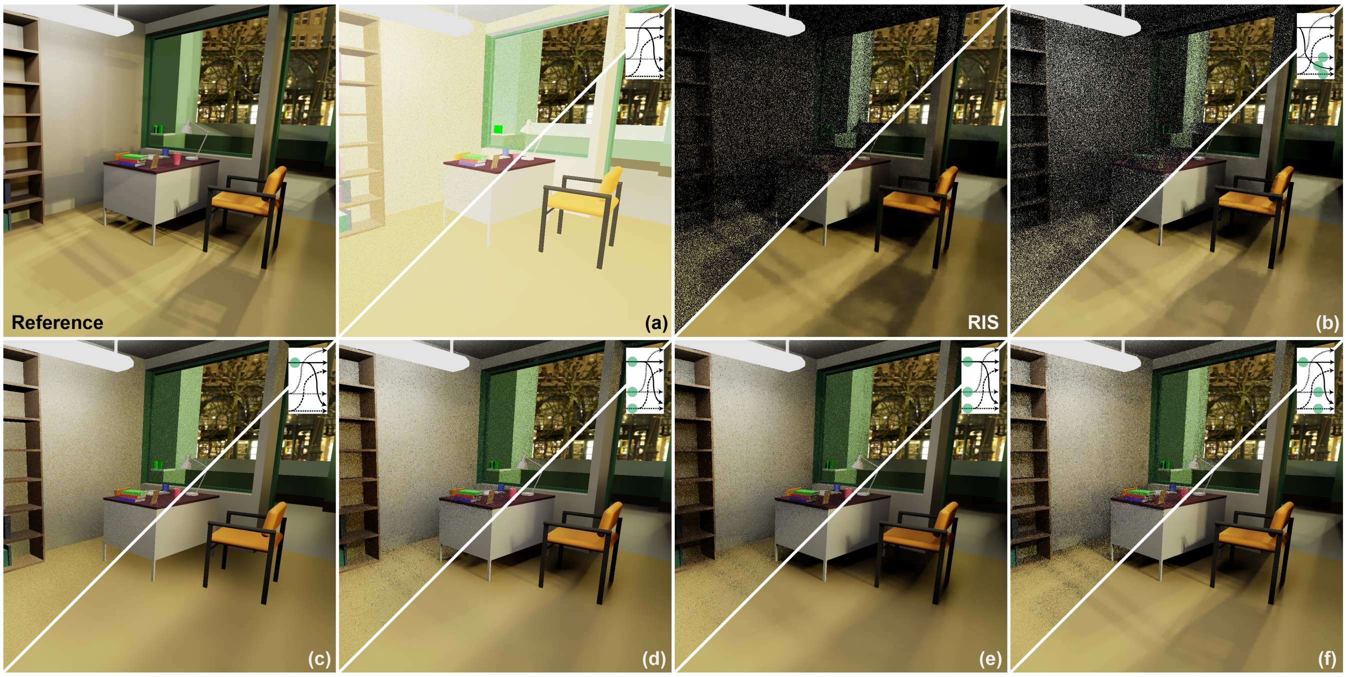


Figure 15: *Examples of biased visibility reuse in the Berkeley Soda Hall model, lit by an HDR environment and 52,000 emissive triangles scattered through over a hundred mutually-occluded rooms on six floors. Each real-time image shows both pre- and post-denoised results (using the ReLAX denoiser [KC21]). To compare, we also show unbiased RIS (without spatiotemporal reuse). ReSTIR results include an overlaid diagram, similar to Figure 13, showing dots where we trace visibility rays. (a) ReSTIR without any visibility, showing the importance of shadows in complex environments. (b) Using visibility only for shading loses many advantages of reuse, but remains unbiased. (c) With one ray per pixel for our winning candidate, and used for both resampling and shading, we gain most benefits of reuse but lose local contact shadows. (d,e) Biased spatiotemporal visibility reuse can occur multiple ways; in either case, future frames never reuse shadowed samples. However, we can (d) discard the entire reservoir, zeroing contributions from all prior samples, or (e) include prior sample weights in future reuse but never select the now-occluded sample. Overzealous discarding in (d) causes some lightening bias, discussed in Section 4.3, whereas (e) has the darkening discussed in Section 4.2. (f) The pipeline in Figure 13 gains the benefit of reusing candidate visibility (from (c)) and uses spatiotemporal visibility only for shading, reintroducing crisp contact shadows. Remaining bias stems from reusing initial visibility, visible as darkening of the fully lit region in the center of the floor (or on the patio).*

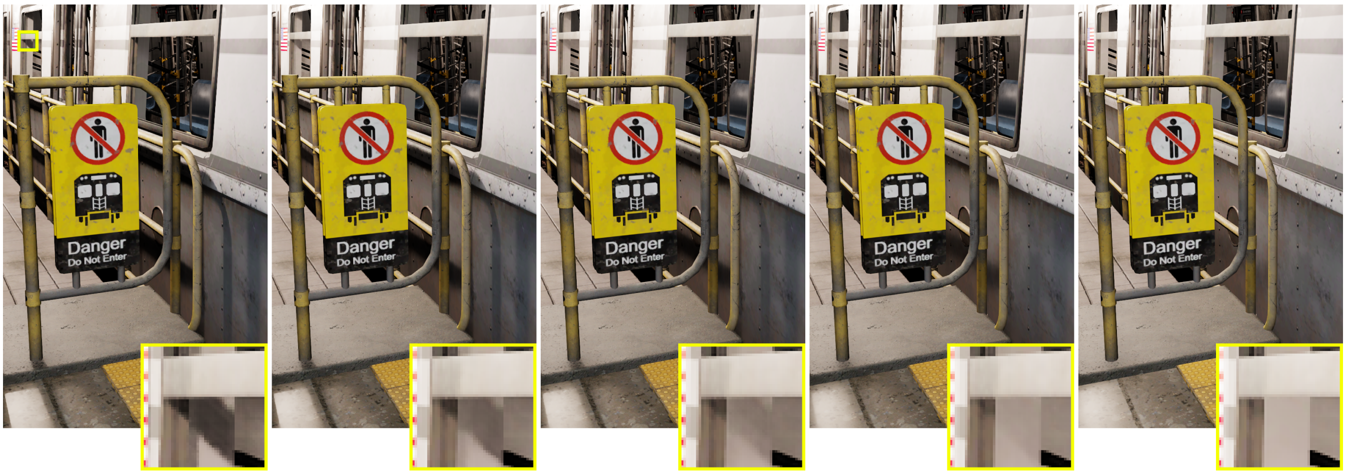


Figure 16: *Denoised results in the Subway scene, changing the per-pixel rays counts by reusing visibility for nearby neighbors. The intuition: when reusing lights from distant pixels, visibility is more likely to change. Left-to-right: 2 rays per pixel (no reuse), 1.75 rays per pixel (reuse within 15-pixel radius), 1.5 rays per pixel (20-pixel radius), 1.25 rays per pixel (25-pixel radius), and 1 ray per pixel (always reuse).*

do parameter sweeps, and validate settings. We profiled on a desktop GeForce RTX 3090 with an Intel Core i7-5820 and a laptop GeForce RTX 2080 Max-Q with an Intel Core i7-8750H.

Source code for a well documented exemplar implementation is available as part of NVIDIA's free RTXDI library [NVI21].

8.1. Implementation Details

We heavily instrumented our prototype to profile performance, and this naturally led to a three-kernel implementation: first, select and resample per-pixel candidates; second, shoot shadow rays for our selected candidates and spatiotemporal samples; and third, perform (decoupled) shading and resampling for next frame.

With no intra-frame dependencies, these can be fused into a single megakernel. But we found the reduced inter-kernel communication from fusing was counterbalanced by increased register usage and divergence caused by mixing ray tracing with compute inside a single, complex kernel. However, revisiting this decision with future API and hardware improvements could save a couple tenths of milliseconds—a big improvement in our simpler scenes.

All scenes in Table 1 (and elsewhere) use identical settings, unless otherwise noted. This includes $M = 32$ candidates per pixel, 1 temporal and 1 spatiotemporal sample, 2 visibility rays per pixel, $\mathbb{S} = 128$, $|S_j| = 1024$, and 8×8 pixel tiles for initial candidates. Comparisons to Bitterli et al. [BWP*20] use the parameters from their paper, again uniformly across all scenes.

While not our focus here, for denoising we use ReLAX [KC21] from NVIDIA's NRD library [NVI20]. This draws inspiration from spatiotemporal variance-guided denoisers [SKW*17,SPD18] while improving the denoising of specular reflections, firefly suppression, temporal stability, robust disocclusion handling, and speed.

8.2. Performance

Table 1 collects detailed performance measurements comparing our optimizations and baseline ReSTIR [BWP*20] across various scenes, ranging from a Cornell Box to the 150 million triangle Paris Opera House. Our scenes contain up to 3.4 million emissive triangles, some textured. Our prototype allows two light types: emissive triangles and environment maps, though other engine integrations support analytic light primitives (see Figure 1), per Section 5.3.

Note, Table 1 only evaluates ReSTIR-specific lighting costs. We do not include G-buffer creation, tonemapping, denoising, or other post-process passes. Our G-buffer takes a couple milliseconds, and our denoiser runs in 1.8 ms at 1920×1080 on our RTX 3090.

The key result: our optimized ReSTIR pipeline is over 4× faster than Bitterli et al. [BWP*20], unless limited by ray tracing (e.g., Emerald Square). In scenes previously dominated by incoherent memory during candidate selection, our pipeline runs up to 7× faster. When memory-constrained, by either scene complexity or device limits, our decoupled pipeline runs up to 20× faster. All our scenes now run interactively on our laptop.

Importantly, compare our results before decoupling shading (yellow lines in Table 1) with unoptimized ReSTIR (red lines). Across the board, all costs decrease, especially in more complex scenes. This is due to continued benefits of improved memory coherence. Caches are not thrashed after candidate selection, and selected samples are probably nearby neighbor samples. Note, incoherent memory still slightly slows performance in our decoupled pipeline; the "Reuse and Shade" cost rises with increased lighting complexity (scenes with either many emissives or an environment map).

In our rearchitected ReSTIR, performance scales linearly across most important parameters, including screen resolution and light pool count and size ($\mathbb{S}$ and $|S_j|$). Candidate selection cost varies non-linearly when changing light pool parameters $\mathbb{S}$ and $|S_j|$, as those values impact caching behavior. Typically, increasing light pool size $|S_j|$ is more destructive to coherence; larger pools use more cache lines, increasing memory contention between warps.

As in Bitterli et al. [BWP*20], we used $M = 32$ candidates across our scenes, as cost is reasonable for good quality. Cost is linear in M. Increasing M is rarely worthwhile; ReSTIR provides exponential growth to the effective sample count, and this generally trumps any linear increase provided by M. However, reducing M provides a performance win in simpler scenes (e.g., the Cornell Box looks similar with $M = 1$), for users willing to tweak settings per-scene.

Our new decoupled pipeline improves quality, discussed below, while maintaining roughly equal performance. On our RTX 3090, decoupling shading always reduces cost, largely due to use of fewer kernels with less launch overhead and intermediate buffers. But decoupled shading never traces a ray to the same sample twice per pixel, which can increase ray tracing costs on slower hardware.

8.3. Image Quality of Decoupled Shading

While providing significant performance improvements, we need to verify that image quality does not suffer.

Figure 17 zooms in on interesting regions of our scenes to compare quality before and after we decouple shading and reuse (i.e., with and without Section 7). Both variants shoot 2 rays per pixel and touch the same samples, i.e., the only difference is our decoupled pipeline shades three samples per pixel instead of just one. In particular quality improves in shadows, as decoupled shading provides three samples that can contribute lighting. This reduces the number of black pixels with zero lighting.

Generally, decoupled reuse improves perceived quality and image metrics. However, Figure 17 shows that in unshadowed regions, when ReSTIR converges well, decoupled shading can slightly increase noise. This also appears in the Cornell Box (see error metrics in Table 1). The problem is shading three light samples per pixel that have very different quality. Specifically, candidate light samples have $M = 32$, while converged (spatio-)temporal samples have effective M in the thousands. Using MIS weights to blend colors reduces this problem, but does not remove it entirely.

Selectively shading only two of the samples may allow designing a decoupled pipeline that always improves quality, but we did not investigate this. Our denoiser easily handles slightly increased noise in well-converged regions; the key problem is reducing noise in poorly-sampled regions, which decoupled shading achieves.

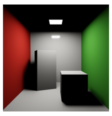
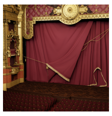
	Cornell Box	Arcade	Subway	Soda Hall	Amazon Bistro	Emerald Square	Amusement Park	Opera House
Emissive Triangle Count	2	124	25,194	52,448	20,638	89,279	3,381,540	538,916
Scene Triangle Count	36	7,798	3,847,379	2,169,211	2,832,120	10,046,405	22,945,649	150,670,732
Uses Environment Map?	✗	✓	✗	✓	✓	✗	✗	✗
Uses Alpha Test?	✗	✗	✗	✗	✓	✓	✓	✗
								
Our Results at Maximal Quality (i.e., using Sections 5, 6, and 7)								Times in milliseconds on a RTX 3090 (and RTX 2080 Max-Q)
Presampling (Sec. 5.2)	0.02 (0.02)	0.02 (0.03)	0.03 (0.10)	0.02 0.06)	0.02 (0.06)	0.02 (0.05)	0.04 (0.29)	0.03 (0.14)
Initial Candidates (Sec. 5.2)	0.5 (1.1)	0.8 (2.1)	0.8 (2.1)	0.7 (2.0)	0.8 (2.7)	0.8 (2.0)	0.8 (2.1)	0.8 (2.0)
Trace Rays (Sec. 7.3)	0.3 (0.7)	0.5 (1.3)	0.9 (2.6)	0.5 (1.5)	1.5 (4.8)	3.9 (14.0)	3.1 (10.8)	2.3 (13.3)
Reuse and Shade (Sec. 7)	0.2 (0.7)	0.6 (2.5)	0.4 (1.5)	0.5 (2.3)	0.8 (4.7)	0.4 (1.5)	0.6 (3.8)	0.6 (4.0)
Total Lighting	1.0 (2.5)	1.9 (6.1)	2.1 (6.3)	1.8 (5.9)	3.1 (12.2)	5.1 (17.7)	4.6 (16.9)	3.8 (19.5)
Speedup Over [BWP*20]	**4.0**× (4.3×)	**4.4**× (4.5×)	**4.8**× (5.9×)	**6.3**× (N/A)	**4.0**× (3.1×)	**2.9**× (2.6×)	**6.7**× (19.2×)	N/A
Speedup w/Decoupled Shading	1.2× (1.3×)	1.1× (1.0×)	1.1× (1.2×)	1.1× (1.1×)	1.1× (0.9×)	1.1× (1.0×)	1.0× (0.9×)	1.0× (0.9×)
Unoptimized ReSTIR [BWP*20]								Times in milliseconds on a RTX 3090 (and RTX 2080 Max-Q)
Initial Candidates	0.5 (1.4)	1.3 (3.0)	3.1 (5.8)	3.7 (N/A)	1.4 (3.6)	1.8 (6.1)	18.3 (98.7)	N/A
Candidate Visibility	0.2 (0.4)	0.3 (0.8)	0.5 (1.1)	0.6 (N/A)	1.3 (4.0)	2.8 (9.4)	2.6 (27.9)	N/A
Spatiotemporal Reuse (10 taps)	1.2 (4.4)	2.9 (14.0)	2.8 (12.4)	2.4 (N/A)	2.8 (10.4)	2.4 (7.8)	3.9 (104.4)	N/A
Shade With Visibility	0.6 (1.5)	2.1 (6.2)	1.7 (14.8)	3.1 (N/A)	5.5 (17.2)	6.1 (20.0)	4.3 (57.7)	N/A
Total Lighting	4.0 (10.8)	8.4 (27.3)	10.0 (37.1)	11.3 (N/A)	12.3 (38.4)	14.7 (46.7)	31.0 (324.1)	N/A
Our Results, Without Decoupling Shading (i.e., before Section 7)								Times in milliseconds on a RTX 3090 (and RTX 2080 Max-Q)
Initial Candidates (Sec. 5.2)	0.5 (1.1)	0.8 (2.1)	0.8 (2.1)	0.7 (2.0)	0.8 (2.7)	0.8 (2.0)	0.8 (2.1)	0.8 (2.0)
Candidate Visibility	0.1 (0.3)	0.2 (0.7)	0.4 (1.4)	0.3 (0.7)	0.9 (2.8)	2.4 (8.7)	2.2 (7.0)	1.4 (8.7)
Spatiotemporal Reuse (1 tap)	0.3 (1.0)	0.7 (2.1)	0.6 (1.8)	0.6 (1.9)	0.7 (2.5)	0.5 (1.6)	0.6 (2.1)	0.6 (1.9)
Spatiotemporal Reuse (10 taps)	1.1 (2.6)	2.5 (6.4)	2.0 (5.5)	2.0 (5.5)	2.4 (6.8)	1.7 (4.6)	2.0 (5.9)	2.1 (5.7)
Shade Visibility (1 ray)	0.2 (0.4)	0.3 (0.7)	0.5 (1.4)	0.3 (0.9)	0.7 (2.2)	1.5 (4.6)	1.0 (3.3)	0.8 (3.2)
Final Shading	0.1 (0.4)	0.2 (0.7)	0.2 (0.6)	0.2 (0.6)	0.2 (0.9)	0.2 (0.6)	0.2 (0.7)	0.2 (0.7)
Total Lighting (1 tap)	1.2 (3.2)	2.1 (6.4)	2.4 (7.3)	2.0 (6.3)	3.3 (11.1)	5.4 (17.5)	4.8 (15.2)	3.8 (16.6)
Varying Candidate Pixel Tile Size, see Section 5.2								Times in milliseconds on a RTX 3090 (and RTX 2080 Max-Q)
Initial Candidates (1x1)	2.6 (3.2)	4.1 (6.7)	4.4 (7.2)	3.9 (6.5)	4.1 (6.8)	4.4 (6.9)	4.4 (7.2)	4.4 (6.9)
Initial Candidates (2x2)	1.0 (2.8)	2.2 (5.7)	1.9 (6.1)	2.1 (3.8)	2.2 (8.6)	1.8 (5.9)	1.8 (6.0)	1.9 (6.1)
Initial Candidates (4x4)	0.5 (1.6)	1.1 (3.1)	1.4 (2.9)	1.0 (2.9)	1.1 (5.2)	0.9 (3.0)	0.9 (2.8)	1.0 (2.9)
Initial Candidates (8x8)	0.5 (1.1)	0.8 (2.1)	0.8 (2.1)	0.7 (2.0)	0.8 (2.7)	0.8 (2.0)	0.8 (2.1)	0.8 (2.0)
Initial Candidates (16x16)	0.5 (1.0)	0.8 (2.1)	0.8 (2.1)	0.8 (2.1)	0.8 (2.2)	0.8 (2.0)	0.8 (2.1)	0.8 (2.0)
Varying Size of Light Subsets $\lvert S_j \rvert$, see Section 5.2								Times on RTX 3090, for presampling / candidate selection
$\lvert S_j \rvert = 1024$, $\mathbb{S} = 128$	0.02 / 0.5	0.02 / 0.8	0.03 / 0.8	0.02 / 0.7	0.02 / 0.8	0.02 / 0.8	0.04 / 0.8	0.03 / 0.8
$\lvert S_j \rvert = 4096$, $\mathbb{S} = 128$	0.04 / 1.0	0.04 / 1.6	0.07 / 1.9	0.05 / 1.5	0.05 / 2.2	0.05 / 1.8	0.14 / 1.8	0.09 / 1.9
$\lvert S_j \rvert = 16384$, $\mathbb{S} = 128$	0.10 / 3.6	0.11 / 5.3	0.22 / 6.2	0.17 / 5.1	0.15 / 6.1	0.14 / 6.2	0.52 / 6.2	0.32 / 6.2
Varying Number of Light Subsets $\mathbb{S}$, see Section 5.2								Times on RTX 3090, for presampling / candidate selection
$\mathbb{S} = 128$, $\lvert S_j \rvert = 1024$	0.02 / 0.5	0.02 / 0.8	0.03 / 0.8	0.02 / 0.7	0.02 / 0.8	0.02 / 0.8	0.04 / 0.8	0.03 / 0.8
$\mathbb{S} = 1024$, $\lvert S_j \rvert = 1024$	0.06 / 0.7	0.07 / 1.0	0.13 / 1.1	0.09 / 0.9	0.09 / 1.2	0.10 / 1.1	0.27 / 1.1	0.17 / 1.1
$\mathbb{S} = 4096$, $\lvert S_j \rvert = 1024$	0.19 / 0.8	0.20 / 1.1	0.45 / 1.3	0.32 / 1.0	0.29 / 1.3	0.29 / 1.3	1.10 / 1.3	0.66 / 1.3
Resolution Scaling of Our Decoupled ReSTIR								Times on RTX 3090, for resolutions 1280 × 720, 1920 × 1080, and 2560 × 1440
Initial Candidates (Sec. 5.2)	0.2 / 0.5 / 0.8	0.4 / 0.8 / 1.5	0.4 / 0.8 / 1.5	0.3 / 0.7 / 1.3	0.4 / 0.8 / 1.5	0.4 / 0.8 / 1.4	0.4 / 0.8 / 1.4	0.4 / 0.8 / 1.6
Trace Rays (Sec. 7.3)	0.2 / 0.3 / 0.5	0.2 / 0.5 / 0.8	0.4 / 0.9 / 1.5	0.2 / 0.5 / 0.9	0.7 / 1.5 / 2.7	1.9 / 3.9 / 7.0	1.4 / 3.1 / 5.6	1.1 / 2.3 / 4.1
Reuse and Shade (Sec. 7)	0.1 / 0.2 / 0.4	0.3 / 0.6 / 1.2	0.2 / 0.4 / 0.8	0.2 / 0.5 / 1.0	0.3 / 0.8 / 1.7	0.2 / 0.4 / 0.7	0.3 / 0.6 / 1.2	0.3 / 0.6 / 1.3
Total Lighting	0.5 / 1.0 / 1.8	0.8 / 1.9 / 3.5	0.9 / 2.1 / 3.9	0.8 / 1.8 / 3.3	1.4 / 3.1 / 5.8	2.5 / 5.1 / 9.1	2.1 / 4.6 / 8.2	1.8 / 3.8 / 7.0
Image Error Versus Offline Reference [Mean Squared Error (MSE)]								
Our Decoupled Pipeline	0.0028	0.0170	0.0716	0.0494	0.1428	0.0471	0.1260	0.0655
Our Pipeline (Before Sec. 7)	0.0023	0.0235	0.1142	0.0721	0.1929	0.0482	0.1592	0.0856
ReSTIR [BWP*20], Biased	0.0038	0.0341	0.1322	0.0950	0.2440	0.0553	0.1801	N/A
ReSTIR [BWP*20], Unbiased	0.0035	0.0217	0.0802	0.0467	0.1434	0.0547	0.1473	N/A
Image Error Versus Offline Reference [ꟻLIP [ANA*20]]								
Our Decoupled Pipeline	0.0326	0.1255	0.1930	0.2246	0.3080	0.2329	0.3238	0.2533
ReSTIR [BWP*20], Biased	0.0345	0.1374	0.1903	0.2457	0.3369	0.2322	0.3419	N/A

Table 1: *Performance with various settings at* 1920×1080. *Times are in milliseconds on a RTX 3090 (and RTX 2080 Max-Q). Green lines show our final performance with maximum quality (2 rays per pixel); fewer rays can be used, per the discussion in Section 7.3. Red lines show comparable times for unoptimized ReSTIR [BWP*20], separated per Figure 2, except spatiotemporal costs are combined; total cost is larger than the component sum, due to synchronization. Yellow lines show our results before decoupling shading and reuse, comparing the reduced spatial samples from Section 6 and Bitterli et al.'s [BWP*20] 10 spatial taps. Un-highlighted lines show how varying the presampled light subsets and pixel tile sizes, in Section 5.2, impacts performance. At the bottom, we explore performance scaling with resolution and compare mean square errors (MSE) between our work and Bitterli et al. [BWP*20]. The ꟻLIP perceptual metric [ANA*20] also validates we maintain or improve image quality with our better performing pipeline.*

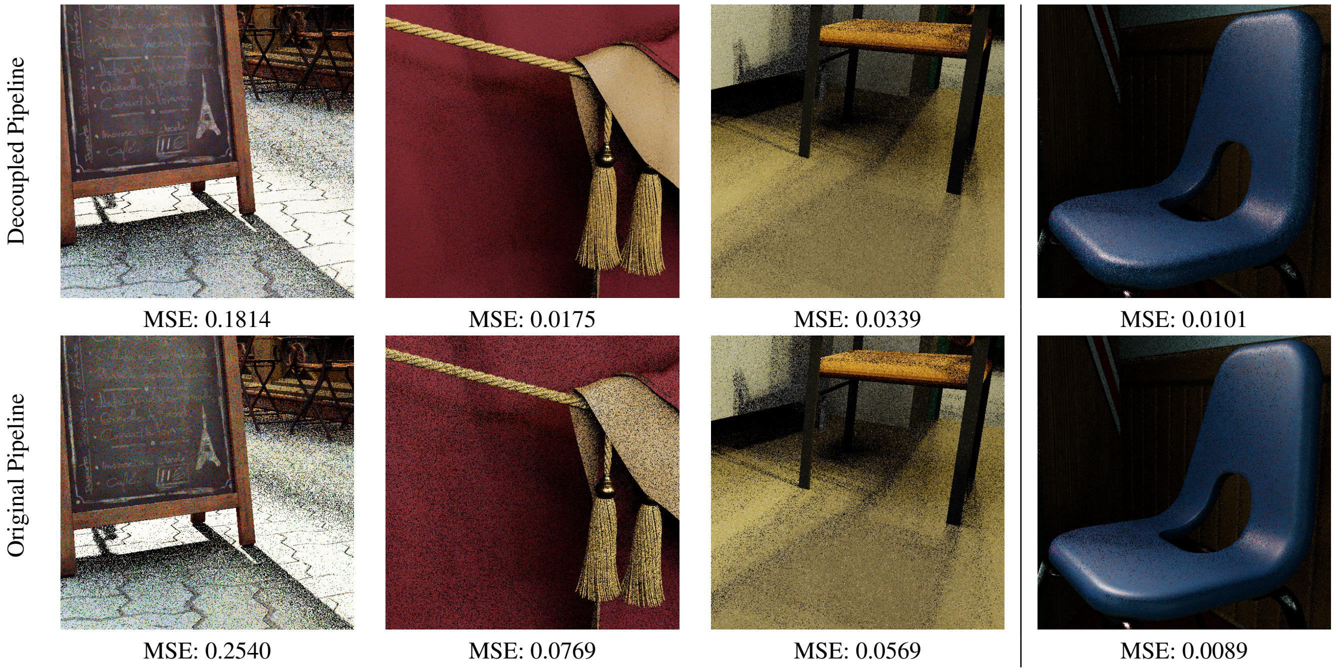


Figure 17: *Compare quality between our decoupled pipeline (Section 7) and the original computation pipeline (i.e., organized per Bitterli et al. [BWP*20]). Both methods shade and reuse the same samples (i.e., access the same memory), and trace 2 rays per pixel. Here, we focus on interesting inset locations in our Bistro, Opera House, Soda Hall, and Arcade scenes. At right, we illustrate a limitation of decoupling: it can somewhat increase noise in well-converged, unshadowed regions. This is due to forced shading of one (possibly poor) candidate each frame. However, decoupled shading still reduces error in typical full-scene views of the Arcade (MSE 0.0170 v.s. 0.0235; see Table 1).*

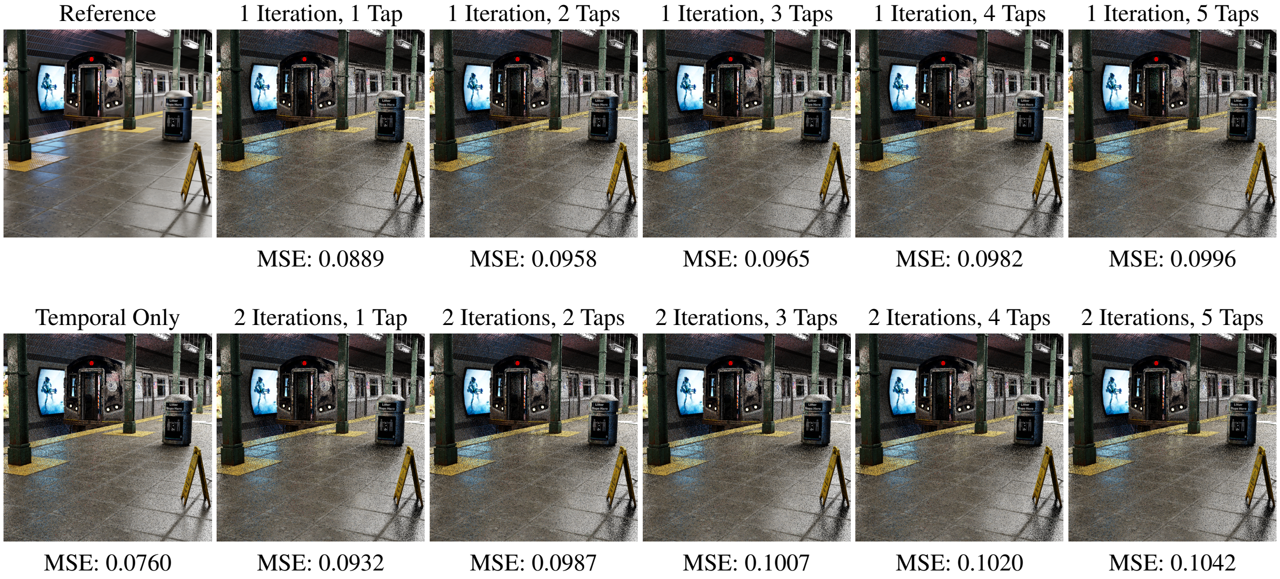


Figure 18: *Comparing different spatial sample counts. Section 6.3 suggests understanding the relative importance of spatial and temporal samples. In our Subway scene, we show a reference, temporal resampling only, and spatiotemporal reuse with various spatial sample counts (between 1 and 10 spatial samples). Spatial samples occur in either one or two iterations of resampling, with between one and five samples per pass. Bitterli et al. [BWP*20] use 2 iterations with 5 spatial samples each. For reference, our decoupled pipeline from Section 7 has an MSE of 0.0425 with just 1 temporal sample and an MSE of 0.0546 with 1 temporal and 1 spatiotemporal sample in this view.*

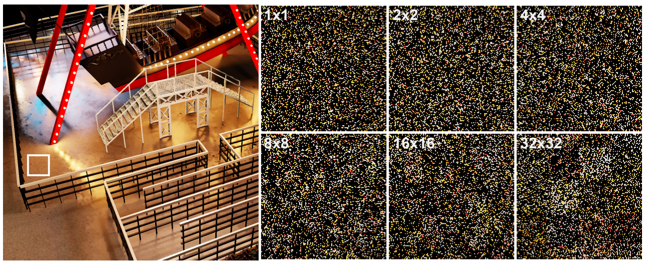


Figure 19: *Using presampled light pools (Section 5.2) introduces correlations within and between tiles of pixels. Insets show lighting directly with our initial candidates (no reuse), using different pixel tile sizes, to show this correlation. All with $\mathbb{S} = 128$ and $|S_J| = 1024$.*

8.4. Quality Impact of Reduced Sampling Frequency

Section 6 significantly changes how spatial, temporal, and visibility sampling occurs. Surprisingly, our changes often increase image quality. The key quality regression is a reduction in visibility queries; but, as shown in Figure 9, tracing four visibility rays for final shading wastes a lot of work. In fact, decoupled shading provides a better noise reduction in these regions (see Table 1).

Figure 18 compares quality with varying numbers of spatial samples used for reuse. Bitterli et al. [BWP*20] uses 2 spatial iterations with 5 taps each, which has the highest MSE shown in Figure 18. Spatial samples simply estimate the current pixel worse than (valid) temporal estimates. Figure 11 shows spatial samples help quickly reduce noise in disocclusions (or other temporal discontinuities), but they can worsen quality elsewhere.

For biased ReSTIR, part of the increased error comes from additional bias introduced by each spatial sample. But measuring MSE for unbiased ReSTIR shows a similar increase in error, albeit of smaller magnitude (i.e., from 0.0770 to 0.0829 as spatial sample count increases from one to five).

This leads to somewhat contradictory conclusions: spatial samples allow the exponential growth in effective sample count M, and thus are vital to ReSTIR. They also help fill discontinuities, when temporal samples get invalidated. However, using too many spatial samples increases error. Adaptively changing spatial sample count only in disoccluded regions can carefully target increased reuse.

8.5. Quality Impact of Prerandomized Light Samples

Section 5 is somewhat surprising—it suggests replacing 32 independent light samples per pixel (64 million independent samples in a 1920×1080 image) by a few, much smaller subsets, independently randomized once per frame. We empirically chose $\mathbb{S} = 128$ subsets of size $|S_j| = 1024$. This reduction, of course, enables much of the performance boost shown in Figure 7 and Table 1.

Obviously, reducing the sample size introduces correlation. This may be worrying, until we remember ReSTIR is fundamentally about leveraging correlated sampling to improve image quality. We simply propose *also* leveraging correlations to reduce cost.

Figure 19 shows these correlations by lighting the initial candidates directly (before reuse) using different size screen tiles; every pixel in tile picks 32 independent samples from the *same* pool of $|S_j| = 1024$ light samples. Correlations clearly occur with 8×8 and larger tiles, and they are more obvious under animation.

Correlations appear because each screen tile samples from pool S_j either $32*8 \times 8$, $32*16 \times 16$, or $32*32 \times 32$ times. All of those are larger than our selected size $|S_j| = 1024$. Pixels become more correlated as tile size increases. Choosing $|S_j| = 16384$ removes any visible correlation for even 32×32 tiles.

ReSTIR handles limited correlations within 8×8 tiles, as pixels also spatiotemporally sample from a larger, 30-pixel neighborhood. These samples come from *different* light pools, as they are re-randomized each frame; this helps decorrelate neighbors before shading and reuse.

Another, less obvious, issue exists with prerandomized light sets. Different screen tiles use the same light pool. This correlates distant parts of the image. But this only matters if such correlations are visible or if they impact resampling quality. Fortunately, these correlations are simply not perceptible with reasonable sized $\mathbb{S}$ and $|S_j|$. (Though $\mathbb{S} = 4$ and $|S_j| = 32$ makes them obvious).

And long-distance correlations do not impact ReSTIR since we draw spatiotemporal neighbors from a limited region. We only need samples in our *neighborhood* to remain relatively uncorrelated. A 30-pixel radius neighborhood contains $30 \times 30 \times \pi \times 32 \approx 90,000$ samples, if pixels use $M = 32$ candidates. And this 30-pixel filter touches 45-50 of our 8×8 pixel tiles. This suggests our empirically-determined $\mathbb{S}$ and $|S_j|$ are reasonable, giving perhaps 40,000 independent samples in ReSTIR's 30-pixel search radius. This approaches the 90,000 needed for truly independent sampling. Interestingly, this logic is independent of scene complexity.

9. Discussion

Beyond specific quality and performance issues discussed above, our explorations highlighted various other interesting observations.

Numerical precision. Using weighted reservoir sampling to sum samples over many frames seems likely to encounter problems from catastrophic cancellation. Fortunately, each pixel only sums $32 + 2$ values, our candidates plus one temporal and one spatiotemporal sample. Reservoirs are renormalized between frames (i.e., sums become averages) by Bitterli et al.'s [BWP*20] clamping of M to $20\times$ the current reservoir's M. This virtually eliminates inter-frame precision issues.

Ray coherence. Shadow rays for per-pixel candidates are extremely incoherent, by design. But the resampling process makes our selected samples more correlated and, thus, more coherent. See the yellow-highlighted lines in Table 1, where one *shade visibility* ray is significantly cheaper than one *candidate visibility* ray. This likely affects relative performance of our decoupled pipeline, which benefits somewhat less from this improved coherence.

Expensive materials. ReSTIR evaluates materials multiple times per pixel, to compute target functions $\hat{p}_i$ during reuse. When integrating into renderers with complex layered or procedural material models, this may be worrying. If materials can be evaluated once per pixel and baked into a G-buffer for deferred shading, ReSTIR

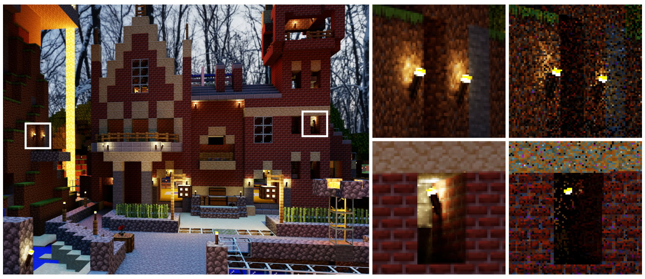


Figure 20: *Resampling can undersample very complex lighting. A reference with insets, (left) reference and (right) ReSTIR, of the extensive "Vokselia" Minecraft world with 200,000 emissives, mostly torches with limited range. Torch lighting flickers temporally, as pixels rarely find a relevant light; when a torch* is *proposed as a candidate, it is rarely at a pixel within range of the torchlight.*

simply consumes this G-buffer data. Alternatively, target functions $\hat{p}_i$ can use simplified material models at the cost of increased noise.

Alternate forms of visibility. Section 7.3 shows we achieve plausible results without accurately knowing each sample's visibility and proposes a simple knob to parameterize quality. We could also imagine using other, faster visibility approximations, e.g., screen-space ray tracing [MM14] or screen-space directional occlusion [RGS09]. Such use will likely inherit their limitations, but in a decoupled pipeline we would apply them only when shading.

Scaling performance lower. Even the knobs we introduce to tune performance and quality may be insufficient for today's lower-end hardware. Our pipeline interacts well with checkerboard rendering [Wih17] and similar techniques.

MIS with BRDF rays. Our source pdfs $q(\omega)$ need not only sample lights; BRDF rays can be included. We advise not combining a BRDF ray with the *result* of ReSTIR; this neglects any benefit the BRDF ray could provide to nearby pixels via reuse. Additionally, ReSTIR combines thousands of samples while BRDF sampling provides just a few; determining an appropriate weighting after reuse occurs is more challenging.

10. Limitations

There are three key issues that underlie the limitations in ReSTIR: perfect specular materials, undersampling, and correlations.

For perfect specular surfaces, firing a reflection ray easily computes correct lighting. For near-specular surfaces, BRDF sampling can perform better than ReSTIR, largely because narrow specular lobes radically limit the set of useful neighbors. This limits the benefits of reuse on these materials.

While ReSTIR multiplies our limited sampling budget, it has limits. Figure 20 shows an extremely large Minecraft world with over 200,000 emissive, mostly tiny torches. These torches have relatively limited range, so pixels need to find the single nearby, important torch. With 32 candidates per pixel and effective M values of perhaps ten thousand, this is insufficient to sample tiny emitters every frame. This leads to temporally flickering torchlight.

Correlated sampling gives ReSTIR its power, but also poses problems. Essentially, ReSTIR identifies important samples, holds on to them, and reweights them for nearby pixels. When it finds too few important samples, pixels all end up reusing the same sample, leading to blotchy, VPL-like artifacts. By reusing from many past frames, ReSTIR generally finds good samples. However, blotchiness worsens in conjunction with specular surfaces or undersampling, i.e., it is more visible on highly specular surfaces or in sparsely lit regions of large, well-illuminated scenes.

11. Conclusions

We presented numerous algorithmic improvements to dramatically improve performance and quality of spatiotemporal resampling for direct lighting in many-light scenes. We show leveraging resampling to reshape computations allows removing many incoherent memory accesses from our inner render loop. We also identified discarded work that, by decoupling shading and sample reuse, improves shading quality at negligible cost. We also empirically explored ReSTIR's parameter space, finding that spatial sampling is less useful than initially imagined; this allows turning down quality settings surprisingly far without reducing image quality.

Combined these provide improved quality and up to a 7× performance improvement, with even larger gains for complex scenes on memory-constrained, lower-performance hardware. We also expose a new performance knob that allows scaling down ray costs in exchange for less defined contact shadows.

We hope that, given our improvements, ReSTIR-based sampling techniques will continue being developed to leverage correlations for improved quality in more complex real-time light transport, e.g., global illumination [OLK*21].

Acknowledgements

Thanks to our numerous colleagues and collaborators who helped improve and make this work possible. Particular thanks to William Newhall, Bob Alfieri, Pete Shirley, and Dave Luebke for discussions, ideas, and brainstorming leading to our use of resampling to reshape computation. Pawel Kozlowski and Tim Cheblokov did heroic work on the ReLAX denoiser. Many thanks to Apollo Ellis, Jakub Bokšanský, Paula Jukarainen, Marco Salvi, Max Eisenstein, Jacopo Pantaleoni, Yaobin Ouyang, Simon Kallweit, and Adam Badke who all reimplemented ReSTIR and, through it, helped identify ways to (hopefully) describe the math more intuitively. And thanks to Benedikt Bitterli, Daqi Lin, Matt Pharr, Wojciech Jarosz, and Cem Yuksel for ongoing discussions to help evolve our understanding of spatiotemporal resampling. Pontus Andersson graciously computed ꟻLIP errors. Mike Songy, Jon Story, and Aaron Lefohn supported the research, development, and engineering.

References

[ANA*20] ANDERSSON P., NILSSON J., AKENINE-MÖLLER T., OSKARSSON M., ÅSTRÖM K., FAIRCHILD M. D.: ꟻLIP: A difference evaluator for alternating images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 3*, 2 (2020), 15:1–15:23. 15

[ANS*19] ANDERSSON P., NILSSON J., SALVI M., SPJUT J., AKENINE-MÖLLER T.: Temporally dense ray tracing. In *High-Performance Graphics* (2019). doi:10.2312/hpg.20191193. 9

[BAC*18] Burley B., Adler D., Chiang M. J.-Y., Driskill H., Habel R., Kelly P., Kutz P., Li Y. K., Teece D.: The design and evolution of disney's hyperion renderer. *ACM Transactions on Graphics 37*, 3 (2018). doi:10.1145/3182159. 1

[BK19] Binder N., Keller A.: Massively parallel construction of radix tree forests for the efficient sampling of discrete probability distributions, 2019. arXiv:1901.05423. 9

[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 Transactions on Graphics 39*, 4 (2020). doi:10.1145/3386569.3392481. 1, 2, 4, 5, 6, 7, 8, 9, 10, 14, 15, 16, 17

[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, 2020. URL: https://github.com/NVIDIAGameWorks/Falcor. 1, 12

[Cha82] Chao M.-T.: A general purpose unequal probability sampling plan. *Biometrika 69*, 3 (1982), 653–656. doi:10.2307/2336002. 4

[Coo86] Cook R. L.: Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1 (1986), 51–72. doi:10.1145/7529.8927. 8

[DBB06] Dutré P., Bekaert P., Bala K.: *Advanced Global Illumination*, 2nd ed. AK Peters, 2006. 2

[Dev86] Devroye L.: *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986. doi:10.1007/978-1-4613-8643-8. 2

[EHDR11] Egan K., Hecht F., Durand F., Ramamoorthi R.: Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Transactions on Graphics 30*, 2 (2011), 1–13. doi:10.1145/1944846.1944849. 10

[FHL*18] Fascione L., Hanika J., Leone M., Droske M., Schwarzhaupt J., Davidovič T., Weidlich A., Meng J.: Manuka: A batch-shading architecture for spectral path tracing in movie production. *ACM Transactions on Graphics 37*, 3 (2018). doi:10.1145/3182161. 1

[FM84] Fishman G. S., Moore L. R.: Sampling from a discrete distribution while preserving monotonicity. *The American Statistician 38*, 3 (1984), 219–223. doi:10.2307/2683661. 9

[Har20] Harada T.: Hardware-accelerated ray tracing in Radeon ProRender 2.0. https://gpuopen.com/videos/rdna2-raytracing-radeon-prorender-2/, 2020. [Online; accessed 8-February-2021]. 1

[JWPJ16] Jimenez J., Wu X.-C., Pesce A., Jarabo A.: *Practical Realtime Strategies for Accurate Indirect Occlusion*. Tech. Rep. ATVI-TR-16-01, Activision, 2016. 10

[Kaj86] Kajiya J.: The rendering equation. In *Proceedings of SIGGRAPH* (1986), pp. 143–150. doi:10.1145/15886.15902. 2

[KC21] Kozlowski P., Cheblokov T.: ReLAX: A denoiser tailored to work with the ReSTIR algorithm. In *GPU Technology Conference (GTC)* (2021). URL: https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s32759/. 13, 14

[KMSB18] Kilgariff E., Moreton H., Stam N., Bell B.: NVIDIA Turing architecture in-depth. https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth/, 2018. [Online; accessed 19-January-2021]. 1

[MM14] McGuire M., Mara M.: Efficient GPU screen-space ray tracing. *Journal of Computer Graphics Techniques 3*, 4 (2014), 73–85. URL: http://jcgt.org/published/0003/04/04/. 18

[MML12] McGuire M., Mara M., Luebke D.: Scalable ambient obscurance. *Proceedings of High-Performance Graphics* (June 2012). doi:10.2312/EGGH/HPG12/097-103. 10

[MR95] Motwani R., Raghavan P.: *Randomized Algorithms*. Cambridge University Press, 1995. doi:10.1017/CBO9780511814075. 5

[MS18] Marschner S., Shirley P.: *Fundamentals of Computer Graphics*, 4th ed. CRC Press, 2018. 2

[NVI20] NVIDIA: NVIDIA® Real-Time Denoiser, Oct. 2020. URL: https://developer.nvidia.com/nvidia-rt-denoiser. 1, 11, 14

[NVI21] NVIDIA: NVIDIA® RTX Direct Illumination, Apr. 2021. URL: https://developer.nvidia.com/rtxdi. 14

[OLK*21] Ouyang Y., Liu S., Kettunen M., Pharr M., Pantaleoni J.: ReSTIR GI: Path resampling for real-time path tracing. *Computer Graphics Forum 40*, 8 (2021), to appear. 18

[PJH16] Pharr M., Jacob W., Humphreys G.: *Physically Based Rendering: From Theory to Practice*, 3rd ed. Morgan Kaufmann, 2016. http://www.pbr-book.org/. 2, 9

[RGS09] Ritschel T., Grosch T., Seidel H.-P.: Approximating dynamic global illumination in image space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (2009), p. 75–82. doi:10.1145/1507149.1507161. 18

[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*. 2017, pp. 1–12. doi:10.1145/3105762.3105770. 1, 10, 14

[SPD18] Schied C., Peters C., Dachsbacher C.: Gradient estimation for real-time adaptive temporal filtering. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 2 (2018). doi:10.1145/3233301. 1, 14

[Tal05] Talbot J. F.: *Importance resampling for global illumination*. Master's thesis, Brigham Young University, 9 2005. URL: http://hdl.lib.byu.edu/1877/etd1021. 2, 4

[TCE05] Talbot J., Cline D., Egbert P.: Importance resampling for global illumination. In *Eurographics Symposium on Rendering* (2005), The Eurographics Association, pp. 139–146. doi:10.2312/EGWR/EGSR05/139-146. 2, 3, 4, 9

[TH16] Tokuyoshi Y., Harada T.: Stochastic light culling. *Journal of Computer Graphics Techniques 5*, 1 (2016), 35–60. URL: http://jcgt.org/published/0005/01/02/. 3

[VG95] Veach E., Guibas L. J.: Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques* (1995), p. 419–428. doi:10.1145/218380.218498. 6

[Vit85] Vitter J.: Random sampling with a reservoir. *ACM Transactions on Mathematical Software 11*, 1 (1985), 37–57. doi:10.1145/3147.3165. 4

[Vos91] Vose M.: A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering 17*, 9 (1991), 972–975. doi:10.1109/32.92917. 9

[Wal77] Walker A. J.: An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software 3*, 3 (1977), 253–256. doi:10.1145/355744.355749. 9

[Wih17] Wihlidal G.: 4k checkerboard in Battlefield 1 and Mass Effect Andromeda. GDC, 2017. URL: https://www.ea.com/frostbite/news/4k-checkerboard-in-battlefield-1-and-mass-effect-andromeda. 18