World Space Spatio-Temporal Reuse

Venkataram Edavamadathil Sivaram
University of California, San Diego

Figure 1: Comparison of Monte Carlo with ReSTIR and our world space resampling algorithms (16 spp, direct lighting only)

Abstract

In this paper, we propose a variation of the original Reservoir-based Spatio-Temporal Importance Resampling (ReSTIR) algorithm to improve direct lighting samples by performing temporal and spatial reuse in three dimensions. Specifically, our algorithm, World Space Spatio-Temporal Reuse (WSSR), proposes methods to arrange reservoirs in scene space to allow for more efficient spatial reuse. The first such algorithm uses a static k-d tree to efficiently store reservoirs in a hierarchical manner, while the second utilizes a dynamic uniform grid to simplify querying and updating its reservoirs.

1 Introduction

Monte Carlo path tracing has received much attention in recent years for the various advancements that have allowed it to become the de facto method in offline-rendering applications, and slowly for real-time applications as well. The progressive improvements in ray-tracing hardware has led to the integration of path-traced global illumination (GI) in traditionally rasterized applications like games. Despite advances in hardware and sampling techniques, generating noise-free images quickly in such interactive applications remains a challenge. The most common approach in commercial applications is to use a denoiser in the rendering pipeline to reconstruct the radiance [Schied et al., 2017, Chaitanya et al., 2017]. While these provide stunning results, artifacts are still present. Alternatively, some rendering engines use algorithms that interpolate radiance with radiance probes [Boissé et al., 2022, Majercik et al., 2019], which introduce their own set of complications to prevent artifacts such as light leakage and proper occlusion.

The primary challenge in real-time applications is to generate noise-free radiance of a dynamic scene within a limited time budget. Denoisers are a crucial part of the pipelines in this domain, but they require at least some level of non-sparsity in the noisy input it receives. Low-sample-per-pixel renderings must therefore extract the most significant contribution of the scene into the renders.

Recent work on resampling [Bitterli et al., 2020, Ouyang et al., 2021, Lin et al., 2022] has shown that the generation of these significant samples can be done much faster through

temporal and spatial reuse. However, for the aforementioned dynamic scenes, many of the reused samples are thrown away as the camera, lights, and objects move in the scene. In this paper, we propose a modification of the original ReSTIR algorithm to improve direct lighting samples by performing temporal and spatial reuse in three dimensions.

The preliminaries for resampling algorithms are discussed in Section 2. We then provide an overview of the original ReSTIR algorithm, before providing our main contributions of World Space Spatio-Temporal Reuse in Section 4. The implementation details and results are provided afterward, and we close with remarks on improving our current algorithm in Section 6.

2 Background

The goal of path-tracing is to evaluate the following integral through Monte-Carlo estimation:

$$L(x,\omega_o) = \int_{\mathbb{S}^2} f(x,\omega_o,\omega_i) L(y,-\omega_i)\, d\omega_i$$

The left-hand side of the equation above represents the radiance at a point x in the scene and outgoing direction ω_o. This radiance is equivalent to the sum of the emission at x as well as all the incoming radiance from directions surrounding x; y is the intersection location of a ray traveling from x with direction $-\omega_i$. In the case of techniques like Next-Event Estimation, f is the lighting term consisting of the BSDF, geometric, and direct lighting terms:

$$f(x,\omega_o,\omega_i) = \rho(\omega_o,\omega_i) L_d(x) G(x,\omega_o,\omega_i)$$

For convenience, let F be the integrand of the radiance integral, and denote X_i to be samples in its domain Γ. The default Monte Carlo estimator,

$$\langle L \rangle^N = \frac{\mu(\Gamma)}{N} \sum_{i=1}^{N} F(X_i)$$

where $\mu(\Gamma)$ is the measure of Γ, has variance which converges to zero at the rate $1/N$. The dimension of Γ increases with increasing levels of path depth because of its recursive nature. Subsequently, by the curse of dimensionality, evaluating it becomes challenging, and often requires numerous samples per pixel – often on the order of thousands – before a satisfactory noise-free result is achieved.

Importance Sampling is a popular technique used to reduce the variance of the samples generated during Monte Carlo integration. Rather than generating uniform samples in the multidimensional domain, importance sampling uses a source distribution p which closely resembles the integrand F and thus yields more significant samples on average. The Monte Carlo estimator for importance sampling becomes

$$\langle L \rangle^N_{IS} = \frac{1}{N} \sum_{i=1}^{N} \frac{F(X_i)}{p(X_i)}.$$

Critically, unbiased importance sampling requires that $p(X)$ is positive whenever $F(X)$ for arbitrary samples X.

2.1 Resampled Importance Sampling

In many cases, the integrand has an intractable source distribution; that is, either p cannot be determined or its evaluation is computationally expensive or sophisticated. Importance sampling in such cases becomes mostly infeasible.

Resampling importance sampling (RIS) [Talbot, 2005] suggests an alternative method for improved sampling which avoids the need for an explicit distribution $\bar{p}$, and instead uses a proxy distribution p that is sub-optimal, but cheap to compute. Then, from M *canonical* samples $X = \langle X_i \rangle$ taken from p, the resampled value z is selected with probability defined as

$$p(z \mid X) = \frac{w(z)}{\sum_{i=1}^{M} w(X_i)},\ w(X_i) = \frac{\hat{p}(X_i)}{p(X_i)},$$

where $\hat{p}$ is a target function that resembles F. The unbiased contribution weight of z, resampling the intractable $1/\bar{p}$ is

$$W_z = \frac{1}{\hat{p}(z)} \left(\frac{1}{M} \sum_{i=1}^{M} w(X_i) \right),$$

and the corresponding estimator becomes

$$\langle L \rangle^{N,M}_{RIS} = \frac{1}{N} \sum_{i=1}^{N} W_{z_i} F(z_i).$$

2.2 Weighted Reservoir Sampling

Naively collecting M canonical samples for resampling can lead to high memory usage, which in some rendering applications can nullify the benefits of RIS. Weighted Reservoir Sampling (WRS) provides an interface, namely *reservoirs*, that processes sequences of samples in a way that allows resampling to be done efficiently.

Reservoirs, and their operations *update* and *merge*, let the collection of streams to be done independently and then combined without explicit knowledge of the streams themselves.

Algorithm 1 shows pseudocode for these methods, as well as the layout of a reservoir. For generalization, we use $\hat{p}_R$ to refer to the target function used to select samples for R; in our algorithms, we use the same target function, $\hat{p} = \hat{p}_R$.

3 Reservoir-based Spatio-Temporal Reuse

Reservoirs provide the framework needed to effectively combine independent streams of samples. Reservoir-based Spatio-Temporal Importance Resampling (ReSTIR) utilizes this power to create a novel rendering pipeline with reuse at its core. Specifically, ReSTIR employs RIS and WRS for direct lighting to perform temporal and spatial resampling to exponentially increase the effective number of light samples

Algorithm 1 Reservoir streaming procedures

struct RESERVOIR
  z ▷ Selected sample
  w_{sum} ▷ Cumulative weight sum
  M ▷ Sample count
end struct

procedure UPDATE(R, X_i, w_i)
  $R.w_{sum} \leftarrow R.w_{sum} + w_i$
  $R.M \leftarrow R.M + 1$
  if RAND() $< w_i / R.w_{sum}$ **then**
    $R.z \leftarrow X_i$
  end if
end procedure

procedure MERGE(R, R_1, R_2, …, R_N)
  RESERVOIR S
  for $Q \in \langle R_i \rangle$ **do**
    UPDATE$(S, \hat{p}_R(Q.z) \cdot Q.w_{sum} \cdot Q.M)$
  end for
  $S.M \leftarrow \sum_{i=1}^{N} R_i.M$
end procedure

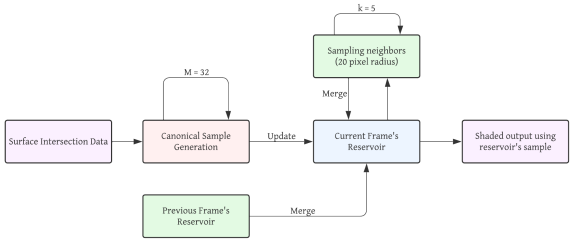


Figure 2: Sampling pipeline for the original ReSTIR work

taken. See Figure 2 to see the complete sampling pipeline for ReSTIR.

The combination of temporal and spatial reuse results in a significant boost in sample efficiency, as Figure 3 shows. However, this comes at the cost of performance.

3.1 Reprojection

The original ReSTIR algorithm focuses only on direct lighting. Interestingly, there is a simple way to extend ReSTIR for improving direct lighting samples taken at non-primary – indirect – intersections. In indoor scenes, many of these indirect intersections are likely to end up in the camera's view frustum. In this case, the intersection point can be reprojected onto the camera film, and the temporal reservoir corresponding to the projected location can be retrieved. Spatial sampling can also be performed since this location is known, leading to a recursive adaptation of ReSTIR for indirect intersections.

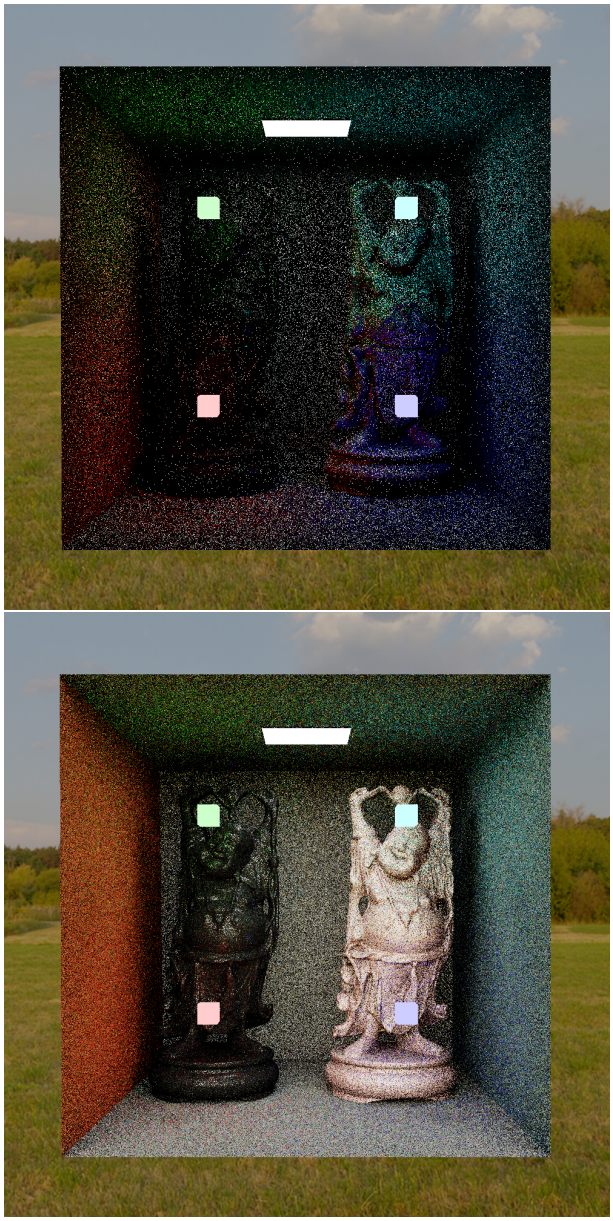


Figure 3: Effect of spatio-temporal reuse (bottom) in a scene with many lights and environment lighting (4 spp, direct lighting only)

Unfortunately, this addition only brings marginal improvements to indirect lighting; see Figure 4. However, we will use a similar idea to boost the performance of one of our algorithms.

4 World Space Spatio-Temporal Reuse

By itself, ReSTIR can outperform many state-of-the-art rendering techniques in sample efficiency given a large number of canonical samples. However, in interactive applications, where there is a limited budget for samples, such sample sizes are infeasible. Furthermore, in scenes where cameras, lighting, and object instances are dynamic, many of the accumulated temporal samples will need to be discarded. Techniques like temporal reprojection mitigate the loss of such information to a certain extent, but the fundamental issue remains – reuse in screen space lacks spatial persistence. The obvious solution to this is to resort to some sort of world space reuse.

The characteristics of a world space resampling algorithm are that it stores reservoirs in scene space, rather than screen space. As a result, each reservoir can amass samples from their respective regions in space; depending on the position of the camera with respect to this region, a reservoir may receive many more canonical samples per frame than it would in ReSTIR. Furthermore, world space reuse lends itself well to recursive resampling, where indirect intersections can query the temporal reservoir at their location and perform reuse again.

The design choices for world space reuse revolve primarily around the organization of reservoirs in space and efficiently accumulating samples. In the following subsections, we provide two variations of World Space Spatio-Temporal Reuse (WSSR). The first algorithm uses a static k-d tree to efficiently store reservoirs in a hierarchical manner, while the second utilizes a dynamic uniform grid to simplify querying and updating its reservoirs. In both algorithms, the samples we reuse store the emission from the sampled light source; see Algorithm 2.

Algorithm 2 Reservoir sample for WSSR

```
struct SAMPLE
    L_e      ▷ Emission from the selected light source
    p̂        ▷ Value of p̂ on this sample
    n        ▷ Light surface normal
    x        ▷ Location of the light sample
end struct
```

4.1 WSSR Using k-d Trees

The hierarchical nature of k-trees allows it to preserve detail in regions of the scene that are geometrically complex. Reservoirs are stored at leaf nodes of the tree and during the sampling stage, the tree is traversed to find the corresponding temporal reservoir, which is updated with new samples for temporal reuse. During the spatial reuse phase, neighboring cells are sampled stochastically, as with ReSTIR. Algorithm 3 outlines this process. The reservoir returned by WSSR_KD will be used to compute the final shading at the intersection point.

Algorithm 3 WSSR procedure for k-d trees

```
1:  procedure WSSR_KD(TREE, x, HIT, L)
2:      NODE ← TRAVERSE(TREE, x)        ▷ NODE is a leaf
3:
4:      R ← NODE.R
5:      X ← SAMPLE(HIT)
6:      UPDATE(R, X, p̂(X)/p(X))
7:
8:      NEIGHBORS ← {}
9:      for i ∈ [1, SPATIAL_SAMPLES] do
10:         N ← NEIGHBOR(NODE, L)  ▷ Randomly select a neighbor, see Section 4.1.1
11:
12:         NEIGHBORS.APPEND(N.R)
13:     end for
14:
15:     R′ ← MERGE(R_i ∈ NEIGHBORS)
16:     return R′
17: end procedure
```

The drawback of using k-d trees beside their inherent complexity includes non-trivial support for efficiently inserting new samples. As entities in a scene move, some nodes of the tree cease to be sampled, and new nodes must take their place. Updating the tree with new samples and pruning older ones is out of the scope of this work, and we instead use a static k-d that is initialized from an initial set of primary intersection points collected in the first frame. Additionally, this algorithms requires a large amount of sample "exploration" before the contributions each reservoir picks up because significant. Even with adequate exploration, the rendered images present largely noticeable quantization artifacts (Figure 5). This can be mitigated to some extent, as the next subsection will explain, but the issue still persists.

4.1.1 Effective Spatial Sampling

Special consideration must be given to performing spatial sampling. Naively sampling neighboring nodes and merging their reservoirs can easily lead to undesired quantization artifacts. In world space reuse, this occurs most commonly when the intersections corresponding to reservoir R are merged into the final shading reservoir. The solution to resolving part of the quantization is to omit R when merging spatial reservoirs into R'; this is done in Line 15 of Algorithm 3.

(a) Ground truth (b) Monte Carlo: PSNR ≈ 10.53 dB (c) Reprojection: PSNR ≈ 12.42 dB

Figure 4: Minor improvements in indirect lighting using reprojection (10 bounce, 16 spp)

To stochastically sample neighboring nodes N, we traverse up the k-tree for L levels, and randomly cascade down to a leaf node. The resulting algorithm for neighbor sampling is shown in Algorithm 4.

Also leaves less grains on the image...

Algorithm 4 Sampling neighboring nodes

```
procedure NEIGHBOR(NODE, L)
    N ← NODE
    l ← 0

    while l < L do
        if N.PARENT = ∅ then
            break
        end if
        N ← N.PARENT
    end while

    while not N.LEAF do
        N ← RANDOM_BRANCH(N)   ▷ Selects a random, valid child of N
    end while

    return N
end procedure
```

The size of L varies between scenes. However, as Figure 6 shows, we observed that low ascension depth leads to more noticeable quantization effects, while high ascension depth severely restricts sample reuse.

We also saw that persistent spatial reuse – in which R' in Algorithm 3 is assigned back to R – does not noticeably improve results. This is because every reservoir in the tree already undergoes spatial reuse by collecting samples from intersections (Line 6, Algorithm 3).

4.2 WSSR Using Grids

The complexity of k-d trees makes them too rigid for a task that requires preserving temporal and spatial locality. Uniform grids, on the other hand, are much simpler to work with, and carry more efficient query and update procedures.

Similarly to k-d trees, each cell in the grid stores a reservoir. At each frame, samples are taken at primary intersection locations, and these are used to update the reservoir at the corresponding cell, assuming that the position is in the grid's range. The major distinction with this algorithm is that we center the grid at the camera location so that the grid follows its motion. Because our samples are agnostic to the location from where they are sampled, we do not need to throw away samples that were evaluated at different times or positions. Stochastic spatial reuse follows the same idea of neighbor sampling, replacing the cascading procedure by simply selecting a random cell in a radius L. Figure 7 shows the result of this.

We considered storing multiple reservoirs per cell, in order to diversify the direct lighting samples used for shading at each frame. Each canonical sample would update the closest reservoir in the cell, essentially creating a Voronoi partition in space. However, we found that for the most part this is equivalent to using a one reservoir per cell in a grid with a higher resolution, and in some cases even performed worse.

4.2.1 Reprojection

Uniform grids by themself lack the range that k trees provide. Fortunately, we can use a similar reprojection hack used in Section 3.1. Instead of projection intersection points onto the film, we project them onto the grid. The corresponding cell index is used to proceed with the reuse algorithm. This works well because of the position-agnostic nature of light emission, as shown by the result in Figure 8.

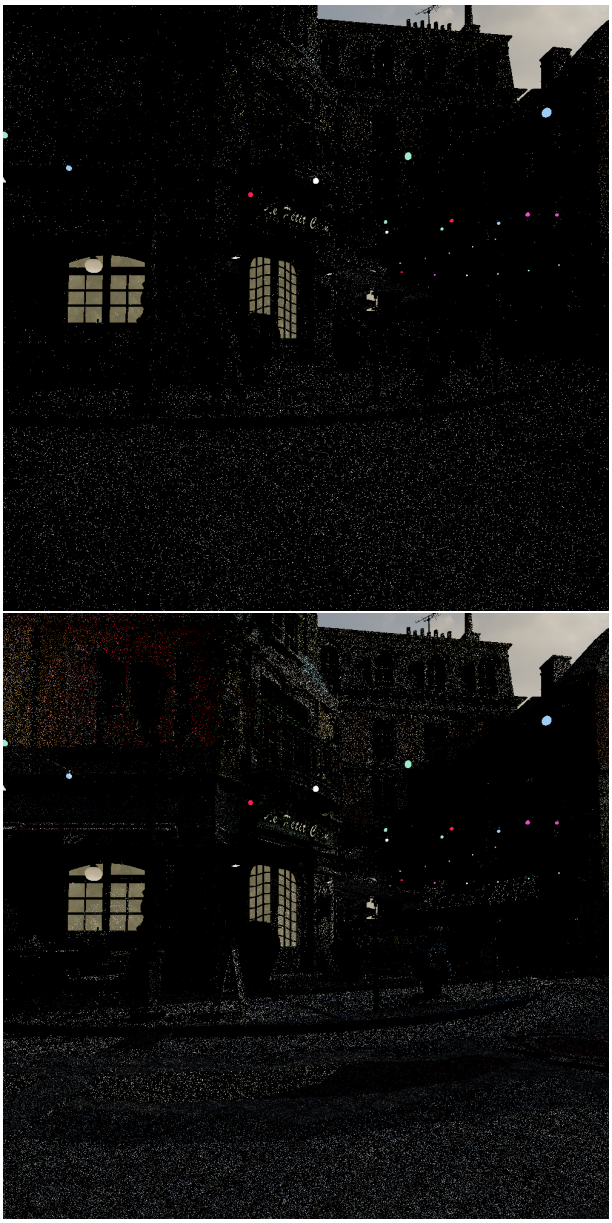


Figure 5: While WSSR using k-d trees (bottom) indeed gathers more significant direct lighting samples than ReSTIR (top), it suffers from quantization artifacts (16 spp, direct lighting only)

5 Implementation Choices

We implemented ReSTIR and the WSSR algorithms in a GPU-based real-time rendering system [1] built using NVIDIA OptiX [Ludvigsen and Elster, 2010] for hardware acceleration on an RTX 3060. Certain modifications have been made to these algorithms in order to make them more suitable for interactive applications. We will describe these modifications and detail other parameters in this section.

ReSTIR. At each frame, we evaluate $M = 32$ canonical sample per intersection, which we feed to the temporal reservoir, and then perform spatial sampling with $k = 5$ neighboring reservoirs in a radius of $0.1 \cdot \min(\text{WIDTH}, \text{HEIGHT})$. We use double-buffering for the reservoirs to prevent data races.

ReSTIR with reprojection. We use a similar configuration as our ReSTIR implementation. However, because we perform reuse at indirect intersections, we limit the number of spatial samples to $k = 3$.

WSSR using k-d trees. Our rendering pipeline for world-space reuse with k-d trees includes the one-time construction of the tree, after which we proceed with spatio-temporal reuse at intersections as previously described. The tree construction uses the position buffer gathered in the first frame to construct a static k-d tree; this is currently done on the CPU. After the sampling stage, reservoirs in the tree are updated atomically. Spatial reservoirs, sampled using $L = 10$, are merged into a thread-local reservoir to prevent numerical errors, and this reservoir is used to compute the final shading.

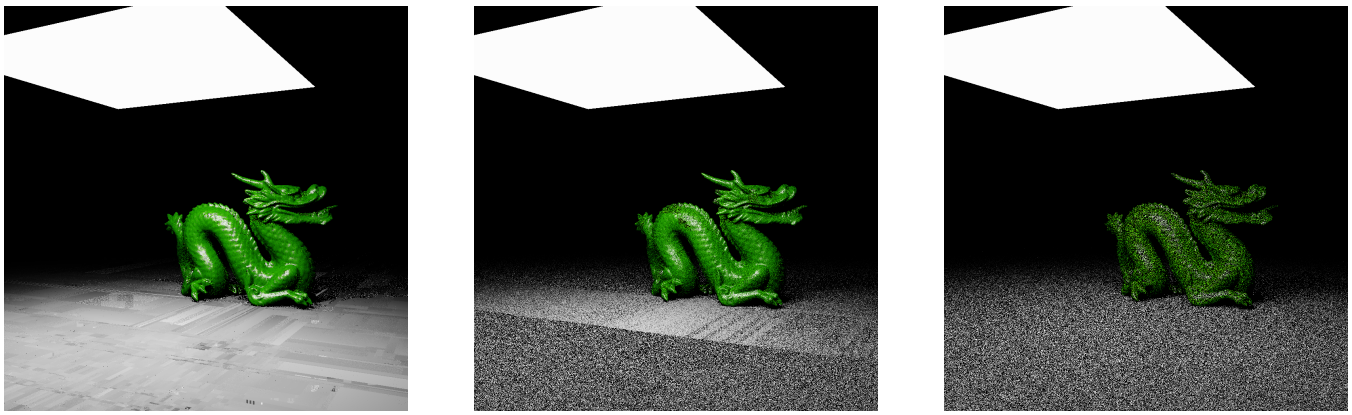
WSSR using grids. The pipeline for this algorithm is split into a sampling stage and an evaluation stage. During the sampling stage, primary intersection points generate direct lighting samples which are collected into an image buffer. These samples are concurrently processed to be updated into their corresponding reservoirs. For sake of efficiency, we limit the number of samples a reservoir can receive to $M = 100$. After updating the reservoirs – which equates to temporal reuse – we perform spatial reuse in the evaluation stage. We sample $k = 3$ spatial neighbors in a radius of $L = 10$ and merge them into a thread-local reservoir, which is used to compute the final direct lighting.

Note that for the WSSR algorithms, we resample direct lighting at indirect intersections as well.

6 Conclusion

In this work, we introduced a variation of the original ReSTIR paper which operates on world space rather than screen space. The resulting algorithms are better able to retain significant samples even when objects are dynamic with respect to the camera. World Space Spatio-Temporal Reuse with Grids, in particular shows, promise for future work.

[1] Kobra, `https://github.com/vedavamadathil/kobra`

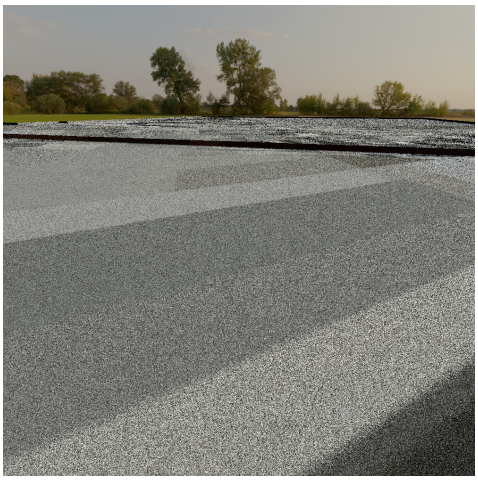


(a) $L = 1$; High quantization (b) $L = 10$; Best of both worlds (c) $L = 100$; Low resampling rate

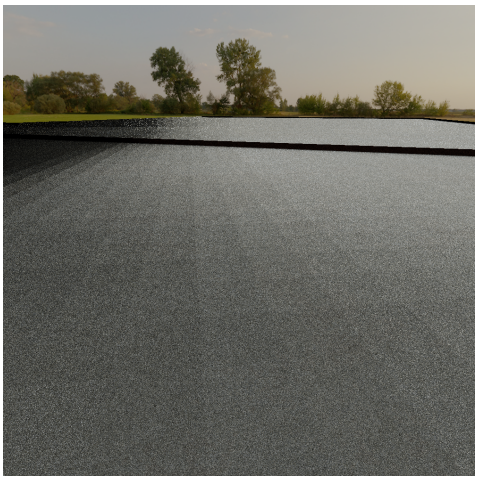
Figure 6: Effectiveness of spatial reuse at varying “radii"

Figure 7: World Space Spatio-Temporal Reuse with Uniform Grids (16 spp, direct lighting only)

Figure 8: WSSR with Grids improved with reprojection (16 spp, direct lighting only)



(a) WSSR using k-d trees; yields visible quantization artifacts



(b) WSSR using grids; displays odd shading at boundaries of the grid as well as aliasing artifacts at cell edges

Figure 9: Limitations of the WSSR techniques

Limitations. The WSSR algorithms we presented render undesirable artifacts in various scenes (Figure 9). Related work including [Boissé, 2021] resolved similar artifacts by jittering the world space coordinates of the intersections points, but we found this only degraded resampling rates. Furthermore, these algorithms carry some bias, even when ignoring these artifacts. This is an issue we were unable to tackle in the scope of this project. Lastly, as with many three-dimensional partition structures, both WSSR algorithms consume large amounts of memory. We plan to address all of these shortcomings in future work.

Future work. While we have focused on direct lighting in this paper, we believe that world space algorithms will work well for indirect lighting. In fact, [Boissé, 2021, Boissé et al., 2022] have shown that this is possible and that the results are competitive with ReSTIR Path Tracing [Lin et al., 2022]. In future work, we wish to explore novel target functions to improve the effectiveness of resampling and ways in which position-dependent samples – paths for indirect lighting – can be preserved as their source locations move out of their reservoir's region. We expect that with additional engineering, our world space reuse algorithm can bring more desirable results than the aforementioned works.

References

[Bitterli et al., 2020] Bitterli, B., Wyman, C., Pharr, M., Shirley, P., Lefohn, A., and Jarosz, W. (2020). Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (TOG)*, 39(4):148–1.

[Boissé, 2021] Boissé, G. (2021). World-space spatiotemporal reservoir reuse for ray-traced global illumination. In *SIGGRAPH Asia 2021 Technical Communications*, pages 1–4.

[Boissé et al., 2022] Boissé, G., Meunier, S., de Dinechin, H., Oliver, M., Bartels, P., Veselov, A., Eto, K., and Harada, T. (2022). Gi-1.0: A fast scalable two-level radiance caching scheme for real-time global illumination.

[Chaitanya et al., 2017] Chaitanya, C. R. A., Kaplanyan, A. S., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai, D., and Aila, T. (2017). Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):1–12.

[Lin et al., 2022] Lin, D., Kettunen, M., Bitterli, B., Pantaleoni, J., Yuksel, C., and Wyman, C. (2022). Generalized resampled importance sampling: foundations of restir. *ACM Transactions on Graphics (TOG)*, 41(4):1–23.

[Ludvigsen and Elster, 2010] Ludvigsen, H. and Elster, A. C. (2010). Real-time ray tracing using nvidia optix. In *Eurographics (Short Papers)*, pages 65–68.

[Majercik et al., 2019] Majercik, Z., Guertin, J.-P., Nowrouzezahrai, D., and McGuire, M. (2019). Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques Vol*, 8(2).

[Ouyang et al., 2021] Ouyang, Y., Liu, S., Kettunen, M., Pharr, M., and Pantaleoni, J. (2021). Restir gi: Path resampling for real-time path tracing. In *Computer Graphics Forum*, volume 40, pages 17–29. Wiley Online Library.

[Pharr et al., 2016] Pharr, M., Jakob, W., and Humphreys, G. (2016). *Physically based rendering: From theory to implementation*. Morgan Kaufmann.

[Schied et al., 2017] Schied, C., Kaplanyan, A., Wyman, C., Patney, A., Chaitanya, C. R. A., Burgess, J., Liu, S., Dachsbacher, C., Lefohn, A., and Salvi, M. (2017). Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*, pages 1–12.

[Talbot, 2005] Talbot, J. F. (2005). *Importance resampling for global illumination*. Brigham Young University.