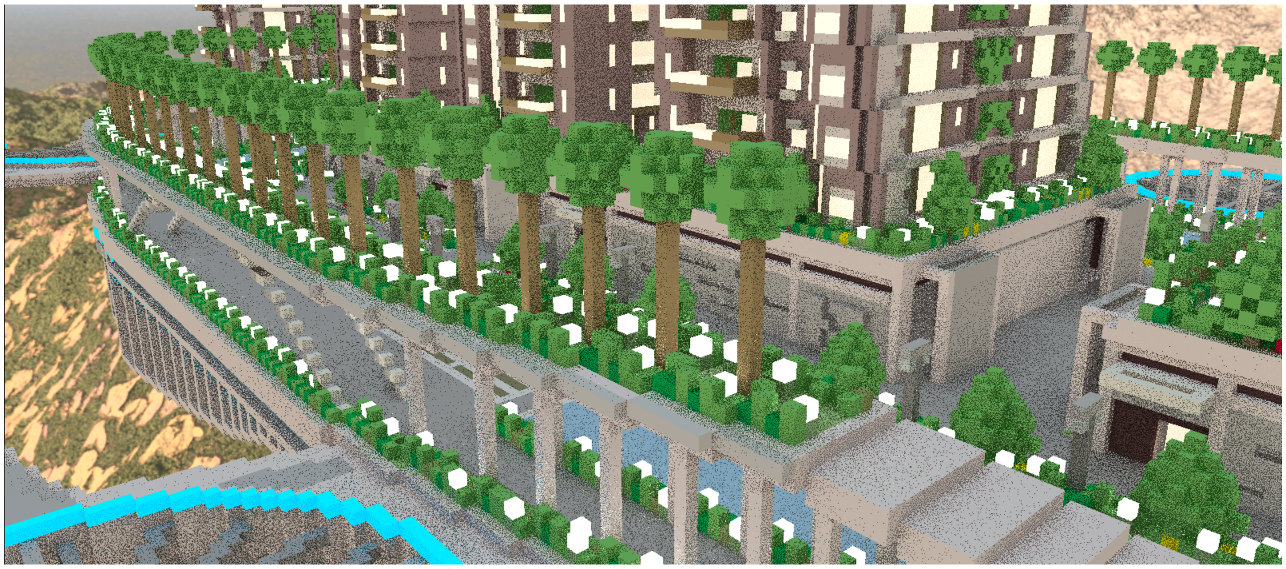
Utrecht University

Game & Media Technology
Master Thesis

The effectiveness of the ReSTIR technique when ray tracing a voxel world



Supervisor:
Dr. ing. J. (Jacco) Bikker

Author:
Xander Hermans

Second supervisor:
Prof. dr. ir. A.C. (Alex) Telea

July 12, 2022

ABSTRACT

Recent work by Bitterli et al. combines previous techniques such as weighted reservoir sampling and resampled importance sampling into an algorithm called ReSTIR capable of rendering of scenes illuminated by many dynamic lights in real-time. We introduce Voxel ReSTIR, an algorithm derived from ReSTIR tailored to voxel worlds, and compare it to ReSTIR. We explore the performance of the algorithms in different settings. We show that that both our point light and area light algorithms perform better in terms of speed and quality than ReSTIR in the same setting. Our algorithm is suitable for GPU implementation and we are able to achieve near real-time performance on consumer level hardware.

1 INTRODUCTION

Creating 3D games from scratch is often viewed as an overwhelming project. Both designing and creating the 3D world, as well as the logic it takes to implement the game mechanics are often nontrivial tasks. Especially for starting game developers.

Using game engines such as Unreal Engine or Unity alleviate this by providing tools which assist greatly in game development. Another approach could be the use of voxels to create the 3D world in the game. Doing so would simplify a lot of aspects of the development such as content creation, procedural content generation, game mechanics and rendering.

Voxel engines almost entirely eliminate the dependence on vector math, making them accessible to beginning programmers.

On the other hand, rendering of a voxel world is not trivial. This is specifically what we are interested in, as we want to look into efficient and intuitive rendering of a properly illuminated voxel game world containing many lights.

Performance is important as we aim to achieve real-time frame rates rendering our voxel world. The performance of a voxel based game scales with the size of the voxel grid. Using an acceleration structure to keep track of the voxels allows for larger worlds. Most voxel worlds are not randomly filled with voxels but consist of bigger blocks of the same voxel type throughout the grid. This can be exploited as it means we do not have to keep track of all individual voxels.

Nevertheless, the size of the grid is a limiting factor when it comes to memory to store it as well as cost to trace a ray through it. Thus, one can target a wider range of hardware easily by focusing on a smaller grid size. Since we are interested in reaching the current day consumer we focus on current day consumer level hardware. This means we will tune our parameters where necessary to be in line with the level of current day hardware.

Minecraft[36] is an example of a popular voxel based game. Minecraft is a simple sandbox game where the player is free to explore and build in the voxel world. It uses a low resolution voxel grid. This game traditionally uses rasterization to render the game world.

A popular modification for Minecraft's render is SEUS[49]. SEUS uses path tracing to achieve realistic lighting even with many lights as can be seen in Figure 1. Recently, a ray tracer renderer using DXR[54] was also added to Minecraft.

Rendering a voxel world to an image that can be shown on screen can be done in two ways. *Rasterization* and *ray tracing*.

Rasterization has been the more common approach not just for rendering voxel worlds but polygon worlds as well. To render a voxel world using rasterization one needs to build a polygon mesh representing the voxel world, every frame, before sending it to the GPU for rasterization. Algorithms for building such a mesh are far from easy to understand, or trivial to implement.

Figure 1: Picture of a scene with many lights using the SEUS render modification for Minecraft

Ray tracing, on the contrary, inherently deals with this visibility problem. When rendering a voxel world using ray tracing it only comes down to ray-grid traversal. The implementation for this traversal depends on the acceleration structure used to represent the grid.

Nevertheless, rasterization has been the more common approach because current hardware has better support for rasterization compared to ray-grid traversal or ray tracing in general.

However, recently consumer hardware specialised in ray intersection computations has become available turning the attention to ray tracing again and spurring new research into the topic. Without this specialised hardware using ray tracing is costly and often not achievable in real-time when aiming for graphical fidelity equal to or better than delivered by rasterization.

Various contributions to ray tracing techniques supplied by this recent research has brought real-time rendering solutions using ray tracing in reach even on previous consumer hardware that lacks the hardware support for ray intersection computations.

Rendering with many lights is a scenario one can encounter when rendering specific scenes. Such as the scene in Figure 1. Simulating these lights, which can be dynamic, is a tall task. Specifically in a voxel world where one can change large amount of voxels, including a large amount of lights, very quickly and even give the player the ability to do so in real-time.

Recent works, such as the ReSTIR method by Bitterli et al. [8] and the ReGIR method by Boksansky et al. [10], have significantly propelled the topic of efficient many light rendering. We are particularly interested in these techniques as they are created for and tested on polygon scenes, not voxel scenes.

Additionally, we want to explore options for efficiently sampling from many lights by exploiting the acceleration structure the voxel world is stored in.

In short, our aim is to render a dynamic voxel world with many dynamic lights in an intuitive and efficient manner. Rendering a voxel world without complex lighting is a well understood problem and while rendering polygon scenes with many dynamic lights has been an ongoing problem, several solutions have been put forward to solve this. Rendering a voxel world with many dynamic lights in real time remains somewhat unexplored and methods in the industry commonly depend on low quality approximations for lighting. We explore the ReSTIR method to solve the many light problem for voxel scenes, as this method is currently state of the art and performs best in many light scenes, but also struggles in certain scenarios inherent to a voxel world. Furthermore, when using voxel worlds we inherently have access to an acceleration structure containing the geometry. We explore ways to leverage this structure in combination with the chosen many lights method.

Therefore to efficiently and intuitively render a dynamic voxel world with many lights we:

- Explore the effectiveness of the ReSTIR method as solution for the many lights problem in voxel scenes.
- Explore exploits specific to the voxel acceleration structure to improve image quality in a performant manner.

2 RELATED WORK

In this chapter we will discuss previous work. We will discuss traditional methods for rendering voxel worlds in Section 2.1. After that we will discuss work related to ray tracing of voxels in Section 2.2 as well as the related topic of acceleration structures for efficient storage and ray tracing of voxel in Section 2.3. Next we discuss the many lights problem and various works proposing methods to deal with this problem in Section 2.4. Section 2.5 is about the concept of Resampled Importance Sampling and how this can be used to efficiently deal with many lights as well as some state of the art solutions using this concept. We touch upon the topic of global illumination briefly in Section 2.6 and conclude with a discussion in Section 2.7.

2.1 Rasterization of voxels

Displaying a voxel game world using rasterization involves building a polygon mesh of the game world. This is necessary because the render pipeline of common graphics hardware uses rasterization on triangles to create a 2D view of the 3D scene.

Traditional solutions for simulating light in a polygon scene often rely on mostly static geometry as well as not too many dynamic lights. These conditions need to be met because many of these solutions rely on precomputing an acceleration structure to efficiently determine light contribution at a point in the scene.

Shadow mapping [59], for example, uses the z-buffer to create shadow maps from the point of view for every light. An update of the world or lights would mean these maps would have to be recreated which means significant overhead which linearly scales with the amount of lights in the scene. For a sandbox world in which the user can place lights arbitrarily this means one would spend a large amount of extra rendering work even for the smallest of world updates.

More recent contributions [47] [40] [33] [51] allow for many lights to be simulated in a scene in real-time using methods similar to shadow mapping.

Other techniques such as light baking [45], that are used to create realistic lighting in polygon scenes, are less suited for large dynamic voxel scenes as they come with significant computation times and memory footprint.

Lastly, games such as Minecraft limit the range of the effect of each light in the scene thereby controlling the upper bound of rendering operations, however this results in a noticeable cut-off of the contribution of lights in the scene.

Building the polygon mesh of the voxel world as mentioned before is no trivial feat. Deciding on an algorithm for building this mesh involves deciding on the trade-off between a fast but naive algorithm which results in a large amount of faces or a slower algorithm which minimises the number of faces needed to represent the game world. This is important because the rasterization time positively scales with the amount of faces.

One must also consider the reuse of meshes and how such operations influence this when the world is updated. An efficient implementation of a mesh building algorithm called stb voxel render [5] can rasterize a voxel world of billions of voxels in real-time. Sadly, this is without any complex lighting.

Games which opt to use a voxel world are often after the sandbox element or aim to support a dynamic world. This means the world can drastically change over the lifetime of the game and even between two game updates. Reusing the meshes as mentioned in the previous paragraph becomes the exception rather than the norm.

In short, the simulation of light in the scene and the voxel to polygon scene conversion are problematical areas when rendering a voxel world. Solutions for these areas rely heavily on precompution when the dynamic nature of the voxel world prevents efficient reuse of the precomputed data. Using another approach such as ray tracing could prove to offer more intuitive solutions.

2.2 Ray tracing of voxels

Ray tracing voxels is a different approach to rendering a voxel world to a 2D image. The rendering pipeline for this approach consists mainly of ray-box intersections. The key benefits of ray tracing a voxel world are the simplicity of ray-box intersections and the fact that the voxel world itself acts as an acceleration structure. This means that by ray tracing we are inherently able to deal with the problems from Section 2.1 as no precomputations are needed and no separate acceleration structure needs to be maintained.

The rendering pipeline includes many stages but in this section we will focus on primary hits.

The simplest form of rendering a voxel world using ray tracing involves casting rays from the camera into the scene and determining for each ray what the closest voxel in the scene is with relation to the camera. This is the primary hit. For each primary hit the renderer plots the color of the voxel related to this hit on the screen. Doing so results in an image where no light contribution is considered. The computation time for this algorithm is dominated by the ray-scene intersection. This comes down to ray-box intersection testing and ray-grid traversal which have been explored by various works over the years [2] [31] [58] [46] [32]. For a recent efficient method and comparison of previous methods of ray-box intersections we would like to point the reader to Majercik et al. [32]. In this work the authors show that for large voxel worlds of up to 50M voxels and common screen resolutions their method, using ray tracing, out performs a traditional method, using rasterization, by 2x to 8x times. A novel but slightly unrelated idea offered in this paper that aids in this speedup comes from exploiting the rasterizer to efficiently approximate which pixels have voxels visible and should be ray traced.

Another work by Guehl [20] is about efficiently ray tracing billions of voxels. However, this method achieves this by use of specific data structures tailored to common hardware.

When extending the ray tracing algorithm to include lighting we face the same problems as mentioned before for rasterization of voxels. The visibility check can be done using methods similar to those mentioned in Section 2.1. These methods do not perform well for dynamic scenes.

Instead of using rasterization to determine visibility from lights in the scene, such as shadow mapping techniques, it is also possible to trace shadow rays. While tracing shadow rays is a more intuitive solution, which benefits from the relatively cheaper intersection techniques inherent to voxel worlds, it suffers from many of the same problems as shadow mapping. The method scales poorly with the amount of lights in the scene. It is also not possible to reuse the visibility information if the scene changes.

On the other hand, tracing shadow rays allows for more types of light sources whereas shadow mapping assumes every light source is of the point light type. Tracing shadow rays can also be done in a stochastic manner. Doing so means the complexity of simulation of light no longer depends on the number of lights. More on this in section 2.4 and 2.5.

2.3 Acceleration structures for voxel worlds

A recent work by Aleksandrov et al. [1] offers a review of different acceleration structures for voxels. It classifies such structures using a couple of properties. The review considers the acceleration structures as either static or dynamic based on the target frequency at which one would like to update the structure. Other properties include the type of geometry, the supported hardware architectures, the type of voxel date structure, attribute conversion and out-of-core support. The latter meaning one can go beyond the available memory as not the entire data structure needs to be kept in memory when performing operations on it.

Aleksandrov et al. [1] notes that each data structure has to compromise on access or compactness. Access meaning read and write access to individual voxels, and compactness meaning the size of the data structure in memory. Where static grid methods perform well for access, dynamic grid methods perform well in compactness and iterative access. For a voxel sandbox game we need a data structure that performs well in both categories. It should be possible to amend a large amount of voxels in real-time, but we also require fast read access as well as coherency to efficiently traverse the structure. An out-of-core approach is not necessary as we only need to render a small part of the voxel world.

Even though we are interested in facilitating solid geometry our data is still of sparse nature. As mentioned before, a typical voxel world has large amounts of empty space. Many of the data structures listed in Aleksandrov et al. [1] try to increase compactness by exploiting sparsity as well. It is critical to not compromise too much on access time, when increasing compactness, as updating and rendering the voxel world needs to happen in real-time. Nevertheless, it should be noted however that compactness and more specifically exploiting sparsity in data is important for ray tracing. The number of ray box intersections can be greatly reduced if the data structure efficiently identifies empty space.

The simplest data structure is a regular grid which is simply a m-dimensional grid stored in a one dimensional array. This data structure provides $O(1)$ time random access but requires n^m storage. Due to the sparse nature of a voxel world as mentioned before there are more compact solutions possible.

A common approach to storing large voxel grids is the Sparse Voxel Octree (SVO). Several works [44][26][30][4][42] have explored the use of SVOs for storage as well as rendering. Their approach are able to exploit sparsity moderately well. However, random access to an element in this structure results in a $O(\log n)$ complexity. Recent research [25] suggests the use of Sparse Voxel Directed Acyclic Graph (SVDAG) which is derived from SVOs as an even more compact structure by exploiting repetitive patterns in the data. Sadly the random access time is not improved by the use of SVDAGs.

In Bridson [11] the author argues that a Sparse Block Grid (SBG) performs more optimal as opposed to a SVO solution. In their case a two-level grid which consists of a coarse top level grid and a dense bottom level grid. The coarse-dense dynamic allows for some level of compactness while still providing constant time random access. The $O(1)$ time access of their approach is beneficial compared to the $O(\log n)$ time access of SVO solutions when it comes to random access. The random access specifically is important for voxel games.

The approach, called brick maps, described in Christensen and Batali [13] is similar to both SBGs as well as SVOs. While the approach in this work is used for photon mapping it should generalize to voxel worlds. It should be noted that the brick map in this work is used for lookups. The efficiency of ray tracing such structure remains to be explored.

Run-length encoding (RLE) is a technique introduced by Curless and Levoy [16] to efficiently store models from range images. Their method generalizes to any voxel data structure. It is intuitive and achieves better compactness over regular grids or SBGs. Houston et al. [22] improves on earlier work by introducing a method to improve random lookup time. The method introduces two tables and restarts encoding at each row to achieve random lookups in

$O(\log r)$ time where r is the average number of runs per row. Random write is more complicated as it potentially requires re-encoding more than just a single run.

Museth [38] introduce a hierarchical voxel data structure tailored to sparse voxel data which changes over time. The purpose of this structure is support for discretization of animated data but it also generalizes to dynamic voxel data. The name VDB stems from volumetric dynamic grid with B+trees, as both are characteristics of this method. The method features amortized constant time random access making it suitable for both dynamic data storage as well as rendering. More recent work [21][60] has improved upon this method in the form of a GPU compatible implementation. Current state of the art work includes an implementation called NanoVDB [39] which features GPU compatibility as well as fast random access times and efficient ray traversal.

A Sparse Paged Grid structure is introduced in Setaluri et al. [48]. The authors compare their method to a state of the art implementation of VDB as well as dense arrays. They note that by using a pyramid of uniform grids it is possible to avoid the use of pointers and therefore avoid incoherent memory access while still exploiting the sparse nature of data and retaining compactness. While the efficient random access is explored in depth, efficient traversal of this structure for ray tracing remains to be explored.

In an article by mikolalysenko [34] about voxel engines the author explains the concept of chunks which are a type of virtual arrays. Virtual arrays allow voxel games to use a theoretically unbounded voxel world as parts of this world are loaded an unloaded based on the player. In this sense virtual arrays can also be seen as a multi-level grid, as the chunk indices make up a higher level grid. The dynamic loading and unloading is why this approach can be labeled as out-of-core approach.

The author of this article also argues that not only random read and write access is important for a data structure used in a voxel game. The cost of iterating over the grid is underestimated. To show why this is important for voxel games the author notes the various aspects of such game and talks about how they interact with the data structure. Whether the dominating cost for such aspect is based on random access or iterating. The article notes that, considering all these aspects, iteration cost is more important than random read and write cost.

However, it is important to consider that the author assumed rendering by rasterization. Therefore they considered mesh generation as one of the aspects. Mesh generation is indeed dominated by iteration cost, but when one uses ray tracing mesh generation is not considered. Instead, one traverses the data structure directly to find the nearest intersection.

To conclude, each of the data structures aims to be very compact while also supporting close to constant access time. While random access and compactness are both important for supporting a large voxel world in real-time, efficient space traversal is of utmost importance for ray tracing. NanoVDB is currently state of the art and seems to most versatile in this respect. Delivering a rich feature set as well as good ray tracing performance. Nevertheless, when a lower level of compactness is sufficient and traversal and access times are more important a simple approach such as SBG could prove to deliver better performance. Both methods use a variant of a multi-level grid structure. In the case where the requirements are known up front a structure of static nature such as SBG with 2 hardcoded levels will likely prove to be faster than a structure of dynamic nature such as NanoVDB.

2.4 Many lights

Direct light calculations are a critical part of modern graphic pipelines. While sampling from a few simple light sources is a well understood problem, efficiently sampling from a large collection of light sources remains a tall task. Even if visibility checks in the form of shadow rays are skipped, simply iterating over the collection of lights can be quite costly.

Several methods [57][6][56][14] try to circumvent iterating over all the light sources by maintaining a acceleration structure from which light contribution for an arbitrary shading point can efficiently be gathered. Furthermore, state of the art methods [28][37][29] are able to maintain these acceleration structures for dynamic scenes in real-time.

The method by Lin and Yuksel [28] is able to handle dynamic scenes containing thousands of emissive surfaces by constructing a grid of virtual point lights (VPLs). The construction time of this grid depends on the number of VPLs. Any change in the VPL data means the grid must be reconstructed. This can lead to a bottleneck in performance as construction of a grid can dominate rendering time. Considering the many changes in light sources over the runtime of a sandbox voxel game this is something we cannot afford.

Moreau et al. [37] describes a method to gather light contribution by traversing a two-level Bounding Volume Hierarchy (BVH) over the lights in the scene in a stochastic manner. While the construction of such BVH takes considerable time, the authors show that refitting can be used for the majority of changes in their dynamic scenes. Nevertheless, the construction time of the BVH could be problematical in voxel setting where lights are not moved around the scene but instead removed and added to the scene. Furthermore, the dynamic loading and unloading of parts of the voxel world would have to be factored in when deciding on how to construct the levels of the BVH. To avoid a complete rebuild when part of the world is loaded or unloaded.

The method by Lin and Yuksel [29] improves on previous methods using light cuts [57][62] by introducing a GPU friendly acceleration structure as well as sharing light cuts in a k x k block of pixels. By using a perfect binary light tree their contribution has considerably faster tree construction times as well as sample time compared to previous works. Since the tree can be efficiently reconstructed every frame this method allows for a scene with many dynamic lights to be rendered in real-time.

In Minecraft the problem of many lights is solved in an entirely different manner[3]. The contribution of nearby light sources is stored per voxel. This contribution affects not only the rendered brightness of a voxel but also various game mechanics such as spawning of monsters. The contribution is calculated using the taxicab distance in combination with the flood fill algorithm whenever the world is updated. Since the game uses light in the same way for various game mechanics this approach is efficient. However, it is possible that this method does not scale well for higher light intensities. As this would result in more voxels to be updated on change of such light. Furthermore, this method also results in visible artifacts as light contribution diminishes linearly over distance.

2.5 Resampled importance sampling

Talbot [53] identified that by using Resampled Importance Sampling (RIS) it is possible to achieve lower variance when performing Monte Carlo integration. The idea behind RIS is to generate a set of samples which mimics the function being integrated. This is achieved by selecting a weighted subset of samples from the initial set of samples.

2.5.1 ReSTIR. Reservoir Spatiotemporal Importance Resampling (ReSTIR), a prominent recent work by Bitterli et al. [8], focuses on solving the many lights problem. ReSTIR is a spatiotemporal method which uses RIS to sample the set of lights in the scene for a shading point. ReSTIR build upon the RIS method proposed by Talbot [53]. The key benefit of this method is that, by using both spatial and temporal sampling reuse in combination with resampling, it creates a sampling that will lead to a light contribution often close to the expected value. Thus resulting in a relatively less noisy image for a low sample per pixel (spp) count.

Spatial and temporal reuse has been used before by denoising methods however the key difference is that these methods reuse color or illumination values whereas ReSTIR reuses the probabilities of picking a certain light source.

ReSTIR only considers direct lighting because the reservoirs are stored in screen-space. The coherency of screen-space pixel information keeps spatiotemporal reuse of reservoirs intuitive. Still, recent methods [9][41] have been proposed which are able to use reservoir based spatiotemporal resampling for global illumination.

The authors of ReSTIR note that improvements in variance may be modest in situations where spatial or temporal reuse is hampered. High geometric complexity can introduce bias because of spatial reuse of samples. Moving lights or geometry can hamper temporal reuse by introducing bias due to incorrect reuse of samples. In both cases an unbiased solution would have to reject incorrect samples which introduces noise. A biased solution which does not reject these incorrect samples creates a darker result as the weight for the sampling is an incorrect representation of the coupled light source for the given shading point.

The ReSTIR method uses weighted reservoir sampling (WRS) [12][19] in combination with RIS as a foundation. Inspired by Efraimidis [18] the authors of ReSTIR created an implementation which uses simple data structures to process random candidate samples in a streaming fashion. The streaming fashion of this implementation is very suitable for GPUs as it can be executed in parallel and needs a small, constant memory amount. The method is specifically aimed at calculating direct light contribution and generalizes well to be used in a ray tracing or rasterization pipeline.

2.5.2 Rearchitecting Spatiotemporal Resampling. In Wyman and Panteleev [61] the authors take a closer look at ReSTIR and come up with a rearchitectured method.

First, the authors look into the cause of bias and came up with an improved heuristic which aids in rejection of bad samples. Bitterli et al. use a simple heuristic to determine if two pixels are similar enough to allow for cross reuse of reservoirs. Wyman and Panteleev note that instead of using a heuristic for a biased algorithm or multiple importance sampling (MIS) for an unbiased algorithm, it is possible to combine the heuristic with MIS by using the heuristic as a form of MIS weighting.

Next, the authors noted that while ReSTIR has constant time computational complexity with relation to the number of lights, non constant performance was often observed. This was attributed to cache thrashing due to the number of random lights picked out of the large collection of possible lights. By creating a subset of lights through presampling, which is used for per-pixel RIS, this memory incoherency issue is resolved. Furthermore, by creating subsets of lights in a stratified manner, the issue of incoherency is solved and better overall sampling is provided.

Bitterli et al. proposed using 4 reservoirs per pixel for their biased algorithm but Wyman and Panteleev found that this is wasteful as often the same light is sampled multiple times due to the weighted sampling. The authors show that one reservoir per pixel is a better choice. This brings the number of shadow rays to be traced per pixel from 5 down to 2. While this does mean reduced quality it is more in line with ray budgets for games.

The spatial reuse in the ReSTIR methods requires a per frame global barrier to sample neighbouring pixels of the current frame. Wyman and Panteleev note that by sampling neighbours from the frame before allows for this barrier to be removed which in practice allows for better occupation of the hardware. The authors do however note that this is undesirable in fast moving scenes as the convergence is delayed by 1 frame.

The authors note that 1 spatial sample reuse is often sufficient at higher framerates as it is masked by temporal reuse. Lower framerates or higher target quality may want 2 to 3 spatial sample reuses however.

The rearchitectured method also decouples the shading stage from reuse. This offers more flexibility in the number of visibility queries and for which of the samples a visibility query must be performed. For example by reusing the visibility for reused temporal samples.

Lastly, the authors note that ReSTIR and rearchitectured ReSTIR do not perform well in scenarios with very complex lighting. In a scene with many small lights resampling often ends up undersampling these lights. As pixels rarely find a relevant light source close enought to have any form of impact.

Our interest lies in the effectiveness of the ReSTIR method when applied to ray tracing voxel scenes. As mentioned before, a problem with this method arises when many lights have potentially equal contribution. Voxel scenes cause this problem inherently as voxels are classified resulting in many lights having the exact same emissive properties. On top of that each voxel counts as a separate light, and therefore all have the same area. These two properties combined lead to many of the lights having equal potential contribution leading to a worst case scenario for ReSTIR. We aim to take a closer look at the effectiveness of the method in such scenario.

Concurrent to our research Lin* et al. [27] generalized RIS. They reformulate ReSTIR's spatiotemporal reuse to remain unbiased for long paths. Their Generalized RIS (GRIS) algorithm is able to achieve interactive path tracing with little noise.

2.5.3 ReGIR. Reservoir Grid-based Importance Resampling (ReGIR) by Boksansky et al. [10] is a method inspired by ReSTIR. This method can be used for primary hits as well as secondary hits, although its intended use is for secondary hits.

ReGIR places a uniform grid in the scene. In each cell multiple reservoirs with light candidates are stored. Each reservoir holds samples selected from a larger set of lights as well as meta data about how it was constructed. Including: The number of candidates evaluated, their total weight, and the target probability density function (PDF).

For each cell in the grid a collection of light candidates is created by uniformly sampling from the pool of all lights. Using this collection multiple reservoirs are created by resampling from the collection with weights based on the potential contribution of each light to the cell center position.

The grid of reservoirs is constructed this way every frame. However, the reservoirs in the grids from previous frames can be merged with the reservoirs in the grid from the most recent frame. This form of temporal reuse is key in continuously improving the grid.

When sampling the grid in order to shade a point the reservoirs in the corresponding grid cell are merged for a final single light sample.

As shown in the paper the intended use of ReGIR is for shading of secondary hits as opposed to shading of primary hits, where it is outperformed by ReSTIR. This is due to the use of a world space grid which results in a larger granularity as opposed to screen space solutions such as ReSTIR.

2.6 Global illumination techniques

Global illumination improves the realism of a scene but is computationally expensive and dependent on the complexity of the scene. Previous methods [15] [55] [43] have offered solutions for real-time global illumination using voxels. These methods use voxels as simple representation of the polygon scene to exploit the fast ray-scene intersection made possible by using a voxel grid.

The method by Crassin et al. [15] uses cone tracing on a data structure to gather indirect illumination in the scene. This data structure is a sparse voxel octree which stores the indirect illumination in the scene. When using voxels as geometry some of the issues mentioned in the paper, such as light leaking, could be reduced or eliminated. This is because the voxel octree would be a closer approximation to the scene geometry as opposed to when using a polygon scene. The complexity of this algorithm as well as the memory needed for the data structure are limiting factors. Specifically for scenes with many dynamic lights updating the data structure would take considerable time. Lastly, this method only supports single bounce illumination.

While the method by Thiedemann et al. [55] focuses on voxelization and an improved ray-voxel intersection test, the authors also describe several techniques for estimating global illumination. The first approach uses shadow maps for fast near-field illumination. Real-time rendering using 1 bounce is achieved. However, the effectiveness of this method when it comes to scenes with many dynamic lights remains to be explored, since shadow maps have to be rendered for each light source in the scene. The next approach involves path tracing global illumination rays through the voxel grid. This approach is simple and intuitive but quite costly. The number of rays is bound by the ray-grid intersection cost whereas a noise free result requires many such rays. The last approach uses VPLs to gather radiance at a shading point. Much like the path tracing approach this approach is computationally expensive because of the visibility tests.

Reflective Shadow Mapping (RSM) [17] is based on traditional shadow mapping and creates a set of VPLs for a set of direct light sources. Recent work [52] is able to use this technique in real-time in combination with ray marching through a sparse voxel octree for visibility testing. The work by Zhang and Oh [63] employ a similar albeit slightly different approach from VPLs. It uses ray marching and marches a few rays per shading point in random direction through a voxelized scene. When an intersection is found a visibility test is performed to see if this point is occluded otherwise the contribution from the light source is propagated.

Creating shadow maps using the rasterization pipeline relies on the assumption that the direct light sources are point light sources. Since a shadow map is created for every light source these methods do not scale well for scenes with many light sources.

The method by Jendersie et al. [24] uses surfels to gather local illumination in the scene and propagate using a hierarchy and a set of precomputed light paths. While this method shows promising performance and is able to handle dynamic lighting as well as some dynamic objects, it heavily relies on pre-computation. This precomputed data loses value quickly in scenes with many dynamic objects at which point this method becomes expensive to maintain.

Recent work by Boissé [9] explores the use of reservoir based resampling methods for light sampling in world space. The method caches the reservoirs for light path vertices in a hash grid. Doing so allows for stochastic reuse of neighbouring across space and time. The implementation of this method is well suited for GPU hardware. The method as proposed in this paper works for single-bounce global illumination. Furthermore, scenes with many emissive surfaces are efficiently handled by the reservoir based resampling.

Another method by Ouyang et al. [41] builds upon ReSTIR to handle paths beyond single bounce. This method places initial sampling of lights in the space of the local sphere of directions around the shading points, as opposed to the ReSTIR method which places the sampling in a global light space. The RIS weights used by this method for spatiotemporal resampling are determined by the radiance which is scattered back by the ray corresponding to the random direction for each shading point. This allows the method to make use of simple screen-space buffers, in contrast to methods such as ReGIR [10] which requires a complex world-space data structure.

Lastly, a method called Ambient Occlusion (AO) is present in many games. This method [64][23] simplifies the rendering equation by assuming the scene is lit uniformly. The method imitates global illumination without simulating the incident and reflected light by calculating the attenuation of ambient light due to occlusion of nearby geometry. Minecraft also applies this method in their renderer as smooth lighting [3].

2.7 Discussion

To efficiently render a dynamic voxel world with many lights in real-time we require:

- A data structure which can be efficiently intersected for a ray as well as efficiently updated to accommodate changes.
- A low noise method for calculating light contribution of many light sources in the scene.

The voxel renderer we will employ uses a multi-level grid similar to SBG with a static number of layers. As concluded at the end of Section 2.3 the state of the art method NanoVDB has many advantages over a simpler structure like SBG. However, multi-level grid structures with a set number of layers have the upper hand when it comes to performance due to the simpler traverse logic. This aspect is important and takes precedence over the features NanoVDB offers, as it is one of the bottlenecks to rendering. While both methods use a multi-level grid structure at their core, a solution with a static number of layers is more intuitive. This is also an important aspect for beginning programmers.

The ReSTIR method and improvements described by Wyman and Panteleev are currently state of the art and best suited to rendering of scenes with many lights in real-time. We are especially interested in the effectiveness of this method when used with problematical scenes.

While this method allows for dynamic scenes with many lights the temporal aspect could prove to be ineffective in various scenarios. Temporal sampling reuse relies on consistent properties of the lights across frames. While properties of emissive surfaces in voxel worlds are often static and therefore consistent across frames the set of light sources is not. It is possible for lights to be removed or added at any point.

Furthermore, the ReSTIR method has problems with scenes with many small lights with equal size and emissive properties, as mentioned in Wyman and Panteleev [61]. This is relevant to voxel worlds as described in Section 2.5.2.

Lastly, we are interested in possible advantages of using a multi-level grid structure in combination with ReSTIR. Using such structure allows for efficiently identifying large blocks of homogeneous space such. This includes large blocks of empty space or large blocks of emissive material. Altering the ReSTIR method to handle such cases specifically could improve performance or visual quality. The grid structure also allows for efficiently finding close proximity lights given a point in the scene, which in term could be used in the weighting function used for resampling.

3 METHODOLOGY

As described in Section 2.5 to solve the many lights problem for direct illumination there is the ReSTIR method and the rearchitectured spatiotemporal resampling method. Both methods include a biased as well as an unbiased variant. In this work we will look at the biased ReSTIR method.

We choose this method instead of the rearchitectured spatiotemporal method as it is easier to understand. The rearchitectured spatiotemporal method builds upon ReSTIR and improves both visual as well as computational performance. Therefore we implement ReSTIR first to have a base line as well as getting a better understanding of the method. The difference in image quality between the biased and unbiased variant is small but visible. The biased variant of the method struggles with a darkening bias. The unbiased variant, on the other hand, uses a heuristic as well as MIS to weight neighbours differently and trades performance for an unbiased result. This trade in performance is unappealing in games as the time budget for rendering is small.

3.1 Approach

To determine the effectiveness of ReSTIR when rendering a voxel scene we evaluate a voxel ReSTIR implementation by comparing it to a ground truth image. We implement ReSTIR in a voxel renderer called WrldTmpl8 [7]. The voxel ReSTIR implementation will be validated to ensure we create useful data for our experiment.

In order to validate our implementation we require an image rendered using an existing ReSTIR implementation and compare this to an image rendered using our implementation. We will use GfxExp [50] as existing ReSTIR implementation. We chose to use this implementation as the source code is readily available. The implementation also features various on screen information about the rendering as well as easy parameter tweaking to match our own implementation.

Lastly, we evaluate the performance of our ReSTIR implementation by comparing an image rendered by it to a ground truth image rendered using path tracing. This ground truth image is path traced using the WrldTmpl8 voxel renderer.

In short we:

- Implement the ReSTIR method in the voxel renderer WrldTmpl8.
- Validate our implementation by comparing a result image to one produced by the ReSTIR implementation GfxExp and the ground truth image.
- Evaluate our implementation by comparing a result image to a path traced ground truth image.

3.2 Setup

Once we have the three methods from the previous section we render a number of scenes and compare the resulting images. As mentioned before we want to determine the difference in image quality between our implementation and GfxExp as well as the difference in image quality between our implementation and a ground truth image. This means we have three similar setups each using another method of rendering.

To do so we handpick various scenes we think will provide distinct results.

- A mountain landscape with a large number of small, equal and dim light sources scattered around. This should be a worst case scenario for any ReSTIR implementation as mentioned in Section 2.5.2.

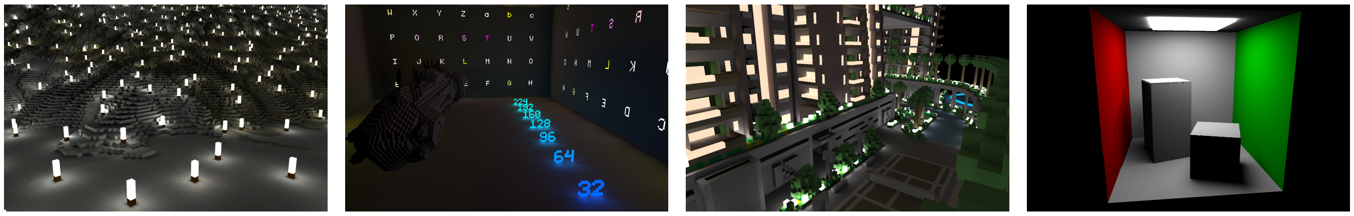


Figure 2: The 4 scenes we used in our setup (left to right): Mountain scene, Emissive letters scene, Flying apartments scene, Cornell box scene.

- A scene with emissive geometry in the shape of letters. To test the effectiveness of the implementation on complex emissive geometry.
- A scene with a very large number of lights of varying potential contribution.
- A scene similar to the Cornell box scene.

We show our scenes in Figure 2.

To compare the image quality we render the scenes using the three renderers and export the images. Each image is rendered using the same parameters including ReSTIR parameters, resolution and gamma correction.

In order to produce test setups for both GfxExp and WrldTmpl8 that can be compared we create the scene as a voxel world in WrldTmpl8 and export this scene as polygon scene using a greedy meshing technique [35]. Furthermore, we export the camera parameters used to capture the scene in WrldTmpl8 and use these parameters to capture the scene in the exact same manner in GfxExp.

ReSTIR parameters can easily be configured in both setups. We chose the parameters as suggested by Bitterli et al. which means spatiotemporal resampling with $M = 32$ (candidates). Spatial sampling is done using 5 taps in a 30 pixel disk around the pixel. Temporal sampling is done using at most 20 times the importance of the candidate sampling.

We chose a resolution of 1600 by 1024. While this is slightly lower than the conventional 1920 by 1080 resolution that is most commonly used, it did allow us to easily test our implementation without having to use it fullscreen. We disabled any form of gamma correction in GfxExp and WrldTmpl8.

Note that a small difference in color can be present between images rendered in WrldTmpl8 and GfxExp. This is due to the way materials are stored in each framework. In WrldTmpl8 each material is simply assumed to be diffuse and has 4 bits per color channel for a total of 12 bits per color. In GfxExp the materials are also diffuse but have 32 bits floating point accuracy per color channel for a total of 96 bits per color. Since our scenes are made in WrldTmpl8 and exported from there the difference in color is introduced when converting the 12 bit fixed point color to 96 bit floating point color. This difference should be negligible, but if we were to convert a polygon scene into a voxel world instead the difference in color precision could prove to be troublesome.

We also collect frame time information when rendering using our ReSTIR implementation as we are interested not only in image quality but also performance. The experiment is executed on a system with an AMD R5 5600x CPU and NVIDIA GTX 1080 GPU.

Lastly, We also look at the images qualitatively to see if any specific artifacts or noise is present.

3.3 Method of analysis

We compare each rendered image to the corresponding ground truth image by calculating the Mean Square Error (MSE) and Mean Absolute Error (MAE). Equations (1) and (2) show how we calculate MSE and Equations (3) and (4) show how we calculate MAE. In these equations P is an array of pixel values p which are vectors containing the red, green and blue values. Both MSE and MAE are commonly used to quantify the per pixel difference in color between two images. Bitterli et al. used the Relative Mean Absolute Error(RMAE) metric. They explained their preference for the metric over MSE because it is less sensitive to outliers. The more recent work by Wyman and Panteleev uses MSE but did not explain their choice further. We decided to use MAE because it is less sensitive to outliers and also include MSE to be comparable to other works.

$$PSE(p_1, p_2) = (p_{1_r} - p_{2_r})^2 + (p_{1_g} - p_{2_g})^2 + (p_{1_b} - p_{2_b})^2 \tag{1}$$

$$MSE(P_1, P_2) = \frac{1}{|P_1| * 3} * \sum_{p_1 \in P_1, p_2 \in P_2} PSE(p_1, p_2) \tag{2}$$

$$PAE(p_1, p_2) = |p_{1_r} - p_{2_r}| + |p_{1_g} - p_{2_g}| + |p_{1_b} - p_{2_b}| \tag{3}$$

$$MAE(P_1, P_2) = \frac{1}{|P_1| * 3} * \sum_{p_1 \in P_1, p_2 \in P_2} PAE(p_1, p_2) \tag{4}$$

While the metrics above are excellent for comparing two images quantitatively, it only considers pixel color difference between the two images. The MAE or MSE values do not indicate if distracting artifacts are present.

The OpenCL kernel execution time values are an indication of performance and allows us to determine if our implementation runs in real time. The frame time is sum of the kernel execution times for every kernel in our pipeline. Real time in games is not well defined but we hope to achieve a stable frame time below 17ms which roughly equals 60 frames per second.

3.4 Method of evaluation

To validate if our voxel ReSTIR implementation is correct we calculate the MSE and MAE values between each image and the ground truth image. We repeat this process for the GfxExp implementation. We calculate the MSE and MAE values between each image rendered with GfxExp and the ground truth image. We deem our implementation correct if the MAE values for our implementation are similar or lower than the MAE values for the GfxExp implementation.

4 IMPLEMENTATION

In this section we explain our implementation of Voxel ReSTIR in the WrldTmpl8 framework as well as the design choices we made. In Section 4.1 we describe necessary details about the WrldTmpl8 framework. In Section 4.2 we explain the grouping of emitters into brick emitters which is a feature unique to voxel worlds. In Section 4.4 we describe our implementation of the initial candidate sampling and temporal resampling. In Section 4.6 we describe our implementation of the spatial resampling. Lastly, in Sections 4.7 to 4.9 we describe our design choices for the sample count per pixel, the reservoir data structure and the weighting of temporal samples respectively.

4.1 WrldTmpl8

We implemented Voxel ReSTIR in the WrldTmpl8 framework. This framework supplies us with an easy to manipulate grid of 1024^3 voxels which can be accessed on the CPU and GPU. By default this grid can be read from and written to on the CPU and will automatically be synced to the GPU to be read from there. This means we have easy low level control over the voxel world. The framework has a simple loop which calls an update function as well as the OpenCL kernels that form the rendering pipeline every frame. The framework provides optimised accelerated ray-grid query methods which enables fast prototyping of ray tracing workloads. And finally, the availability of motion vectors assisted us in the implementation of temporal reprojection for the Voxel ReSTIR implementation.

As mentioned before each voxel stores a 12 bit color value which includes 4 bits per color channel. Together with this 12 bit color value we store 4 bits for emission strength. In our implementation this means we can only use integer emission values from 1 to 15 however this could easily be changed to assign a different range of emission strength values with 4 bit precision.

4.2 Brick lights

Reducing the number of light sources is a simple way of reducing variance as less light source candidates have to be considered for a shading point. Reducing the number of light sources with loss of information is however hardly possible in polygon scenes with complex emitting geometry. In voxel worlds this is another story. We can simply group a block of emitting voxels into a single light source as long as the block is homogeneous.

The WrldTmpl8 framework works with a multi-level grid. A top level grid of 128^3 contains bricks which contain a voxel color value if the brick is a homogeneous volume of voxels or an index into the bottom level grid otherwise. To keep the implementation simple we decided to use this feature with the aforementioned idea of grouping emitting voxels to implement brick lights. A brick light is a single light source which represents 8^3 emitting voxels. In our implementation the brick light needs to be aligned with a brick in the top level grid however this is only the case to quickly recognize bricks of emitting voxels and does not have to be a hard requirement in future implementations.

4.3 Direct light sampling

ReSTIR by Bitterli et al. only considers direct lighting and so does our Voxel ReSTIR implementation. We implemented two ways to sample a voxel emitter: as point light or as area light.

When the emitters are considered as point lights a candidate light will simply always be sampled using the center point of the voxel or brick. Naturally this results in hard shadows but it also results in lower variance as well as higher performance. The latter are discussed in Section 6.4.

When sampling the emitters as area light we generate a random point on the surface of the emitter and sample the emitter given that point. Our emitters are either single voxels or brick lights, both of which are cubes. We consider each side of the 6 sides of the emitter as area lights and importance sample these sides using the solid angle given the shading point and a random point on the side. Much like other solid angle sample techniques this technique also suffers when the shading point is near the emitter surface as the solid angle approaches 0. This issue is commonly solved using MIS at the cost of an extra visibility check. By sampling the emitters as area lights we are able to render soft shadows and converge to the ground truth. The trade-off is lower performance and higher variance.

4.4 Candidate generation

Our Voxel ReSTIR rendering pipeline has a stage for initial candidate generation and temporal resampling. In this stage we generate 32 candidates and importance sample these candidates using weighted reservoir sampling. We test if the chosen candidate is occluded. If it is occluded we reject the reservoir and set the weight of the reservoir to 0.

Next we use temporal reprojection to retrieve the reservoir for the current pixel from last frame. The reprojection is done using motion vectors and rejected when the reprojected pixel depth from last frame is not within 10% of the depth of the current pixel or if the normal is not equal to the normal of the current pixel. Since we only work with 6 normals in a voxel world we decided to only accept normals in this step that are equal. If the reprojection is accepted we combine the reservoir from last frame with the reservoir of the current frame.

We decided to do initial sampling using 32 candidates as proposed in Bitterli et al.. These candidates are uniformly generated from the pool of lights. We did not implement a method to generate the candidates using importance sampling however implementing such method using for example alias tables could prove to be beneficial as it could increase the average quality of the pool of candidates.

4.5 Target PDF

For our target PDF $\hat{p}$ we simply use the unshadowed path contribution, as shown in Equation (5). In this equation x is the selected candidate light to sample from, ρ is the BSDF term, L_e is the emitted radiance, and G is the geometry term which include the inverse squared distance and cosine terms. This PDF is used when filling the reservoirs during initial candidate sampling. The unshadowed path contribution is a 3 component vector as it contains red, green, blue values so we take the magnitude of this vector as value for our PDF.

$$\hat{p}(x) = \rho(x)L_e(x)G(x) \tag{5}$$

Algorithm 1 Point on voxel using importance sampling

Input: Shading point p, Center of voxel c
Result: Point on voxel p_v, weight w

```
1: $NumberOfSides \leftarrow 6$
2: $SolidAngles \leftarrow$ new Array[$NumberOfSides$]
3: **foreach** side $s \in$ Voxel:
4:     $p_{surface} \leftarrow$ pointOnSide($s$)
5:     $SolidAngles[s] \leftarrow$ solidAngle($p$, $p_{surface}$)
6: $pdf \leftarrow$ createPdfFromValues($SolidAngles$)
7: weight $w$, side $s \leftarrow$ importanceSample($pdf$)
8: $p_v \leftarrow$ randomPointOnSide($s$)
9: **return** $p_v$, $w$
```

In the case of area light sampling we also importance sample the sides of the emitter. The PDF corresponding to this importance sampling is included in our target PDF in the form of w in Algorithm 1. This results in our target PDF being a compound PDF, as shown in Equation (6) where $w(x)$ is the weight resulting from picking a point on x using Algorithm 1.

$$\hat{p}(x) = \rho(x)L_e(x)G(x)w(x) \tag{6}$$

In the case of point light sampling we use Equation (6) but with $p_v = c$ which is the center of the voxel, and $w(x) = 1$.

4.6 Neighbour selection

Our Voxel ReSTIR rendering pipeline has a stage for spatial resampling which is performed after the initial and temporal resampling stage. In this stage we resample the reservoir for the current pixel based on the reservoirs of neighbouring pixels. We sample 5 random pixels in a 30 pixel disk around the current pixel. If the neighbouring pixel has a depth within 10% of current pixel depth and the same surface orientation we combine the reservoir of that pixel with the reservoir of the current pixel. As discussed before we only have 6 surface orientations in a voxel world so we only accept surface orientations that are equal.

It is also possible to use world space distance between a neighbouring pixel and the current pixel to determine if we should reject it. The current heuristic which accepts pixels within 10% of the current pixel depth is not perfect and it is possible to spot darkening bias for pixels at greater depth.

4.7 Evaluated sample count

Our implementation traces 3 rays per pixel. The first ray is used to trace the albedo image. The second ray is a shadow ray traced after the initial candidate sampling to determine if the chosen light source is occluded. The third and final ray is traced when shading the pixels, to determine if the chosen light source is occluded. Bitterli et al. propose using 5 reservoirs per pixel for shading and therefore trace more rays per pixel. Wyman and Panteleev shows however that this is mostly wasteful so we decided to use a single reservoir per pixel. The implementation does not support multiple samples per reservoir.

4.8 Reservoir storage

We store a single reservoir per pixel which is filled during the initial candidate generation stage and resampled using spatiotemporal resampling.

The reservoir structure we employ holds the sum of weights of all samples seen, the number of samples seen, the index of the chosen sample into the pool of lights, the adjusted weight for the chosen sample, the position on the surface of the chosen emitter, $\frac{1}{pdf}$ where pdf is the PDF value from importance sampling the sides of the emitter, the normal at this position on the surface of the chosen emitter and the weight of the chosen emitter. This is different from the structure described by Bitterli et al. which only stores the sum of weights, the number of samples , the index of the chosen sample and the adjusted weight. We included more information to keep track of the information around the chosen emitter as the structure passes through multiple kernels.

4.9 Temporal weighting

As described in Bitterli et al. the temporal reuse leads to unbounded growth of reservoirs. Similar to Bitterli et al. we combat this effect by reweighting the reservoir when it exceeds 20× the number of samples seen compared to the current reservoir. This also prevents the reservoir from going stale as a large sum of weights due to a large number of samples seen would cause the probability to keep reusing previous samples to be disproportionately high.

5 RESULTS

In this section we show the results obtained using our implementation and the GfxExp application. Renderings and timings were obtained using an AMD R5 5600x CPU, NVIDIA GTX 1080 GPU and 16GB of RAM. We used OpenCL profiling functions to measure kernel execution times.

As discussed in Section 3 we use four scenes. We show the results of using our Voxel ReSTIR implementation on various scenes with 32 candidates ($M = 32$) and compare our implementation to GfxExp, the reference ReSTIR implementation, as well as a ground truth image, which is calculated using a single bounce so only direct lighting is considered. All scenes rendered with the two ReSTIR algorithms use both temporal and spatial resampling. For spatial resampling we use 5 taps in a 30 pixel radius as described in Section 3.2. The images are rendered with a static camera after more than 20 frames. This provides an artificial advantage as we are temporally super-sampling a single view, however this advantage is shared across every test setup. We show the kernel execution times in milliseconds in Tables 1 and 2. In Table 3 we show the measured error values for each setup.

We use the following four scenes for our tests, which emphasize various characteristics of the used algorithms.

The mountain scene has a large number of small dim lights and simple geometry yet with many occlusions. Due to the nature of the

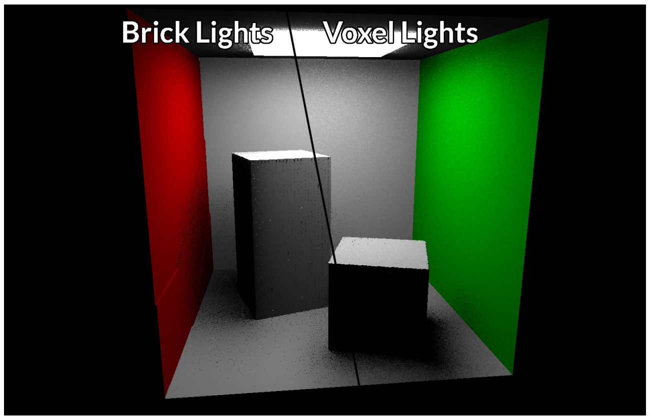


Figure 3: Left: 256 bricks emitters. Right: 131072 voxels emitters

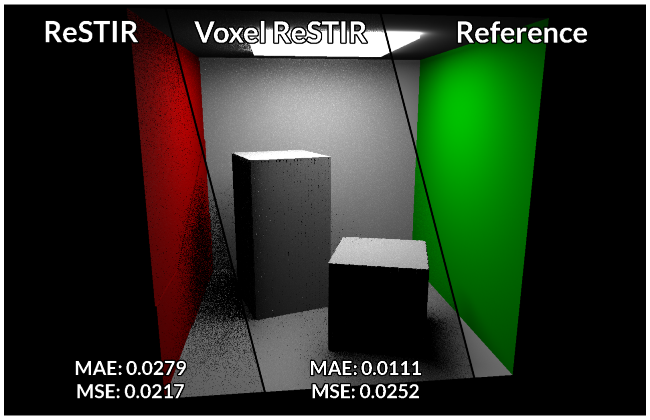


Figure 4: Cornell box scene with 256 emitting bricks

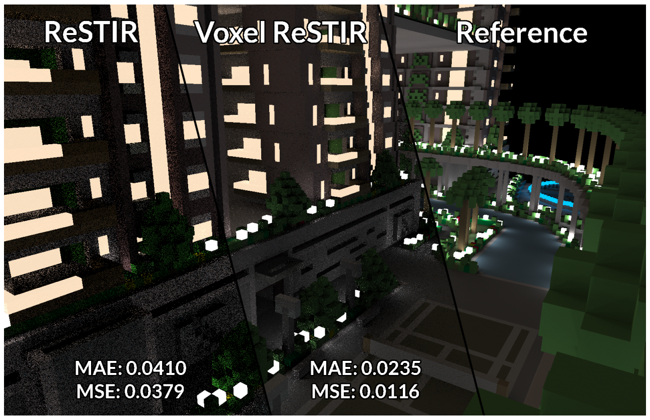


Figure 5: Flying apartments scene 1

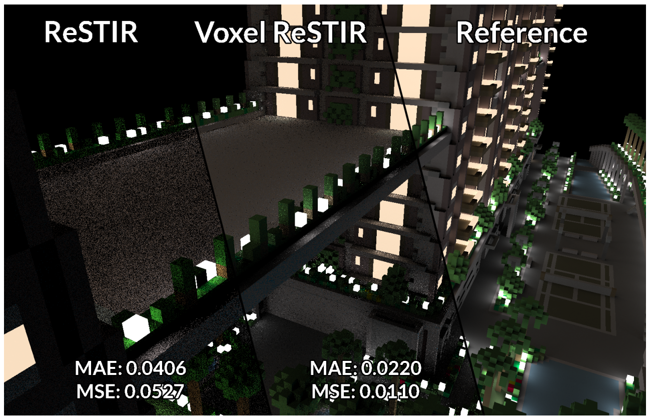


Figure 6: Flying apartments scene 2

geometry Voxel ReSTIR cannot effectively apply spatial resampling. In general this should result in noise and a darker image as for many pixels no relevant candidate can be found to sample from.

The scene with emissive text has complex emitting geometry. The emissive letters and numbers are formed by light emitting voxels. The letters embedded into the walls are enclosed on all sides but one. The emissive numbers on the floor indicate the depth of the scene in voxels.

The flying apartments scene consists of complex geometry mixed with many emissive voxels and most closely represents a scene in a voxel game. This scene has a mix of emissive voxels of different colors. The complex geometry creates many occlusions and restricts the effectiveness of reuse across pixels. Even so the scene has flat surfaces where spatial reuse is effective and the algorithm has the ability to generate high quality samples.

The Cornell box scene has a single large light source in the top which consists of many emitting voxels which can be grouped into bricks to reduce the number of emitters. This scene is a standard scene in computer graphics research. It is most ideal for ReSTIR compared to the other scenes, as the many voxel emitters are triangulated into only 20 emitters using greedy meshing. This means that ReSTIR samples from a pool of lights much smaller than the pool of lights used by Voxel ReSTIR. This is the only scene where we could easily group existing emitting geometry into voxel bricks.

In Figure 3 we demonstrate the difference between using a large number of individual emitting voxels and grouping such voxels as emitting bricks. In both cases the scene has the exact same geometry and should converge to the same result. However, in real-time the case of brick lights should be favored as the number of lights is much smaller. A smaller number of lights allows for better cache utilization as well as a higher quality candidate sampling, as the number of candidates is closer to the number of lights. The difference is mostly visible in occluded regions as well as on surfaces near the light sources. Both images have been rendered using $M = 32$ and area light sampling. We show the effect of grouping emitting voxels into bricks on execution times in Tables 1 and 2.

In Figure 4 we show the same scene but here we demonstrate the difference between ReSTIR and Voxel ReSTIR. On the left we demonstrate the result using ReSTIR, in the middle we show the result using our Voxel ReSTIR implementation, and on the right we show the reference image. The difference in variance can be seen best in occluded regions such as the area where the sides of the

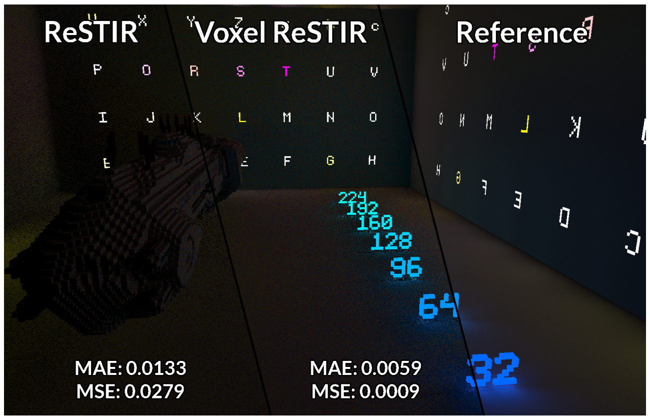


Figure 7: Emissive letters scene 1

boxes meet the floor. Areas where geometry changes such as the corners between the ceiling and the walls also show a darkening effect.

In Figures 5 and 6 we demonstrate the difference between ReSTIR and Voxel ReSTIR using the flying apartments scene. As noted before the spots with most noise are in places where geometry is inconsistent such as corners. Another point of interest is the highlights near voxel lights. This can be seen best for the bright white voxel lights in the lower half of Figure 5, as well as the bright white voxel lights on the edge of the bridge in Figure 6.

Figures 7 and 8 show the difference between ReSTIR and Voxel ReSTIR using the emissive letters scene. This scene contains complex emissive geometry and complex geometry in the middle of the scene. The emissive numbers on the floor denote the depth and are useful to compare the effectiveness of the method at greater depth. Note the difference in noise on the floor near the number in the back compared to the noise on the floor near the number in the front. The numbers show a gradual change in color but otherwise have the same emissive strength. The purpose of the complex geometry in the middle is to show the level of noise when there are generally little emitters, each placed at a distance. The comparison between the three methods is best made by looking at the level of noise on the floor as it is visible for each method in the figures.

In Figure 9 we demonstrate the difference between ReSTIR and Voxel ReSTIR using the mountain scene scene. This scene works especially well to compare the three methods as the geometry visible in each section is roughly equivalent. Areas of note are highlights on the surfaces near the lights and the gradual change in brightness of the image for each section as the depth increases. The floor close to the camera also serves as a clear indicator in the different levels of noise each method can achieve.

We show the difference in kernel execution times between the two methods in Table 1.

Figures 10 to 13 show the impact of varying numbers of candidates. For an overview of impact on performance of this setting, see Table 2.

In Figure 10 we demonstrate the Cornell box scene when using various number of initial candidates. This scene, in contrast to

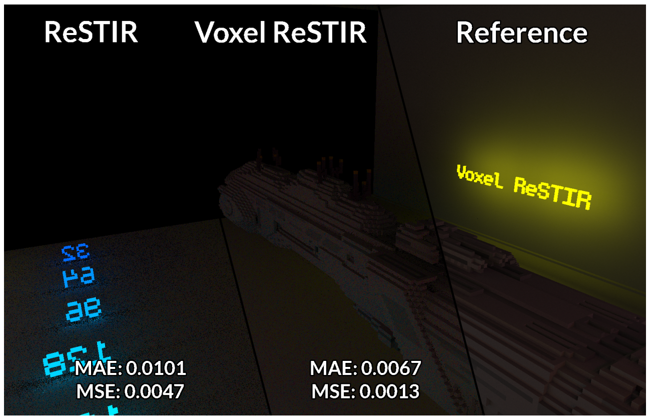


Figure 8: Emissive letters scene 2

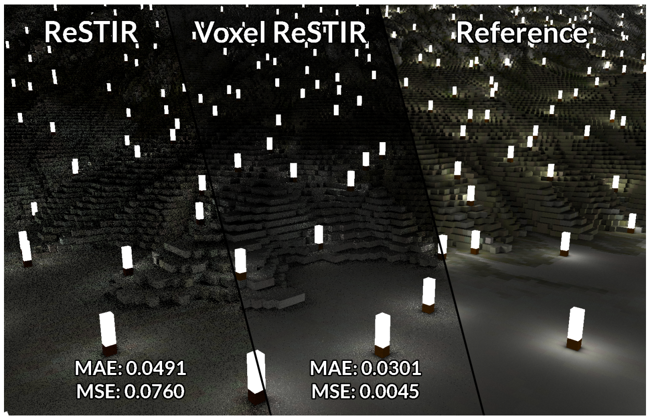


Figure 9: Mountain scene 1

following scenes, has a relative small number of lights as well as simple geometry. The impact of using a small M should be insignificant as the initial candidate sampling as well reuse provide a high quality sample.

Figure 11 shows the flying apartments scene when using various number of initial candidates. When looking closely at this Figure one can see that, as M gets smaller, there are many fireflies scattered across the image even for surface points that are not in near vicinity of a light. This scene has a mix of complex geometry and large flat surfaces as well as a large number of light sources. The impact of using a smaller M is best visible near lights that are in close proximity of complex geometry or near lights that are far from the camera. This is where the quality of the candidate sampling is most prominently visible.

In Figure 12 we demonstrate the emissive letters scene when using various number of initial candidates. This scene, in contrast to the mountain scene, has large flat surfaces and a significant smaller number of lights. Specifically largely open regions such as the floor surface demonstrate the impact, or lack thereof, of using a smaller number of M.

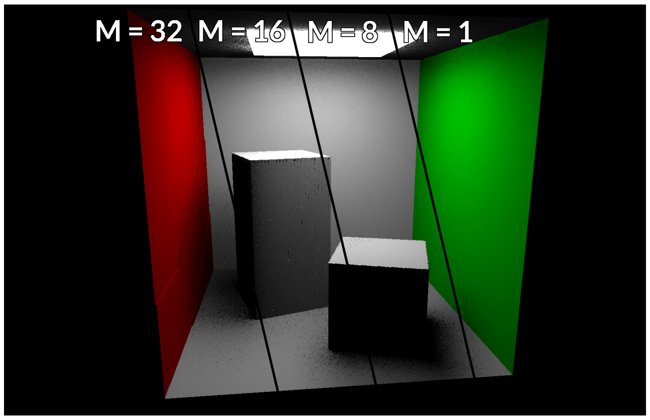


Figure 10: Cornell box scene with 256 emitting bricks

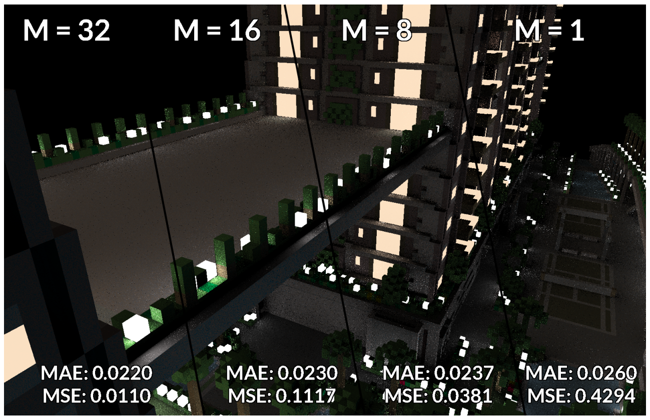


Figure 11: Flying apartments scene 2

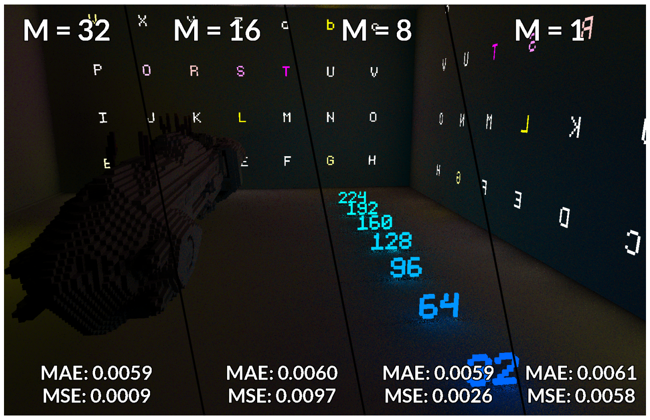


Figure 12: Emissive letters scene 1

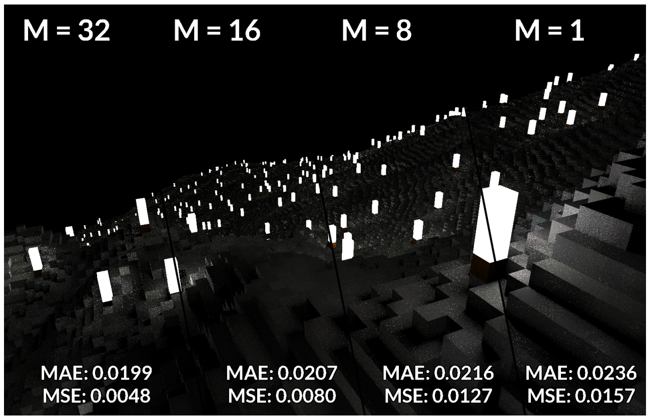


Figure 13: Mountain scene 2

In Figure 13 we demonstrate the mountain scene when using various number of initial candidates. As noted before the mountain scene with its distinct geometry hampers spatial reuse. As spatiotemporal reuse is a mechanism to effectively resample many more samples than the initial candidate sampling, having a scene where reuse is not effective demonstrates the impact of a smaller number of initial candidates best.

In Figures 14 to 17 we test the impact on image quality of the point and area light sampling process.

In Figure 14 we demonstrate the difference between point and area light sources using the Cornell box scene. The difference is best observed on the upper parts of the walls where artifacts from point light sampling are clearly visible. This scene shows the short comings of point light sampling best as the small number of light sources are in close proximity. Many flat surfaces allow for visibility reuse through spatiotemporal reuse.

In Figure 15 we demonstrate the difference between point and area light sources using the flying apartments scene. The difference is best observed on the railing of the bridge. The highlights of the voxel close to this railing are missing. Occlusion artifacts are also prominent on the floor of the bridge near the edges.

In Figure 16 we show the difference between point and area light sources using the emissive letters scene. The many lights embedded into the wall cause this scene with otherwise little occlusions to suffer from obvious occlusion artifacts when using point light sampling. Note the difference in brightness between the left and middle parts of the image.

In Figure 17 we demonstrate the difference between point and area light sources using the mountain scene. There are obvious occlusion artifacts on the floor near the lights in the case of point light sampling. When using point light sampling the highlights are not as well defined due to these artifacts. Note the level of noise in corners, especially well visible in the corners close to the camera, as both methods struggle in such cases.

In Table 1 we show the difference in kernel runtime between the methods. Note that only the initial candidate sampling stage differs between the two methods. The other stages share the same code paths and should result in roughly equal kernel execution times.

Appendix A contains more images using the same scenes in a similar setting as the figures discussed in this section.

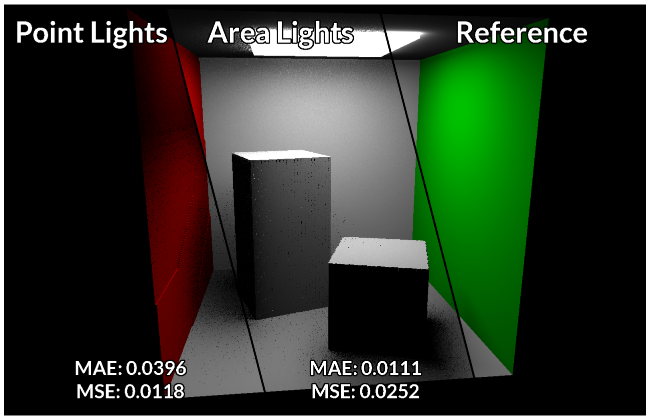


Figure 14: Cornell box scene with 256 emitting bricks

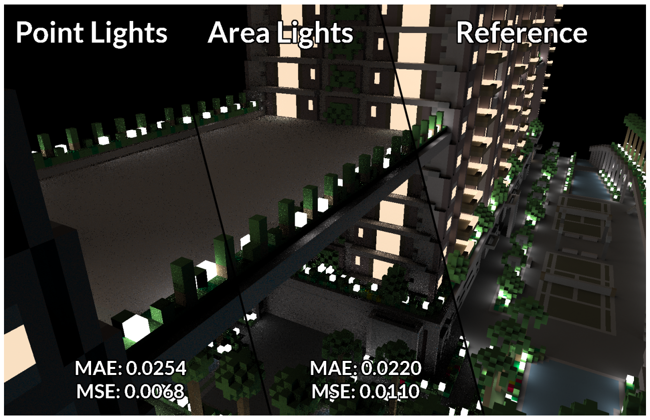


Figure 15: Flying apartments scene 2

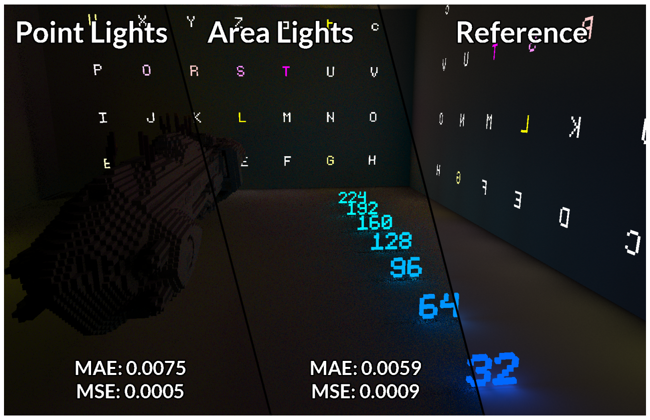


Figure 16: Emissive letters scene 1

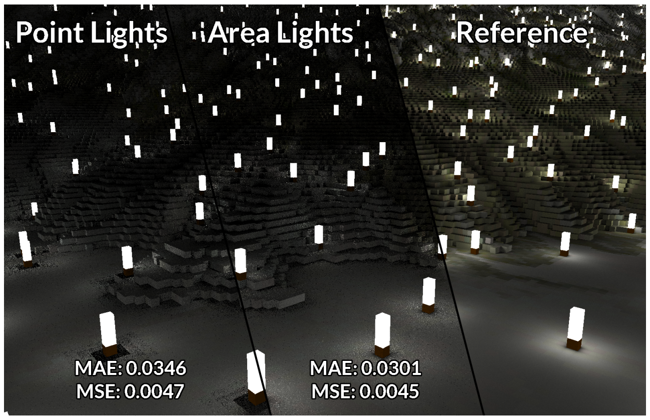


Figure 17: Mountain scene 1

6 DISCUSSION

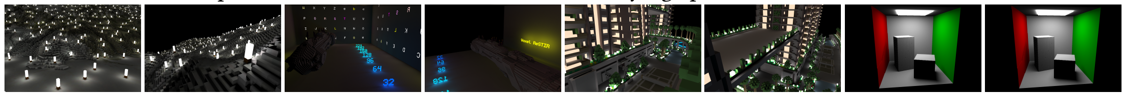
As outlined in Section 3 our goal is to determine if our voxel ReSTIR implementation produces valid results. To do this we compare images rendered using our method to the results obtained by using the ReSTIR implementation in GfxExp in Section 6.1. Next we discuss the performance of our implementation as well as the impact of varying the number of candidates in Section 6.2. Lastly, in Sections 6.3 and 6.4 we discuss the effect of features unique to our implementation in terms of performance and image quality.

6.1 ReSTIR and Voxel ReSTIR

In Figures 4 to 9 we showed the comparison between ReSTIR and our Voxel ReSTIR implementation. The difference in noise between the two implementations is clearly visible. Voxel ReSTIR produces images with lower overall noise. This is confirmed by the MAE values, which can be found in Table 3. A clear example of the difference in noise produced by the two techniques can be seen on the bridge in Figure 6 which is almost noise free when using Voxel ReSTIR.

An interesting point of note is that the ReSTIR image suffers from a lot of dark noise, which means no relevant light could be sampled or the chosen light is occluded. This is can be seen best in Figures 4 and 6. On the other hand the Voxel ReSTIR image has significantly less black noise but does suffer from fireflies as can be seen well in Figure 4 on the white wall in the back near the light. This can be attributed to the target pdf used by Voxel ReSTIR being a compound pdf. This compound pdf importance samples the emitting voxels as well as the side of the voxel using the solid angle. Both ReSTIR implementations are biased implementations which suffer from darkening bias. This is most noticeable in highly occluded regions such as near the walls of the apartments in Figure 5 or at a greater pixel depth such as higher up the mountain in Figure 9. Figure 9 seems like a worst case scenario as the large number of lights often leads to a low quality candidate sampling and the complex geometry prevents effective reuse of neighbouring pixels. Figures 4, 7 and 8 show scenes with little occlusion and as such Voxel ReSTIR is able to perform well with little to no darkening from bias.

In terms of performance both the ReSTIR and Voxel ReSTIR implementation are not quite able to perform in real time. As shown in Table 1 the Voxel ReSTIR implementation is faster for the Mountain and Emissive letter scenes (Figures 7 to 9) whereas the ReSTIR implementation is faster for the Flying apartments and the Cornell box scenes (Figures 4 to 6). Both implementations have a wide range mountain landscape emissive letters flying apartments cornell box



Area light sampling

	mountain landscape		emissive letters		flying apartments		cornell box	
No. emissive voxels	9408		1249		18080		256 (bricks)	131072
albedo (ms)	1.09	1.24	0.90	0.86	0.93	0.85	0.93	0.94
32 candidates (ms)	77.36	68.80	98.13	74.39	100.82	83.44	56.46	82.19
spatial reuse (ms)	23.83	14.38	27.45	18.31	21.74	18.97	14.23	15.77
shading (ms)	12.91	5.50	8.71	6.19	7.85	8.02	3.93	4.04
total time (ms)	115.18	89.92	135.20	99.75	131.35	111.28	75.55	102.94

Point light sampling

	mountain landscape		emissive letters		flying apartments		cornell box	
No. emissive voxels	9408		1249		18080		256 (bricks)	131072
albedo (ms)	1.03	1.21	0.94	0.88	0.97	0.89	0.91	0.95
32 candidates (ms)	22.65	11.30	15.87	12.20	15.15	14.71	11.12	11.05
spatial reuse (ms)	23.87	14.63	27.52	18.60	22.65	20.74	14.79	16.35
shading (ms)	12.49	5.58	8.95	6.09	7.52	7.93	3.67	3.94
total time (ms)	60.03	32.73	53.28	37.78	46.29	44.28	30.49	32.29

GfxExp

	mountain landscape		emissive letters		flying apartments		cornell box	
No. emissive tris	31356		2872		19890		20	
albedo (ms)	10.94	9.04	5.90	5.46	8.11	10.25	3.90	
32 candidates (ms)	136.56	87.36	157.07	117.83	61.15	49.23	22.98	
spatial reuse (ms)	7.66	4.95	8.19	5.22	6.03	5.74	3.97	
shading (ms)	15.15	9.45	7.19	5.80	13.57	12.21	3.40	
total time (ms)	170.31	110.79	178.35	134.31	88.85	77.43	34.25	

Table 1: In this table we show the kernel execution times for the stages of our rendering pipeline and GfxExp. The kernel execution times are profiled using OpenCL profiling functions in milliseconds. The albedo stage serves as primary ray intersection stage, also known as the G-buffer stage. The 32 candidate stage is the initial candidate and temporal reuse stage. Our default number of initial candidates is 32. Temporal reuse is enabled in every scenario we profiled. 1 brick light is 8^3 emissive voxels.

of candidate sampling time which can indicate cache thrashing as explained by [61]. This would explain why candidate sampling is generally done faster with fewer light sources when the number of iterations (candidates) in the kernel remains the same. The greedy meshing technique used to convert the voxel worlds to polygon worlds also works in favor for the ReSTIR implementation for scenes with very simple geometry such as the Cornell box scene (Figure 4). The number of light sources is brought down from 256 bricks (or 131072 voxels) to just 20 emissive triangles.

6.2 Varying the number of candidates

In Bitterli et al. the authors chose 32 initial candidates per pixels as middle ground between performance and quality. We are interested if we can achieve acceptable amounts of noise when resampling using a lower number of initial candidates. Since the number of candidates has significant impact on the execution time of the candidate sampling stage, as can be seen in Table 2. In this table we can see that the MAE when using just 8 candidates is within 10% of the MAE when using 32 candidates. As seen in Figures 10 to 13 the difference between using various number of candidates is small and even using just 1 candidate gives good results. Using 1 candidate results in a MAE lower than the equivalent MAE when using GfxExp with 32 candidates, however the noise becomes more evident.

In Figure 11 the number of fireflies are more obvious when using 1 candidate compared to 32 candidates. A firefly is often caused by sampling a sample with a high adjusted weight as the adjusted weight is used to weight the contribution of the sample when shading a pixel. The adjusted weight is inversely related to the target pdf value of the selected candidate. If we happen to sample a side of an emitting voxel with relatively small solid angle, out of the total solid angle of the voxel, the target pdf value is small and the adjusted weight value is large. However, since the pdf value is relatively small, the probability to retain this candidate becomes smaller as we resample more candidates. This explains why fireflies are less obvious when increasing the number of candidates. On the other hand, in Figure 12 there are little to no fireflies as many of the emissive voxels are enclosed which leaves only a single side of the voxel visible to the scene. The variance in target pdf values when sampling these voxels is eliminated by the visibility check.

mountain landscape
emissive letters
flying apartments
cornell box

Area light sampling

	mountain landscape		emissive letters		flying apartments		cornell box	
No. emissive voxels	9408		1249		18080		256 (bricks)	131072
32 candidates (ms)	77.36	68.80	98.13	74.39	100.82	83.44	56.46	82.19
16 candidates (ms)	39.07	30.73	40.82	31.26	41.32	33.98	25.67	37.61
8 candidates (ms)	27.93	15.54	20.53	15.53	19.43	18.80	12.45	17.20
1 candidate (ms)	15.05	6.03	14.74	10.60	8.78	9.55	5.96	6.85

Point light sampling

	mountain landscape		emissive letters		flying apartments		cornell box	
No. emissive voxels	9408		1249		18080		256 (bricks)	131072
32 candidates (ms)	22.65	11.30	15.87	12.20	15.15	14.71	11.12	11.05
16 candidates (ms)	19.98	8.65	12.17	9.51	10.99	10.93	7.30	8.95
8 candidates (ms)	17.98	7.14	11.19	8.29	9.57	9.70	5.36	7.56
1 candidate (ms)	12.26	4.50	12.83	9.10	7.35	8.35	4.58	5.57

GfxExp

	mountain landscape		emissive letters		flying apartments		cornell box	
No. emissive tris	31356		2872		19890		20	
32 candidates (ms)	136.56	87.36	157.07	117.83	61.15	49.23	22.98	

Table 2: In this table we show the kernel execution times for the candidate sampling stage with various number of candidates. Note that this stage includes temporal reuse.

One important thing to mention is that static images such as our figures cannot convey the annoyance from temporal noise at a low number of candidates.

The ideal number of candidates is depends on the amount of noise that is deemed acceptable. The amount of acceptable noise depends on how much noise can be filtered by a denoiser implementation, as the amount of noise is likely too distracting to use our implementation in a game without a denoiser. Unfortunately we did not test our setup in conjunction with any denoiser implementations.

As shown in Table 2 the performance gain when using a smaller number of candidates is significant. Again, this could indicate a bandwidth bottleneck such as cache thrashing. Note that the initial candidates are uniformly generated in our implementation. Using importance sampling to generate initial candidates could lead to reduced noise, especially for a smaller number of candidates. However, this improvement is limited by the difference in importance of different light sources. Which is often not very big in voxel worlds, as most light sources are simply single voxels with similar emission.

6.3 Voxel point lights or area lights

When considering voxels as area lights we need to consider a point on the voxel to sample from. As described in Section 4.4 we do so by importance sampling the solid angle for the sides of the voxel. Doing so results in some performance overhead as can be seen in Table 1. The candidate sampling kernel execution time for 32 candidates for point light sampling is significantly lower than the 32 candidates sampling kernel for area lights. This brings the performance almost in the range of real time for our system. Sampling voxels as point lights may be desirable for some applications such as games which put high priority on performance.

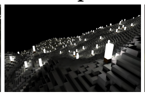
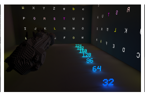
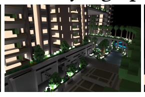
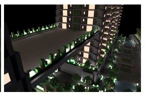
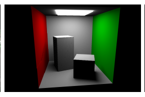
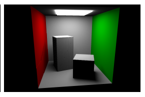
Sampling voxels as point lights does result in some noticeable artifacts. Since the sample point is always at the center of the voxel some occlusions occur which should not occur. Such occlusions can be observed clearly in Figure 17 on the ground near the lights, in Figure 14 on the walls near the light and in Figure 15 on the side of the bridge. Many candidate samplings lead to high potential contributing candidates which subsequently do not pass the visibility test. The same holds for spatial and temporal resampling. The visibility reuse leads to selection of occluded candidates. As a result hard shadows form and lead to occlusion artifacts.

Scenes with many lights and little occlusions such as Figure 16 still suffer from such occlusion artifacts albeit much less obvious. Visibility reuse still leads to shadows in places there should not be such as the shadow on in the back on the left wall of Figure 16.

In return the amount of noise is lower as the sampling point for a given light source is always at the same point in space. The noise does not change as much over time as it does when sampling area lights as the sampling point on the emitting voxel remains consistent. This generally results in a lower amount of perceived noise which can be beneficial when the final image is not denoised.

6.4 Brick lights

One of the features unique to using a voxel world is being able to group geometry easily as a cluster of voxels. This can be done without loss of detail assuming the cluster of voxels is the a homogeneous group of voxels. As described in Section 4.2 we implemented this to group blocks of 8^3 equal emitting voxels into a single brick

	mountain landscape		emissive letters		flying apartments		cornell box	
								

Area light sampling

No. emissive voxels	9408		1249		18080		256 (bricks)	131072
32 candidates MSE	0.0045	0.0048	0.0009	0.0013	0.0116	0.0110	0.0252	0.1660
16 candidates MSE	0.0063	0.0080	0.0097	0.0022	0.0086	0.1117	0.0086	0.0127
8 candidates MSE	0.0088	0.0127	0.0026	0.0015	0.0841	0.0381	0.0068	0.0536
1 candidate MSE	0.0252	0.0157	0.0058	0.0039	0.0520	0.4294	0.0135	0.0111
32 candidates MAE	0.0301	0.0199	0.0059	0.0067	0.0235	0.0220	0.0111	0.0142
16 candidates MAE	0.0311	0.0207	0.0060	0.0068	0.0239	0.0230	0.0111	0.0139
8 candidates MAE	0.0322	0.0216	0.0059	0.0068	0.0253	0.0237	0.0110	0.0143
1 candidate MAE	0.0354	0.0236	0.0061	0.0070	0.0277	0.0260	0.0114	0.0145

Point light sampling

No. emissive voxels	9408		1249		18080		256 (bricks)	131072
32 candidates MSE	0.0047	0.0043	0.0005	0.0010	0.0055	0.0068	0.0118	0.0140
32 candidates MAE	0.0346	0.0215	0.0075	0.0089	0.0290	0.0254	0.0396	0.0430

GfxExp

No. emissive tris	31356		2872		19890		20
32 candidates MSE	0.0760	0.0526	0.0279	0.0047	0.0379	0.0527	0.0217
32 candidates MAE	0.0491	0.0316	0.0133	0.0101	0.0410	0.0406	0.0279

Table 3: In this table we show the error values for our renders when compared to the ground truth.

light source. Only the Cornell box scene contains a setup of emitting voxels that could be turned into brick lights.

In Figure 3 we show the difference between using 256 brick lights on the left and the equivalent 131072 emitting voxels on the right. As can been seen in this figure, grouping the voxels as bricks leads to less noise. The smaller number of lights in the pool of lights leads to a candidate sampling more representative of the entire pool.

In Tables 1 and 2 we show the MAE values and kernel execution times for both setups. Grouping emitting voxels into single light sources outperforms the equivalent setup both in MAE as well as kernel execution time of the candidate sampling. This can be explained by the smaller number of light sources when grouping emitting voxels, as the time to sample candidates increases with the number of light sources.

As mentioned before, using a form of importance sampling when generating initial candidates could improve the results of grouping voxels into bricks even further as the difference in importance between a brick light and a voxel light is substantial. Sadly we did not manage to implement a form of importance sampling for initial candidates.

7 CONCLUSION

We have implemented two variants of Voxel ReSTIR in the WrldTmpl8 framework. Our point light Voxel ReSTIR implementation considers voxel lights as point lights and is able to deliver low noise images at near real-time performance on our system at the cost of artifacts common to point light sampling, such as hard shadows. It is able to outperform a ReSTIR implementation in terms of required computing time. Our area light Voxel ReSTIR implementation considers voxel lights as area light sources and is able to deliver low noise artifact free images at the cost of a performance hit over our point light Voxel ReSTIR implementation. We showed that our area light Voxel ReSTIR implementation outperforms a ReSTIR implementation for polygon scenes in terms of image quality and in most cases in terms of computing time. Our algorithms rely on the same simple image-space data structures as ReSTIR and are suitable for GPU implementation.

We exploit the fact that voxel light sources consist of six emitting sides. We use this to our advantage by importance sampling the sides during candidate sampling and incorporating this into the potential contribution of the light source which is used as our target PDF. Another property of the voxel world we explored is grouping voxel light sources positioned in a 8^3 homogeneous cube into a single light source. This reduction of the number of light sources proved fruitful as it leads to higher quality candidate sampling as well as faster candidate sampling.

In short our contributions are:

- A literature study about the current state of computer graphics using voxel acceleration structures, as well as the state of the art solutions to the many lights problem.
- An implementation of Voxel ReSTIR[1] and description of our implementation.

[1]Voxel ReSTIR repository: https://github.com/xanderhermans/WrldTmpl8

- A comparison between ReSTIR and Voxel ReSTIR. Demonstrating visual differences side by side as well as error values in relation to a path traced reference image.

7.1 Limitations and future work

It would be interesting to explore different sizes of grouping of homogeneous space of light sources. This results in further reducing the number of light sources. The difference in potential contribution of the grouped lights can be exploited when using importance sampling for candidate generation. The performance impact of grouping homogeneous spaces can be kept low by using multilayered data structures such as the two layer grid in WrldTmpl8. This two layer grid allows us to check in constant time if a brick of 8^3 voxels is homogeneous.

As described before, area light sampling becomes problematic when the angle to the emitting surface approaches 0. Using MIS can alleviate this problem at the cost of tracing an extra ray, however tracing an extra ray is relatively cheap for a voxel world. Exploring the possibility of implementing MIS together with Voxel ReSTIR could further reduce noise with limited overhead.

We have shown that Voxel ReSTIR achieves better quality images than ReSTIR and we attribute this to the fact that we effectively importance sample the side of the light source with best potential contribution. Due to the cube shape we always have at least one side with a positive solid angle. It could be worthwhile to group emitting surfaces for ReSTIR, for example based on what mesh they originate from. It would serve as a form of stratification over the orientation of emitting surfaces. Although this increases the time complexity of the candidate sampling it could lead to higher quality candidate sampling, as it is more likely to sample surfaces with different normal vectors. Sampling emitting surfaces with different normal vectors means a higher probability of at least one surface with potential contribution greater than 0.

The time used for the candidate sampling stage has high variance. As Wyman and Panteleev noted this is most likely due to cache thrashing. It could be interesting to explore the candidate sampling improvement and other improvements as suggested by Wyman and Panteleev and how these improvements could be applied to Voxel ReSTIR.

Our implementation relies on image-space methods for resampling. This is visible as small and dim emitters at a distance are generally undersampled. By using a world-space approach such as ReGIR one can improve the candidate sampling and resampling in such cases. If this could be applied to Voxel ReSTIR it could improve image quality as well as possibly exploit the data structure used to store the voxels as it similar to the structure used to store the reservoirs in world space.

REFERENCES

[1] Mitko Aleksandrov, Sisi Zlatanova, and David Heslop. 2021. Voxelisation Algorithms and Data Structures: A Review. *Sensors* 21 (12 2021), 8241. https://doi.org/10.3390/s21248241

[2] John Amanatides and Andrew Woo. 1987. A Fast Voxel Traversal Algorithm for Ray Tracing. *Proceedings of EuroGraphics* 87 (08 1987).

[3] Various authors. 2022. *Lighting in Minecraft.* https://minecraft.fandom.com/wiki/Light

[4] Jeroen Baert, Ares Lagae, and Philip Dutré. 2013. Out-of-Core Construction of Sparse Voxel Octrees. In *Proceedings of the 5th High-Performance Graphics Conference* (Anaheim, California) *(HPG '13)*. Association for Computing Machinery, New York, NY, USA, 27–32. https://doi.org/10.1145/2492045.2492048

[5] Sean Barrett. 2015. *Stb voxel render.* https://github.com/nothings/stb/blob/master/stb_voxel_render.h

[6] Jacco Bikker. 2007. Real-time Ray Tracing through the Eyes of a Game Developer. *2007 IEEE Symposium on Interactive Ray Tracing* (2007), 1–1. https://doi.org/10.1109/RT.2007.4342583

[7] Jacco Bikker. 2022. *WrldTmpl8: Retro coding in C/C++ in a 3D template with full low-level control.* https://github.com/jbikker/WrldTmpl8

[8] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. 2020. Spatiotemporal Reservoir Resampling for Real-Time Ray Tracing with Dynamic Direct Lighting. *ACM Trans. Graph.* 39, 4, Article 148 (jul 2020), 17 pages. https://doi.org/10.1145/3386569.3392481

[9] Guillaume Boissé. 2021. *WORLD-SPACE SPATIOTEMPORAL RESERVOIR REUSE FOR RAY-TRACED GLOBAL ILLUMINATION.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3478512.3488613

[10] Jakub Boksansky, Paula Jukarainen, and Chris Wyman. 2021. Rendering Many Lights with Grid-Based Reservoirs. *Ray Tracing Gems II* (2021), 351–365.

[11] Robert Edward Bridson. 2003. *Computational Aspects of Dynamic Surfaces.* Ph.D. Dissertation. Stanford, CA, USA. Advisor(s) Fedkiw, Ronald. AAI3090563.

[12] M. T. CHAO. 1982. A general purpose unequal probability sampling plan. *Biometrika* 69, 3 (12 1982), 653–656. https://doi.org/10.1093/biomet/69.3.653 arXiv:https://academic.oup.com/biomet/article-pdf/69/3/653/591311/69-3-653.pdf

[13] Per H. Christensen and Dana Batali. 2004. An Irradiance Atlas for Global Illumination in Complex Production Scenes. In *Eurographics Workshop on Rendering*, Alexander Keller and Henrik Wann Jensen (Eds.). The Eurographics Association. https://doi.org/10.2312/EGWR/EGSR04/133-141

[14] Alejandro Conty Estevez and Christopher Kulla. 2018. Importance Sampling of Many Lights with Adaptive Tree Splitting. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2, Article 25 (aug 2018), 17 pages. https://doi.org/10.1145/3233305

[15] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. 2011. Interactive Indirect Illumination Using Voxel Cone Tracing. *Computer Graphics Forum* 30, 7 (2011), 1921–1930. https://doi.org/10.1111/j.1467-8659.2011.02063.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.02063.x

[16] Brian Curless and Marc Levoy. 1996. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 303–312. https://doi.org/10.1145/237170.237269

[17] Carsten Dachsbacher and Marc Stamminger. 2005. Reflective Shadow Maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (Washington, District of Columbia) *(I3D '05)*. Association for Computing Machinery, New York, NY, USA, 203–231. https://doi.org/10.1145/1053427.1053460

[18] Pavlos S. Efraimidis. 2015. Weighted Random Sampling over Data Streams. arXiv:1012.0256 [cs.DS]

[19] Pavlos S. Efraimidis and Paul G. Spirakis. 2006. Weighted random sampling with a reservoir. *Inform. Process. Lett.* 97, 5 (2006), 181–185. https://doi.org/10.1016/j.ipl.2005.11.003

[20] Pascal Guehl. 2013. GigaVoxels, Real-time Voxel-based Library to Render Large and Detailed Objects. *NVIDIA GTC - GPU Technology Conference* (mar 2013). http://maverick.inria.fr/Publications/2013/Gue13

[21] Rama Karl Hoetzlein. 2016. GVDB: Raytracing Sparse Voxel Database Structures on the GPU. In *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*, Ulf Assarsson and Warren Hunt (Eds.). The Eurographics Association. https://doi.org/10.2312/hpg.20161197

[22] Ben Houston, Frantic Films, Mark Wiebe, and Chris Batty. 2004. RLE sparse level sets. (01 2004). https://doi.org/10.1145/1186223.1186394

[23] A. Iones, A. Krupkin, Mateu Sbert, and S. Zhukov. 2003. Fast, realistic lighting for video games. *Computer Graphics and Applications, IEEE* 23 (06 2003), 54– 64. https://doi.org/10.1109/MCG.2003.1198263

[24] Johannes Jendersie, David Kuri, and Thorsten Grosch. 2016. Precomputed Illuminance Composition for Real-Time Global Illumination. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Redmond, Washington) *(I3D '16)*. Association for Computing Machinery, New York, NY, USA, 129–137. https://doi.org/10.1145/2856400.2856407

[25] Viktor Kämpe, Erik Sintorn, and Ulf Assarsson. 2013. High Resolution Sparse Voxel DAGs. *ACM Trans. Graph.* 32, 4, Article 101 (jul 2013), 13 pages. https://doi.org/10.1145/2461912.2462024

[26] Samuli Laine and Tero Karras. 2011. Efficient Sparse Voxel Octrees. *IEEE Transactions on Visualization and Computer Graphics* 17, 8 (2011), 1048–1059. https://doi.org/10.1109/TVCG.2010.240

[27] Daqi Lin*, Markus Kettunen*, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman. 2022. Generalized Resampled Importance Sampling: Foundations of ReSTIR. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2022)* 41, 4, Article 75 (07 2022), 23 pages. https://doi.org/10.1145/3528223.3530158 (*Joint First Authors).

[28] Daqi Lin and Cem Yuksel. 2019. Real-Time Rendering with Lighting Grid Hierarchy. *Proc. ACM Comput. Graph. Interact. Tech.* 2, 1, Article 8 (jun 2019), 17 pages. https://doi.org/10.1145/3321361

[29] Daqi Lin and Cem Yuksel. 2020. Real-Time Stochastic Lightcuts. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 1, Article 5 (apr 2020), 18 pages. https://doi.org/10.1145/3384543

[30] Charles Loop, Cha Zhang, and Zhengyou Zhang. 2013. Real-Time High-Resolution Sparse Voxelization with Application to Image-Based Modeling. In *Proceedings of the 5th High-Performance Graphics Conference* (Anaheim, California) *(HPG '13)*. Association for Computing Machinery, New York, NY, USA, 73–79. https://doi.org/10.1145/2492045.2492053

[31] Jeffrey Mahovsky and Brian Wyvill. 2004. Fast Ray-Axis Aligned Bounding Box Overlap Tests with Plucker Coordinates. *Journal of Graphics Tools* 9 (01 2004). https://doi.org/10.1080/10867651.2004.10487597

[32] Alexander Majercik, Cyril Crassin, Peter Shirley, and Morgan McGuire. 2018. A Ray-Box Intersection Algorithm and Efficient Dynamic Voxel Rendering. *Journal of Computer Graphics Techniques (JCGT)* (2018). https://jcgt.org/published/0007/03/04/

[33] Erison Miller Santos Mesquita, Creto Augusto Vidal, Joaquim Bento Cavalcante-Neto, and Rafael Fernandes Ivo. 2021. Non-overlapping geometric shadow map. *Computers & Graphics* (2021), 59–71. https://doi.org/10.1016/j.cag.2021.08.013.

[34] mikolalysenko. 2012. *An Analysis of Minecraft-like Engines.* https://0fps.net/2012/01/14/an-analysis-of-minecraft-like-engines/

[35] mikolalysenko. 2012. *Meshing in a Minecraft Game.* https://0fps.net/2012/06/30/meshing-in-a-minecraft-game/

[36] Mojang. 2022. *Minecraft.* Mojang. https://www.minecraft.net

[37] Pierre Moreau, Matt Pharr, and Petrik Clarberg. 2019. Dynamic Many-Light Sampling for Real-Time Ray Tracing. *High-Performance Graphics - Short Papers* (2019). https://doi.org/10.2312/hpg.20191191

[38] Ken Museth. 2013. VDB: High-Resolution Sparse Volumes with Dynamic Topology. *ACM Trans. Graph.* 32, 3, Article 27 (jul 2013), 22 pages. https://doi.org/10.1145/2487228.2487235

[39] Ken Museth. 2021. NanoVDB: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation. , Article 1 (2021), 2 pages. https://doi.org/10.1145/3450623.3464653

[40] Ola Olsson, Markus Billeter, Erik Sintorn, Viktor Kämpe, and Ulf Assarsson. 2015. More Efficient Virtual Shadow Maps for Many Lights. *IEEE Transactions on Visualization and Computer Graphics* 21, 6 (2015), 701–713. https://doi.org/10.1109/TVCG.2015.2418772

[41] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and Jacopo Pantaleoni. 2021. ReSTIR GI: Path Resampling for Real-Time Path Tracing. *Computer Graphics Forum* 40 (12 2021), 17–29. https://doi.org/10.1111/cgf.14378

[42] Martin Pätzold and Andreas Kolb. 2015. Grid-Free out-of-Core Voxelization to Sparse Voxel Octrees on GPU. In *Proceedings of the 7th Conference on High-Performance Graphics* (Los Angeles, California) *(HPG '15)*. Association for Computing Machinery, New York, NY, USA, 95–103. https://doi.org/10.1145/2790060.2790067

[43] Francisco Sans and Esmitt Ramirez. 2013. Real-Time Diffuse Global Illumination based on Voxelization. *Proceedings of the 2013 39th Latin American Computing Conference, CLEI 2013.* https://doi.org/10.1109/CLEI.2013.6670656

[44] Michael Schwarz and Hans-Peter Seidel. 2010. Fast Parallel Surface and Solid Voxelization on GPUs. *ACM Trans. Graph.* 29, 6, Article 179 (dec 2010), 10 pages. https://doi.org/10.1145/1882261.1866201

[45] Henry Schäfer, Jochen Süßmuth, Cornelia Denk, and Marc Stamminger. 2012. Memory efficient light baking. *Computers & Graphics* 36, 3 (2012), 193–200. https://doi.org/10.1016/j.cag.2011.12.001 Novel Applications of VR.

[46] Scratchapixel. 2016. *A Minimal Ray-Tracer: Rendering Simple Shapes.* https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection

[47] K. Selgrad, J. Müller, C. Reintges, and M. Stamminger. 2016. Fast Shadow Map Rendering for Many-Lights Settings. *Proceedings of the Eurographics Symposium on Rendering: Experimental Ideas Implementations* (2016), 41–47.

[48] Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: A Sparse Paged Grid Structure Applied to Adaptive Smoke Simulation. *ACM Trans. Graph.* 33, 6, Article 205 (nov 2014), 12 pages. https://doi.org/10.1145/2661229.2661269

[49] SEUS. 2022. *Sonic Ether's Unbelievable Shaders*. Sonic Ether. https://www.sonicether.com/seus/

[50] shocker 0x15. 2022. *GfxExp: Sandbox for graphics paper implementation.* https://github.com/shocker-0x15/GfxExp

[51] Jean-François St-Amour, Eric Paquette, and Pierre Poulin. 2005. Soft shadows from extended light sources with penumbra deep shadow maps. *Graphics Interface* 2005 (2005), 105–112.

[52] Che Sun and Emmanuel Agu. 2015. Many-Lights Real Time Global Illumination Using Sparse Voxel Octree. In *Advances in Visual Computing*, George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Ioannis Pavlidis, Rogerio Feris, Tim McGraw, Mark Elendt, Regis Kopper, Eric Ragan, Zhao Ye, and Gunther Weber (Eds.). Springer International Publishing, Cham, 150–159.

[53] Justin F. Talbot. 2005. Importance Resampling for Global Illumination. (2005). https://scholarsarchive.byu.edu/etd/663

[54] D3D Team. 2018. *Announcing Microsoft DirectX Raytracing!* Microsoft. https://devblogs.microsoft.com/directx/announcing-microsoft-directx-raytracing/

[55] Sinje Thiedemann, Niklas Henrich, Thorsten Grosch, and Stefan Müller. 2011. Voxel-Based Global Illumination. In *Symposium on Interactive 3D Graphics and Games* (San Francisco, California) *(I3D '11)*. Association for Computing Machinery, New York, NY, USA, 103–110. https://doi.org/10.1145/1944745.1944763

[56] Petr Vévoda and Jaroslav Křivánek. 2016. Adaptive Direct Illumination Sampling. In *SIGGRAPH ASIA 2016 Posters* (Macau) *(SA '16)*. Association for Computing Machinery, New York, NY, USA, Article 43, 2 pages. https://doi.org/10.1145/3005274.3005283

[57] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. 2005. Lightcuts: A Scalable Approach to Illumination. *ACM Trans. Graph.* 24, 3 (jul 2005), 1098–1107. https://doi.org/10.1145/1073204.1073318

[58] Amy Williams, Steve Barrus, R. Morley, and Peter Shirley. 2005. An Efficient and Robust Ray-Box Intersection Algorithm. *J. Graphics Tools* 10 (01 2005), 49–54. https://doi.org/10.1145/1198555.1198748

[59] Lance Williams. 1978. Casting Curved Shadows on Curved Surfaces. *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques* (1978), 270–274. https://doi.org/10.1145/800248.807402

[60] Kui Wu, Nghia Truong, Cem Yuksel, and Rama Hoetzlein. 2018. Fast Fluid Simulations with Sparse Volumes on the GPU. *Computer Graphics Forum* 37, 2 (2018), 157–167. https://doi.org/10.1111/cgf.13350 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13350

[61] Chris Wyman and Alexey Panteleev. 2021. Rearchitecting Spatiotemporal Resampling for Production. *High-Performance Graphics - Symposium Papers* (2021). https://doi.org/10.2312/hpg.20211281

[62] Cem Yuksel. 2019. Stochastic Lightcuts. In *High-Performance Graphics (HPG 2019)* (Strasbourg, France). The Eurographics Association, 27–32. https://doi.org/10.2312/hpg.20191192

[63] B. Zhang and K. Oh. 2021. Indirect illumination with efficient monte carlo integration and denoising. (2021), 10167–10185. https://doi-org.proxy.library.uu.nl/10.1007/s11042-020-09884-5

[64] Sergey Zhukov, Andrei Iones, and Grigorij Kronin. 1998. An Ambient Light Illumination Model. In *Rendering Techniques*.

APPENDIX A

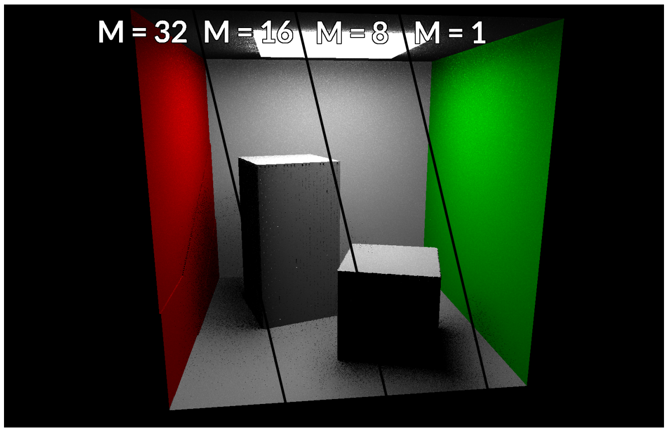


Figure 18: Cornell box scene with 131072 emitting voxels. For error values refer to Table 1

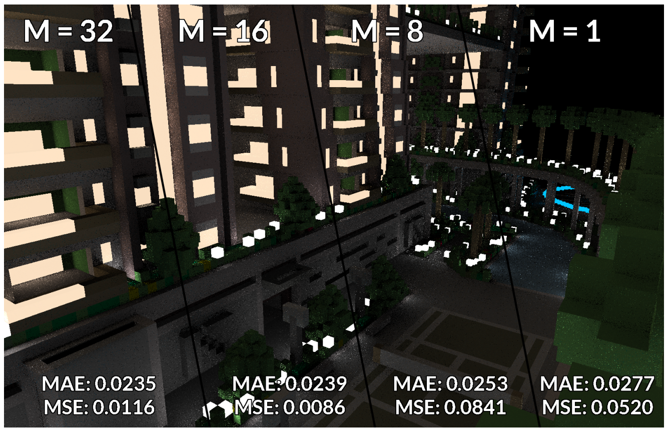


Figure 19: Flying apartments scene 1

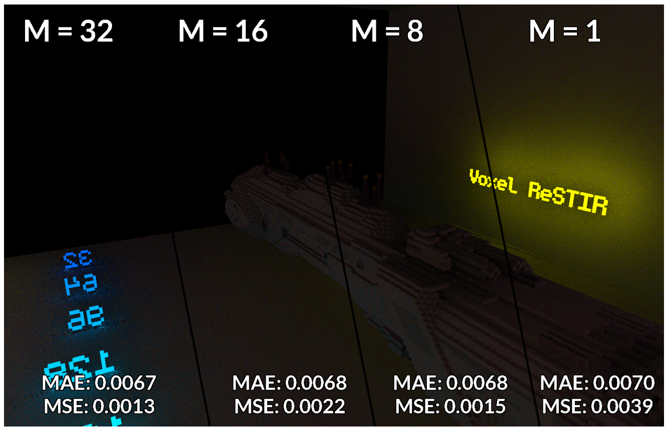


Figure 20: Emissive letters scene 2

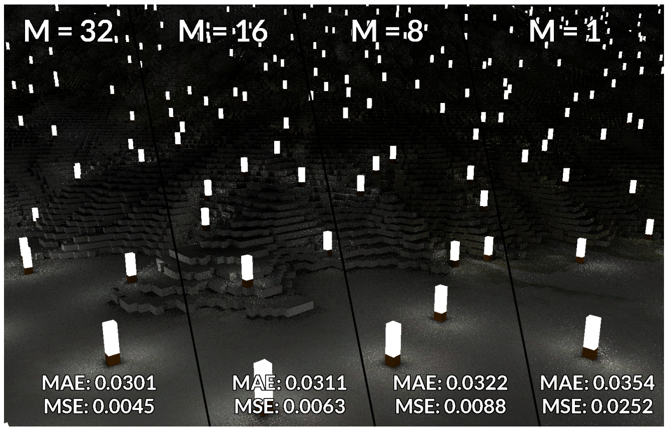


Figure 21: Mountain scene 1

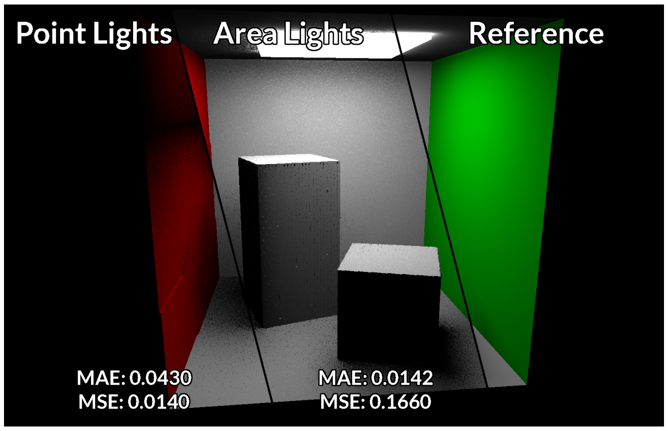


Figure 22: Cornell box scene with 131072 emitting voxels

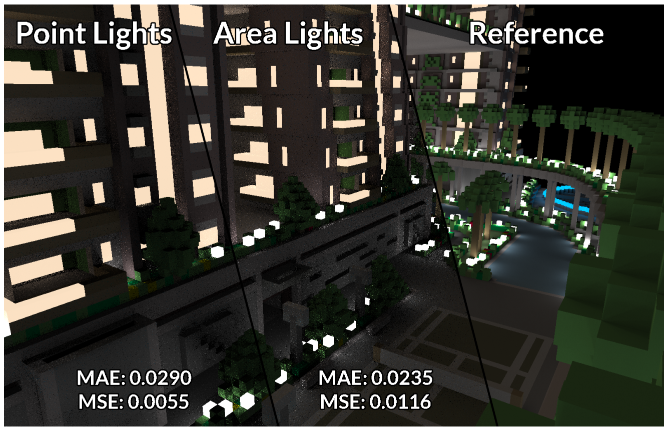


Figure 23: Flying apartments scene 1

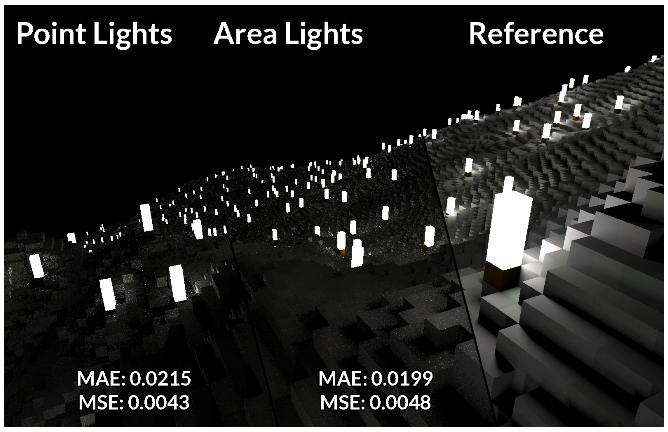


Figure 24: Mountain scene 2

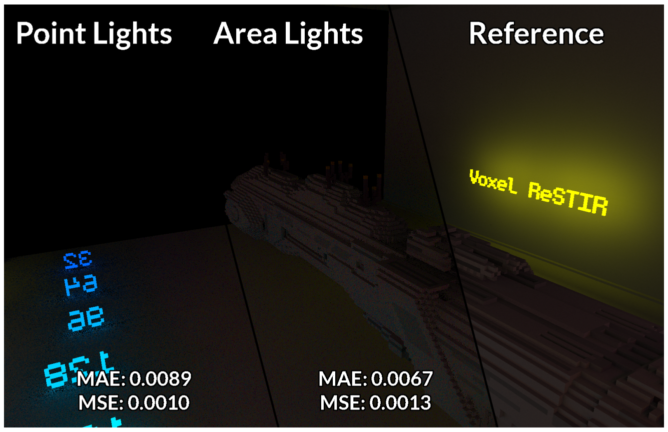


Figure 25: Emissive letters scene 2

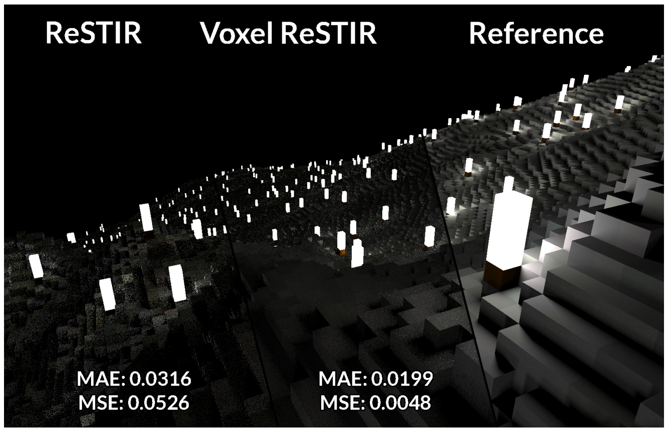


Figure 26: Mountain scene 2