Worldspace ReSTIR for direct illumination

Storing precomputed reservoir values with a normal-aware hashgrid

Vlad T. Stefanescu
Supervisors: Michael Weinmann, Elmar Eisemann, Christoph Peters
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2025

Name of the student: Vlad T. Stefanescu
Final project course: CSE3000 Research Project
Thesis committee: Michael Weinmann, Christoph Peters, Georgios Smaragdakis

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

This thesis addresses the challenge of initial frame noise in real-time ray tracing when using ReSTIR. We propose and evaluate an approach that integrates a normal-aware hash grid for precomputed reservoir caching to improve direct illumination. The research investigates how reservoir caching enhances visual quality alongside ReSTIR and analyzes the associated trade-offs in memory usage and performance. Our contribution includes the implementation and analysis of this caching strategy, assessing its impact on visual fidelity across diverse scenes. Although the method significantly reduces noise and improves initial sampling convergence, it can introduce visible grid artifacts in scenes with many light overlaps. Furthermore, this approach incurs notable memory overhead and increased frame times. This work demonstrates the potential of normal-aware hash grids for ReSTIR improvements, providing a proof-of-concept algorithm for stable, high-quality initial samples.

1 Introduction

Ray tracing simulates the physical behavior of light to generate photorealistic images, making it central to modern computer graphics. Despite its visual fidelity, ray tracing has historically been too computationally expensive for real-time applications such as video games and interactive simulations. This bottleneck has limited its widespread adoption in contexts where both speed and quality are required as late as 2017 [4].

Recent advances in hardware and rendering algorithms have begun to change this landscape. A particularly impactful development is Reservoir-based Spatiotemporal Importance Resampling (ReSTIR) [2]. Directly computing the contribution of light sources in complex lighting environments is computationally infeasible [4]. To handle this, Monte Carlo integration is used to approximate the lighting by sampling. However, it typically requires many samples, and it is difficult to predict which samples will be effective. ReSTIR addresses this by using resampled importance sampling [9] and reusing samples from previous frames and neighboring pixels to improve efficiency and reduce noise. These samples are stored in reservoirs, a data structure that maintains a representative sample from a potentially large number of inputs. ReSTIR has enabled real-time ray tracing with significantly improved visual quality and computational efficiency. However, its reliance on temporal data leads to a critical limitation: the technique performs poorly in the initial frames of a sequence, when no historical lighting information is available. This results in high visual noise and poor image quality.

To address this limitation, this research investigates the effectiveness of a technique called *reservoir caching*, implemented using a hash-grid structure that allows reservoirs to be merged with pre-computed ones, thereby enabling the selection of higher-quality samples. This approach aims to reduce noise and improve image quality. Similar approaches have been explored, such as the work by [3] and [10] who used grid-based reservoirs to effectively manage many lights for global illumination when sampling secondary rays, although with increased memory usage. However, there remains a gap in understanding how cached reservoirs can be reused effectively for direct illumination.

This leads to the **research question** of this paper regarding how reservoir caching improves quality in real-time ray/path tracing when used alongside ReSTIR, and what are the associated trade-offs in terms of memory usage and performance.

Contribution. This work implements and analyzes a reservoir caching strategy designed to improve ReSTIR sampling. We evaluate how the use of cached reservoirs affects visual

fidelity. Our experiments span two scenes, exploring the method's benefits in different situations. We analyze performance trade-offs and identify where caching provides the most benefit.

The remainder of this paper is organized as follows: Section 2 reviews the background and related work; Section 3 describes the specifics of the hash grid; Section 4 explains the implementation choices; Section 5 presents the results; Section 6 addresses responsible research considerations; Section 7 discusses the findings; and Section 8 concludes the paper and outlines directions for future work.

2 Background and Related Work

2.1 Ray Tracing and Direct Illumination

Ray tracing is a rendering technique that aims to realistically simulate the behavior of light in a virtual environment. It operates by casting rays from the camera through each pixel of the image plane into the scene. For the purposes of this research, our focus is exclusively on modeling direct illumination. This means that we consider only the light that travels directly from a light source to a surface point, without accounting for secondary light transport effects such as reflections, refractions, or global illumination. When a ray intersects a surface, the algorithm determines the contribution of direct light from all light sources visible from that surface point to accurately calculate its color and shading. The direct illumination L_o at a surface point $\mathbf{x}$ in direction ω_o is given by the following equation [6]:

$$L_o(\mathbf{x}, \omega_o) = \int_\Omega V(\mathbf{x}, \mathbf{x}_s)\, f_r(\mathbf{x}, \omega_i, \omega_o)\, L_e(\mathbf{x}_s, -\omega_i)\, |\omega_i \cdot \mathbf{n}|\, d\omega_i \tag{1}$$

where V represents the visibility between two points, f_r is the BRDF, $L_e(\mathbf{x}, \omega_o)$ is the emitted radiance at point $\mathbf{x}$ in direction ω_o, $\mathbf{x}_s$ is the point on the light source in the direction ω_i, $\mathbf{n}$ is the surface normal at $\mathbf{x}$, and Ω is the hemisphere above the surface of incoming directions to point $\mathbf{x}$.

2.2 Monte Carlo integration and Importance Sampling

Monte Carlo integration operates by selecting numerous random samples from the integral domain. The integrand is then evaluated at each of these sample points. The average of these evaluated values is taken as an estimate of the value of the entire integral. Importance sampling improves Monte Carlo integration by drawing samples from a distribution that emphasizes important regions of the integrand. These regions are places where the integrand has higher values. This approach reduces variance by weighting the integrand values according to the chosen sampling distribution. The integral is estimated as the average of these weighted values:

$$\int f(x)\, dx \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}, \tag{2}$$

where x_i are samples drawn from the distribution $p(x)$. It helps if $p(x)$ is similar in shape to $f(x)$.

2.3 Resampled Importance Sampling

Resampled Importance Sampling (RIS) [9] is a technique for approximating integrals. It uses two probability density functions (PDFs): a source distribution p, and a target distribution $\hat{p}$. In practice $\hat{p}$ is not a distribution as it does not need to be normalized but for the purposes of this paper it will be called as such. The method is useful when p is easy to sample from but poorly matches the shape of the integrand f, while $\hat{p}$ is a better approximation of f but more difficult or expensive to sample from directly.

RIS works by drawing M samples x_i from the source distribution p, and then resampling or weighting them based on how well they align with the target distribution $\hat{p}$:

$$w(x) = \frac{\hat{p}(x)}{p(x)} \tag{3}$$

and using these weights to obtain good samples according to the probability mass function:

$$q(z) = \frac{w(z)}{\sum_{i=1}^{M} w(x_i)} \tag{4}$$

This produces a set of samples that can be used effectively in importance sampling to approximate the integral of f. Using only one sample z we can estimate the integral using:

$$\int f(x)\,dx \approx \frac{f(z)}{\hat{p}(z)} \frac{1}{M} \sum_{i=1}^{M} w(x_i) \tag{5}$$

This approach effectively leverages the shape of $\hat{p}$ while avoiding the need to sample from it directly. The RIS estimator is unbiased if at least one sample ($M > 0$) is used and both p and $\hat{p}$ are strictly positive wherever the integrand f is non-zero [9].

Algorithm 1: Weighted Reservoir Sampling [2]

```
class Reservoir:
    y ← 0                                   // The output sample
    w_sum ← 0                               // Sum of weights
    M ← 0                                   // Number of samples seen so far
    W                                       // The sample contribution factor
    function update(x_i, w_i)
        w_sum ← w_sum + w_i
        M ← M + 1
        if rand() < (w_i / w_sum) then
            y ← x_i
function reservoirSampling(S)
    Reservoir r
    for i ← 1 to M do
        r.update(S[i], weight(S[i]))
    return r
```

Algorithm 2: Streaming RIS using weighted reservoir sampling [2]

foreach *pixel* $q \in$ *Image* **do**
  Image[q] $\leftarrow$ `shadePixel` (`RIS` (q), q)
function `RIS`(q)
  Reservoir r
  for $i \leftarrow 1$ **to** M **do**
    generate $x_i \sim p$
    r.update($x_i, \hat{p}_q(x_i)/p(x_i)$)
  $r.W \leftarrow \frac{1}{\hat{p}_q(r.y)} \cdot \frac{1}{r.M} \cdot r.w_{sum}$ `// Equation 5`
  return r
function `shadePixel`(*Reservoir* r, q)
  return $f_q(r.y) \cdot r.W$

Algorithm 3: Combining the Streams of Multiple Reservoirs [2]

Input: Reservoirs $r_1, r_2, \ldots, r_k$ to combine
Output: A combined reservoir equivalent to the concatenated input streams of $r_1, \ldots, r_k$

function `combineReservoirs`($q, r_1, r_2, \ldots, r_k$)**:**
  Reservoir s
  foreach $r \in \{r_1, \ldots, r_k\}$ **do**
    s.update($r.y, \hat{p}_q(r.y) \cdot r.W \cdot r.M$)
  $s.M \leftarrow r_1.M + r_2.M + \cdots + r_k.M$
  $s.W \leftarrow \dfrac{1}{\hat{p}_q(s.y)} \cdot \dfrac{1}{s.M} \cdot s.w_{sum}$ `// Equation 5`
  return s

2.4 ReSTIR

ReSTIR [2] is built directly on top of RIS. It uses a reservoir to store a single representative light sample along with its weight, the sum of all sample weights, and the number of samples considered as shown in Algorithm 1. New samples can be added incrementally, with the reservoir maintaining the correct statistical distribution as if all samples were retained. In addition, multiple reservoirs can be merged into one, effectively combining their sample data as shown in Algorithm 3.

In each frame, multiple light samples are generated for every pixel's shading point and added to that pixel's corresponding reservoir using streaming RIS (Algorithm 2). An optional visibility pass may then be performed to verify whether the selected samples are valid. During the spatial reuse pass, these reservoirs are merged with those of neighboring pixels to enhance sampling density. In the temporal reuse pass, reservoirs are further combined with those from the previous frame, allowing a small number of samples to approximate a much larger effective sample set. Finally, during shading, Equation 5 is used to compute the final color of the pixel. This is described in Algorithm 4.

Algorithm 4: RIS with spatiotemporal reuse [2]

Input: Image sized buffer containing the previous frame's reservoirs
Output: The current frame's reservoirs

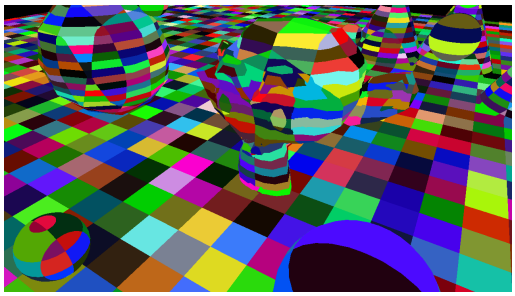
function *reservoirReuse(prevFrameReservoirs)*
  reservoirs ← new Array[ImageSize]
  `// Generate initial candidates`
  foreach *pixel* $q \in$ *Image* **do**
    reservoirs[q] ← RIS(q)
  `// Evaluate visibility for initial candidates`
  foreach *pixel* $q \in$ *Image* **do**
    if *shadowed(reservoirs[q].y)* **then**
      reservoirs[q].W ← 0
  `// Temporal reuse`
  foreach *pixel* $q \in$ *Image* **do**
    q' ← pickTemporalNeighbor(q)
    reservoirs[q] ← combineReservoirs(q, reservoirs[q], prevFrameReservoirs[q'])
  `// Spatial reuse`
  for $i \leftarrow 1$ **to** n **do**
    foreach *pixel* $q \in$ *Image* **do**
      Q ← pickSpatialNeighbors(q)
      $R \leftarrow \{\text{reservoirs}[q'] \mid q' \in Q\}$
      reservoirs[q] ← combineReservoirs(q, reservoirs[q], R)
  `// Compute pixel color`
  foreach *pixel* $q \in$ *Image* **do**
    Image[q] ← shadePixel(reservoirs[q], q)
  return reservoirs

2.5 Related works

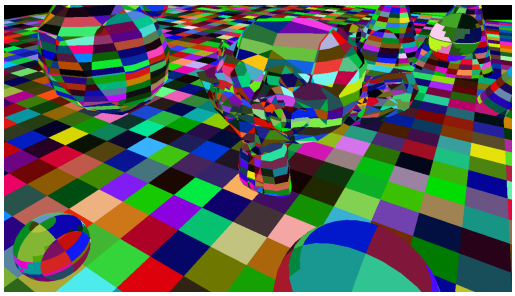
Boksansky et al. [3] introduced a method for real-time global illumination using grid-based reservoirs organized in an array structure. In their approach, the center of each grid cell is used as a representative shading point for light sampling, which simplifies data access and spatial reuse. However, their method does not account for geometric features like surface orientation, which can limit sample quality near boundaries or in highly detailed scenes.

Zhang et al. [10] extended spatiotemporal reuse in path tracing by introducing a normal-aware hash grid to store and access secondary path samples more effectively. Their method uses surface normals in conjunction with spatial position to improve neighborhood matching, reducing artifacts and increasing reuse efficiency. While effective for indirect lighting and secondary bounces, their technique was not applied to direct illumination.

In contrast, our work leverages the benefits of normal-aware hash grids specifically for direct illumination, enabling more accurate and feature-aware reuse in primary visibility. By incorporating surface orientation into the grid structure, we improve the spatial fidelity of reservoir sampling, especially in scenes with high geometric complexity.



(a) Grid without normal binning: spatial cells only.



(b) Grid with normal binning: spatial-directional cells.

Figure 1: Visualization of the hash grid structure. The spatial grid (a) is extended with directional bins (b) to account for surface normal orientation, yielding finer partitions of sample space.

3 Hash grid

We render static scenes illuminated by triangle area lights, assuming Lambertian diffuse materials and focusing solely on direct illumination. Our approach builds on ray tracing, where rays are cast from the camera into the scene. When a ray intersects a triangle, the shading point, i.e. the intersection point, is computed, and the rendering equation (Equation 1) is evaluated at that location. Given the high computational cost of this process, we employ importance sampling in combination with ReSTIR, enhanced through a hash grid-based reservoir caching strategy to improve quality. This hash grid is precomputed and then used during the initial sampling step of ReSTIR.

3.1 Hash Grid Construction

The scene is discretized into a uniform 3D grid over the spatial domain as can be seen in Figure 1a. To incorporate directional information, each spatial cell is further divided across a discretized representation of the sphere of surface normals. This is achieved by uniformly partitioning the polar angle $\theta \in [0, \pi]$ and the azimuthal angle $\phi \in [0, 2\pi)$ into a fixed number of intervals, yielding a regular angular binning of the sphere. This can be seen in the middle of the monkey in Figure 1b. Each combined (spatial, directional) cell corresponds to a unique pair of spatial location and normal orientation bin.

A large number of surface samples are generated prior to reservoir construction. Triangle primitives are sampled proportionally to their surface area, and a point is drawn uniformly over the selected triangle. The surface normal at the sampled point is used to compute its corresponding directional bin via its spherical coordinates (θ, ϕ), and the sample is assigned to the appropriate cell's sample pool.

Each spatial-directional cell in the hash grid stores a fixed number of independently constructed light sample reservoirs. These reservoirs are built directly during a preprocessing pass, where we use surface samples assigned to each cell to generate light samples. For each reservoir, multiple surface points are randomly selected from the cell's sample pool. At each point, a light source is chosen and sampled, and the resulting sample is evaluated using the rendering equation to compute a ReSTIR-compatible importance weight. This light sample is then incorporated into the reservoir using standard one-sample RIS. Repeating

this process yields a set of representative, weighted light sample reservoirs per cell, capturing the spatial and angular lighting distribution across the scene. These prefilled reservoirs form the cache that is later used to guide initial sampling during rendering.

3.2 Using the Hash Grid

To improve the quality and stability of the initial sampling stage in ReSTIR, we extend the standard RIS approach by incorporating information from the precomputed hash grid of reservoirs. At each visible surface point, instead of generating an initial reservoir purely from local light sampling, we first gather a set of reservoirs from the hash grid. These reservoirs correspond to the cell containing the surface point in the combined spatial and angular grid.

A number of reservoirs are selected at random from this cell and merged using Algorithm 3. This produces an intermediate reservoir that encapsulates statistical information from precomputed samples in similar spatial and angular contexts. Following this merge step, additional light candidates are sampled using standard RIS (as described in Algorithm 2) and incorporated into the same reservoir. The result is a hybrid initial sample that leverages both global precomputed information and local sampling. These enriched initial reservoirs are then passed to the spatiotemporal reuse phases of Algorithm 4, where they are further refined through screen-space and temporal reservoir merging.

4 Implementation Choices

We implemented both the baseline ReSTIR algorithm and our hash grid variant on the CPU. This decision was driven by project time constraints, as well as the desire for a direct performance comparison between the two methods. Our experiments are limited to scenes using only Lambertian diffuse materials.

4.1 ReSTIR Specifics

Candidate light samples are generated using streaming RIS with a fixed sample count of $M = 8$ as shown in Algorithm 2. Triangle area lights are selected uniformly, followed by uniform sampling of a point on the chosen triangle. As a result, the source probability density is given by $p(x) = 1/N_{\text{lights}} \cdot 1/A(x)$, where $A(x)$ is the area of the light triangle containing sample x, and N_{lights} is the total number of triangle lights.

The *target density* $\hat{p}(x)$ corresponds to the unnormalized integrand of the rendering equation (excluding the visibility term) $\hat{p}(x) = L_e \cdot f_r \cdot |\omega_i \cdot \mathbf{n}|$, where L_e is the emitted radiance, f_r is the BRDF (Lambertian in our case), and G is the geometry term. Because light samples are generated uniformly over surface area, the target density must also be expressed in area measure. Converting the source density instead (e.g., to solid angle measure) leads to visual artifacts and emphasizes the hash grid cell boundaries, which we aim to avoid.

We use a *temporal reuse cap* to prevent excessive growth in reservoir weights across frames. We cap the multiplicity M of the previous frame's reservoir before performing temporal reuse. Specifically, we enforce a maximum value of $M_{\text{prev}} = 10 \times M_{\text{curr}}$. This heuristic stabilizes temporal accumulation and aligns with the way reservoir weights are computed during merging, where the contribution of a sample is proportional to $\hat{p}_q(r.y) \cdot r.W \cdot r.M$,

and the original weight sum w_{sum} is ignored.

During the *spatial reuse* pass, we select $K = 8$ neighboring pixels for merging from a circle with a radius of 5 pixels centered around the reference point. These neighbors are filtered based on both spatial distance to the shading point and the angular difference between surface normals. This filtering improves the quality of the reused samples by favoring geometrically and directionally similar points.

4.2 Hash Grid Configuration

The hash grid is implemented as a 64^3 uniform 3D grid over scene space, extended with 64 directional bins to account for surface normal orientation. Each cell stores 64 independent reservoirs. During the preprocessing stage, each reservoir is filled using 1024 randomly selected surface samples from within the corresponding spatial-directional bin, as described in subsection 3.1. The same source and target distributions are used such that the reservoirs are directly compatible with the ReSTIR reservoirs. During the initial sampling stage, $X = 4$ reservoirs from the grid are merged before RIS is performed.

5 Experimental Setup and Results

5.1 Experimental Setup

For the experimental evaluation of our ReSTIR improvements using precomputed hash grid reservoirs, we employed two distinct test scenes designed to challenge different aspects of the sampling and reuse process. The first scene, referred to as City Grid, features a structured urban layout composed of buildings, streets, and sidewalks. Illumination is limited to the sidewalks, which are lit by streetlamps, creating direct lighting scenarios and occlusion patterns between the urban structures.

The second scene, referred to as Monkeys, consists of a large, flat plane populated with a diverse set of geometric primitives including spheres, cones, and several instances of the Suzanne mesh (Blender's stylized monkey head). The scene is illuminated by multiple spherical light sources positioned at varying heights and distances, each emitting different colored light. This setup produces a wide variety of shading conditions and overlapping lighting effects.

In addition to the ReSTIR and Hash Grid methods, a uniform sampling renderer is added to our comparison, which uses 1 uniform light sample per pixel. The reference images are using the 2000 uniform light samples per pixel.

The experiments were conducted on a Windows-based system equipped with an Intel Core i7-6700K with 16GB of RAM. The code was compiled using MSVC.

5.2 Qualitative Analysis

The rendering comparison in Figure 2 illustrates the effectiveness of different sampling techniques in the city grid scene, with particular emphasis on the performance of the Hash Grid method. Visually, the Hash Grid approach significantly reduces noise compared to Uniform Sampling and ReSTIR. The illumination on surfaces such as the sidewalk and lamp post appears much cleaner and more stable, more closely approximating the reference image. This

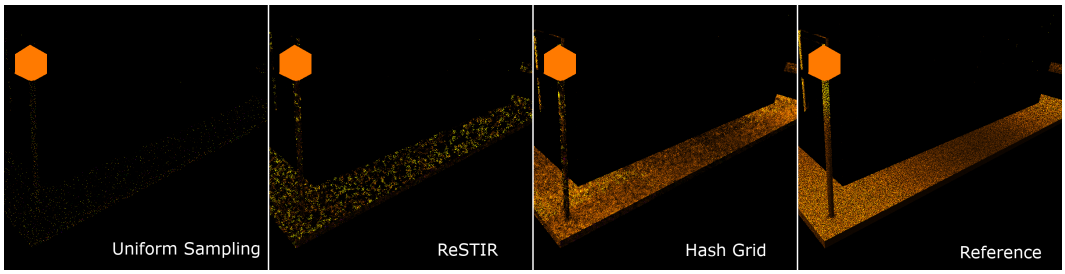


Figure 2: Comparison of rendering methods for the City Grid scene illuminated mainly by lanterns along the sidewalks. Shown are the first frames using 4 methods from left to right: uniform sampling, ReSTIR, hash grid, and reference rendering.

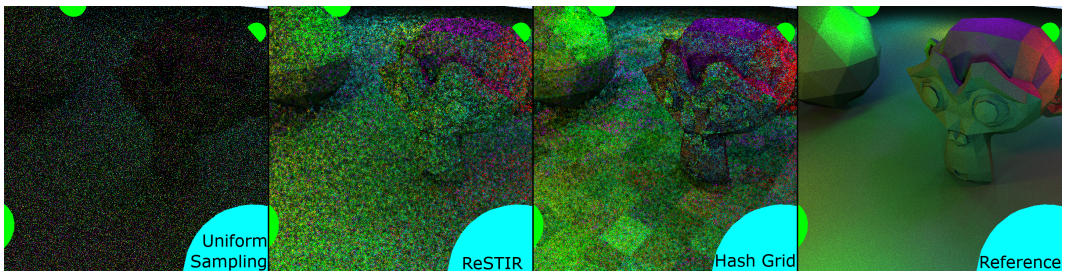


Figure 3: Comparison of rendering methods for the Monkeys scene containing Suzannes and geometric shapes illuminated by spheres. Shown are the first frames using 4 methods from left to right: uniform sampling, ReSTIR, hash grid, and reference rendering.

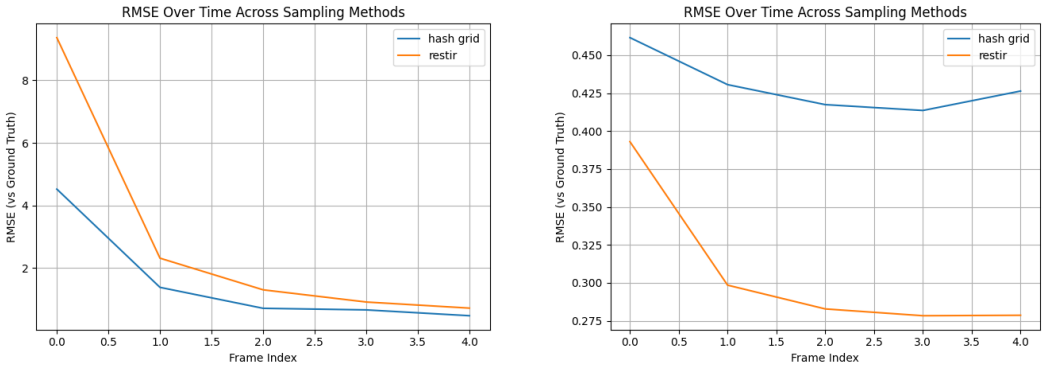
suggests that the Hash Grid effectively captures direct lighting while maintaining spatial coherence, resulting in a much higher quality visual output than base ReSTIR.

In contrast to the City Grid scene, the results of the Monkey Scene shown in Figure 3 reveal limitations of the Hash Grid approach. While it still performs better than Uniform Sampling and ReSTIR in terms of overall shape definition, the noise reduction is less effective, and a distinct artifact pattern emerges. The Hash Grid introduces visible grid-like structures across the surface, particularly noticeable on the ground plane, which breaks the illusion of physically plausible lighting. These artifacts likely stem from limitations of the grid's resolution and reservoir count per cell.

Despite these issues, the Hash Grid still offers better geometry clarity and material separation than ReSTIR, which exhibits high-frequency color noise and poor shading coherence. Uniform sampling, as expected, fails to capture any meaningful lighting structure due to extreme noise. Overall, while the Hash Grid improves detail recognition and lighting strength in this scene, it suffers from visible spatial artifacts that detract from the final image quality, highlighting the method's sensitivity to scene complexity and density variation.

5.3 Quantitative Analysis

In the City Grid scene (Figure 4a), the Hash Grid method initially achieves a lower RMSE compared to ReSTIR, suggesting superior convergence in early frames. However, ReSTIR's error rapidly decreases and nearly matches Hash Grid after a few frames, implying dimin-

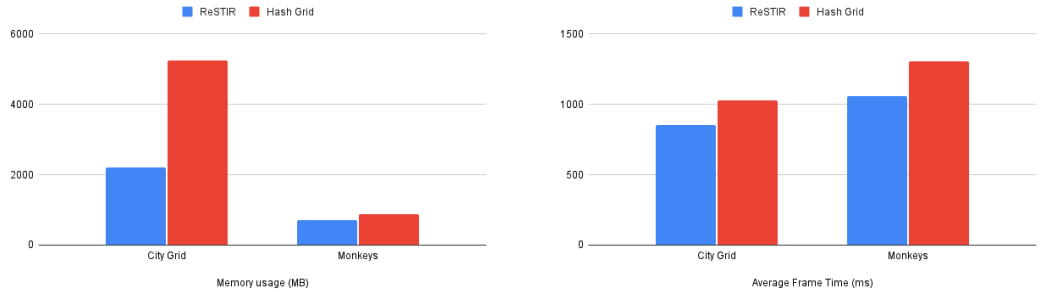


(a) City Grid

(b) Monkeys.

Figure 4: Error (root mean squared error) in the scenes over the first 5 frames.

ishing advantages for Hash Grid over time. On the other hand, in the Monkeys scene (Figure 4b), ReSTIR maintains a consistently lower RMSE than Hash Grid throughout the sequence, possibly due to the visible grid artifacts noted earlier.



(a) Memory usage (measured in MB) comparison by scene and method.

(b) Average frame time (measured in ms) comparison by scene and method.

Figure 5: Performance measurements.

The hash grid introduces a significant memory overhead, particularly in the city grid scene, where memory usage more than doubles, as illustrated in Figure 5a. This is likely due to the scene's high density, which limits the benefits typically gained from using a sparse grid. Conversely, in the Monkeys scene, the abundance of empty space results in much lower memory consumption by the grid. It should be noted that the reservoir structure is currently unoptimized, and memory consumption could be substantially reduced for both the ReSTIR and Hash Grid implementations. Nevertheless, the relative memory usage measurements presented here remain a valid and accurate comparison. Regarding performance, frame times increase by approximately 20 to 23 percentage points, as shown in Figure 5b. This increase can be attributed to the overhead of hash grid access times. Optimizing the hash function could potentially reduce this performance impact.

6 Responsible Research

This research project was conducted with careful consideration of both ethical aspects and the reproducibility of the methods employed. Throughout the development, we relied on several well-established open source libraries, including SDL [7], TinyBVH [1], and tinyobjloader [5]. By using these tools, which are distributed under permissive licenses, we ensured that our work respects intellectual property rights and contributes transparently to the existing body of knowledge. Proper citation and adherence to licensing conditions were strictly followed to maintain ethical standards.

The visual scenes used in this project were created by Rafayel Gardishyan, who kindly granted explicit permission for their use. This respectful collaboration highlights our commitment to ethical research practices by acknowledging and safeguarding the intellectual property of content creators. Additionally, the base ReSTIR project was developed jointly with Rafayel Gardishyan and Samuel Bruin.

To promote reproducibility, the full source code of this project has been made publicly available on GitHub [8]. This openness enables other researchers and practitioners to reproduce our results, verify our findings, and build upon the work. Along with the code, all necessary assets and scene files are included to ensure that the research environment can be replicated as closely as possible. By relying on widely used libraries with stable APIs, we also minimized dependency-related discrepancies that could affect reproducibility.

7 Discussion

This research demonstrates that integrating a normal-aware hash grid for reservoir caching can improve ReSTIR's initial sampling, addressing its early-frame limitations due to insufficient temporal data. Qualitatively, the Hash Grid method yielded cleaner, more stable illumination in the City Grid scene compared to ReSTIR, visually approximating reference images. This is quantitatively supported by lower early-frame RMSE values in the City Grid, confirming enhanced initial samples and accelerated convergence. This approach effectively fills a gap by applying normal-aware hash grids specifically to direct illumination, improving image quality by leveraging spatial and directional coherence.

However, critical trade-offs and limitations emerged. The Hash Grid method introduced distinct grid artifacts in the Monkeys scene, suggesting the fixed grid resolution or binning strategy may struggle with highly varied lighting. Furthermore, the implementation incurred significant memory overhead, particularly in dense scenes, and increased frame times due to hash grid access. These performance considerations indicate that while the method improves visual fidelity, its practical application requires careful attention to scene characteristics.

7.1 Limitations

Although the hash grid method demonstrates potential for enhancing geometric detail and light selection, it exhibits inherent bias. As such, it may be more suitable for use during the initial phases of rendering, with the possibility of being disabled in later stages. Due to time constraints, our analysis is limited to a CPU-based implementation of the algorithm. However, a GPU implementation is essential to fully evaluate its performance in real-time scenarios. Another limitation is the narrow selection of test scenes, which may not fully represent the method's effectiveness across broader use cases.

8 Conclusions and Future Work

Our work demonstrates the potential of a normal-aware hash grid for caching precomputed reservoirs to enhance ReSTIR's direct illumination capabilities. Our findings across City Grid and Monkeys scenes showed improved geometry clarity and lighting strength, particularly in mitigating initial frame noise. While the method showed promise in structured environments, its sensitivity to complex scenes resulted in grid-like artifacts, highlighting it's limitations. This outcome, alongside observed memory overhead and increased frame times, collectively provides a good proof-of-concept for the benefits of reservoir caching with a normal-aware hash grid for direct illumination. This is a good starting point for further analysis of this method and which cases are better suited to it.

Future research should explore updating hash grid values dynamically during rendering by integrating newly generated light samples into the reservoirs, effectively enabling temporal reuse in world space. Experimenting with adaptive grid structures and optimized hash functions is crucial to address artifacts and reduce performance overhead. Further investigation of the benefits of the method during camera movements could improve temporal stability. Additionally, optimizing the reservoir memory layout is essential to broaden applicability; this can be further extended to changing the structure from a hash grid to an octree. Finally, a GPU implementation of this algorithm is critical for exploring memory layout considerations and optimizing real-time performance.

References

[1] Jacco Bikker. tinybvh: Single-header bvh construction and traversal library. `https://github.com/jbikker/tinybvh`, 2025. Accessed: 2025-06-19.

[2] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 39(4), July 2020.

[3] Jakub Boksansky, Paula Jukarainen, and Chris Wyman. *Rendering Many Lights with Grid-Based Reservoirs*, pages 351–365. Apress, Berkeley, CA, 2021.

[4] Yangdong Deng, Yufei Ni, Zonghui Li, Shuai Mu, and Wenjun Zhang. Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques. *ACM Computing Surveys*, 50:1–41, 08 2017.

[5] Syoyo Fujita and contributors. Tinyobjloader: A tiny but powerful single-file wavefront .obj loader. `https://github.com/tinyobjloader/tinyobjloader`, 2016. Accessed: 2025-06-19.

[6] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. Association for Computing Machinery.

[7] SDL Development Team. Simple DirectMedia Layer (SDL). `https://github.com/libsdl-org`, 2025. Accessed: 2025-06-20.

[8] Vlad Stefanescu. Hash-grid. `https://github.com/vload/Hash-Grid`, 2025. GitHub repository.

[9] Justin Talbot, David Cline, and Parris Egbert. Importance Resampling for Global Illumination . In Kavita Bala and Philip Dutre, editors, *Eurographics Symposium on Rendering (2005)*. The Eurographics Association, 2005.

[10] Hangyu Zhang and Beibei Wang. World-space spatiotemporal path resampling for path tracing. *Computer Graphics Forum (Proceedings of PG 2023)*, 2023.