High-Performance Graphics (2023)
J. Bikker and C. Gribble (Editors)

# Efficient Rendering of Participating Media for Multiple Viewpoints

R. Stojanovic[1] and A. Weinrauch[1] and W. Tatzgern[1] and A. Kurz[1] and M. Steinberger[1,2]

[1]Graz University of Technology, Institute of Computer Graphics and Vision, Austria
[2]Huawei Technologies, Austria

**Figure 1:** *Rendering outcome of our method on the San Miguel scene, showcasing an effective representation of participating media effects like fog, dust, and smoke. Our approach prioritizes minimizing redundant computations across multiple viewers, resulting in superior scalability compared to the current state-of-the-art as the number of viewers increases.*

**Abstract**

*Achieving realism in modern games requires the integration of participating media effects, such as fog, dust, and smoke. However, due to the complex nature of scattering and partial occlusions within these media, real-time rendering of high-quality participating media remains a computational challenge.*

*To address this challenge, traditional approaches of real-time participating media rendering involve storing temporary results in a view-aligned grid before ray marching through these cached values. In this paper, we investigate alternative hybrid world- and view-aligned caching methods that allow for the sharing of intermediate computations across cameras in a scene. This approach is particularly relevant for multi-camera setups, such as stereo rendering for VR and AR, local split-screen games, or cloud-based rendering for game streaming, where a large number of players may be in the same location.*

*Our approach relies on a view-aligned grid for near-field computations, which enables us to capture high-frequency shadows in front of a viewer. Additionally, we use a world-space caching structure to selectively activate distant computations based on each viewer's visibility, allowing for the sharing of computations and maintaining high visual quality. The results of our evaluation demonstrate computational savings of up to 50% or more, without compromising visual quality.*

**CCS Concepts**

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

## 1. Introduction

Over the last fifteen years, the trend to shift computations to the cloud has gained increasing popularity. Performing computations in the cloud has the obvious advantages of allowing for easy sharing of resources, access to the latest hardware, mobility, scalability, and cost savings for the end-user. While game streaming services have become more commonplace during that same period, through increased bandwidths and reduced latency, real-time rendering in

the cloud is still largely focused on rendering scenes from a single viewpoint. In shared environments, such as the "Metaverse", or multiplayer gaming worlds, many of the computations are duplicated across all users, e.g. global illumination calculations. By taking advantage of the shared resources when rendering in the cloud, these duplicate computations can be minimized, while at the same time increasing performance when compared to rendering the scene from a single viewpoint without any shared resources. The ability to also stream the rendering results from the cloud to comparatively weak end-user devices is also an advantage, as this not only increases the battery life of such devices but can also decrease the hardware costs of such devices significantly, as the need for the latest hardware can be reduced. Furthermore, the improved performance of decreasing duplicate calculations is not limited to shared environments in the cloud but also applies to a locally shared end-user device. Many console games offer a split-screen mode to accommodate multiple local users, so depending on the location and direction of the different cameras, the impact on rendering times can be significant. This also applies to virtual and augmented reality applications, where an image for each eye has to be generated, and their location and direction only differ slightly. This small difference in location and direction leads to a large overlap in the resulting images.

Real-time rendering encompasses a very wide array of computationally expensive effects, and the most realistic results are usually achieved by ray tracing. In this work, we only focus on a single aspect of real-time rendering, which are the phenomena caused by participating media such as fog, light shafts, smoke, and dust. These phenomena are important components in making a rendered image look believable, while also adding a sense of scale to the scene, and realistic results are usually expensive to calculate. The need for efficient computation of these phenomena for real-time applications is thus still an ongoing research problem. As mentioned previously, the current state-of-the-art approaches [Bau19, Hil15, Wro14] only consider the scene to be rendered from one viewpoint, and as such are not well-suited for a multi-viewer or cloud-rendered approach, as they only provide linear scaling for multiple viewpoints at best.

In this work, we attempt to minimize the duplicate computations inherent in a multi-user environment for participating media effects, when no resources are shared across the different users, as well as to increase performance by using a shared shadow grid that stores ray-traced shadows at world-space-aligned positions.

Specifically, the contributions we present with this work are:

- An extension to traditional froxel-based volumetric rendering methods with a shared shadow grid, and a per viewer shadow frustum, which allows for efficient rendering of participating media effects from many active viewpoints and sub-linear performance scaling.
- An evaluation of the visual quality and performance scaling of our method when compared to a state-of-the-art approach for rendering volumetric fog effects.

## 2. Related Work

Rendering the effects of participating media is usually based on some form of volumetric rendering approach, where a volume is sampled at different locations and the result is an accumulation of all those samples, usually referred to as raymarching. For an overview of volumetric rendering, and effects with participating media, we refer to the report by Novák et al. [NGHJ18].

For volumetric rendering to achieve real-time performance is a long-standing research problem. Some early works by Delalandre et al. [DGMF11] and Gautron et al. [GDM11] calculate maps that simulate the effects of light interacting with the participating media, and could achieve real-time performance. Other approaches attempt to leverage shadow volumes to speed up the raymarching process like in Wyman et al. [WR08].

Another approach is to take advantage of a changed coordinate system, that allows for easier computation of the participating media effects, such as in Baran et al. [BCRK*10], that also added a hierarchical element to further increase performance.

Recent advances in hardware performance, and also the advances in neural rendering, partially based on neural denoising, as can be seen in Hofmann et al. [HHCM21], allow for the rendering of high-quality interactive volumetric effects. In a further combination of image denoising with spatiotemporal reservoir resampling, as seen in the work by Lin et al. [LWY21], nice results at interactive rates can be achieved.

For real-time applications, such as games, the current state-of-the-art methods are typically based on the frustum voxel approach by Wronski [Wro14]. The general idea behind this method revolves around a 3D voxel grid texture, which spans the camera clip space. The resolution of this texture is typically set to occupy 8 screen-space pixels for each voxel in both x- and y-directions, while the amount of depth slices varies from 64 to up to 128 for highly detailed results. These voxels store the scattered radiance that was computed at their world-space centers, as well as the volume density of the participating media at that voxel location. Since these voxels are bound to the camera frustum, they are also often referred to as froxels in literature, and we will also be using that term. For an efficient sampling of the volume from the camera, the in-scattered radiance and media density within the volume grid needs to be accumulated from front to back. For a detailed explanation of this process, and the physical basis behind it, we refer to Akenine-Möller et al. [AMHH*18]. This accumulation results in a new volume that contains the amount of in-scattered light that reaches the camera, and the transmittance over that distance. The resulting fog effects look convincing and can be computed efficiently, despite not being physically based.

Hillaire [Hil15] also used a similar method based on froxels with the difference being that they followed a physically based approach. They include the ability to define the absorption, scattering, emission, and phase function parameter for their participating media volumes explicitly and compute the scattered light using those parameters. Additionally, they propose a method to include and efficiently compute volumetric shadows which are created when the light gets occluded by high-density participating media.

Bauer [Bau19] extends the previous ideas further by using a dynamic depth range for the frustum volume and adding support for rainbows by treating them as an additional light source that can be adjusted via an additional water droplet density material parameter.

They also introduce an efficient raymarching scheme for far-away participating media like clouds outside the frustum volume range.

Sharing computations for cloud rendering was recently explored by the work of Weinrauch and Tatzgern et al. [WTS*23] which focuses on sharing view-independent calculations in object space and a world space aligned grid. Work from Neff et al. [NBD*23] explores sharing rendering computations by sampling points on the surface and storing them in a hash grid.

## 3. Shared Computation of Participating Media

In this section, we will discuss the general idea behind our approach, as well as go into detail about how our pipeline is set up and what kind of data structures are necessary for our approach to function.

Approaches based on froxels are efficient and widely used in production, which is why they also serve as the foundation upon which we build our approach. More specifically, we base our approach on Hillaire [Hil15], while also using it as the reference implementation we will compare our approach against in Section 4. Typically, froxel-based techniques use some variant of shadow mapping to evaluate the shadowing at the froxel centers. Our rendering pipeline does not use shadow mapping but instead uses ray tracing to compute the shadow term of the individual froxels. This not only allows our approach to easily deal with multiple dynamic lights but also with changes in the environment. Even with the hardware acceleration provided by modern GPUs, ray tracing is still computationally expensive, and as a result, the vast majority of the total computational cost for volumetric rendering lies within shadow ray tracing. Our initial testing found that it amounted to roughly 80% of the total cost. For a single viewpoint this is still manageable, but computing the volumetric scattering for multiple viewpoints increases the computational cost linearly with the number of viewpoints, and eventually exceeds the real-time threshold. This is why we were specifically interested in finding a way to share the cost for the shadow computation in the case of a multi-viewer setup. To this end, we extend the traditional froxel-based approaches with a shadow grid that is shared among all viewers, and per viewer local shadow frustums, which are used to replace the shadow mapping step of state-of-the-art approaches.

Our approach consists of the following four stages:

1. **Shared Voxel Update:** ray traced update of the shadow grid data structure (Section 3.1.1)
2. **Shadow Frustum Update:** ray traced update of the local shadow frustum volumes (Section 3.1.2)
3. **Lighting Computation:** evaluate the scattered lighting and estimate the media density (Section 3.1.3)
4. **Volume Ray March:** accumulate the computed scattering & media density (Section 3.1.4)

### 3.1. Pipeline

In the following sections, we will describe every stage of our pipeline in detail. We will also describe the shared shadow grid and shadow frustum volume, and highlight the changes we made to the original pipeline of Hillaire [Hil15] to make them fit our approach. An illustration of the pipeline is provided in Figure 2.

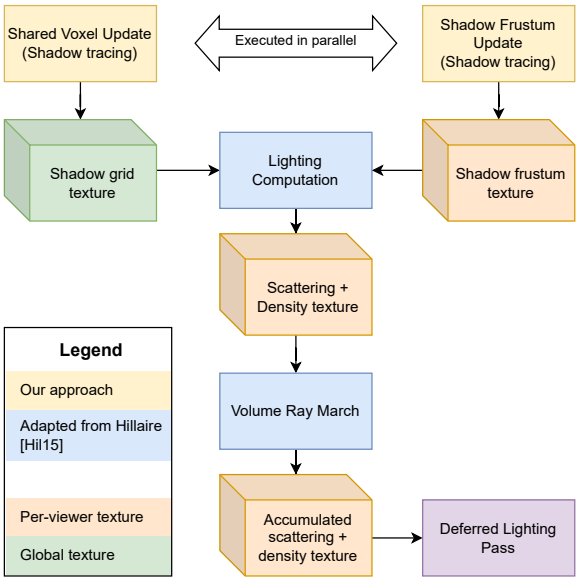



**Figure 2:** *Algorithm pipeline overview. The main differences to standard froxel-based methods are the shared voxel update and shadow frustum update stages. They replace the shadow map lookup in traditional froxel-based methods. The remainder of the pipeline is largely identical to standard froxel-based methods.*

#### 3.1.1. Shared Voxel Update

The main addition of our approach is the shared shadow grid data structure. It is a single world-space-aligned uniform 3D voxel grid, which stores the result of the shadow ray tracing operation as binary shadow values in a single channel 8-bit 3D texture. Each of these 8 bits corresponds to a light source, which allows us to store the results for up to 8 light sources in the form of a bitmask. The single channel 8-bit 3D texture was enough for our testing scenes, as they all contain less than 8 active light sources. For scenes that contain more than 8 active light sources, we recommend using only the 8 light sources, which have the strongest influence over that shadow grid voxel, or use even less, if performance is a concern. One way to implement this efficiently would be by using a method like clustered shading [OBA12]. If more active light sources are desired, it would also be possible to increase the size of the underlying texture, either by increasing the number of channels, or the channel size. Depending on the size increase of the texture, the memory consumption of the shared shadow grid will increase accordingly. An exemplary visualization of the distribution of a shadow grid in the Battle of the Trash God scene is shown in Figure 3.

To avoid confusion with the naming the above voxels will be referred to as (world-space) voxels with the grid they are contained in being called the (shared) shadow grid, while the frustum-aligned voxels will exclusively be referred to as froxels.

The update of the shared shadow grid is done every frame in a single ray tracing shader. We first check for all voxels, whether they are located in at least one of the active camera frustums. This check is done by transforming the world-space position of each voxel center into the clip space of each active camera. If the re-

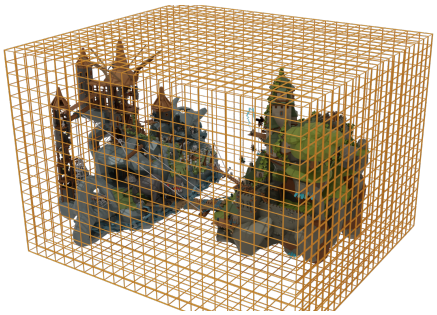


**Figure 3:** *Visualization of the (shared) shadow grid in Battle of the Trash God. The voxel size is increased for a clearer representation.*

sulting coordinates are not contained in any of the active camera frustums, no ray tracing has to be performed for this voxel. If the voxel is visible, i.e. contained in at least one active camera frustum, we shoot one shadow ray towards every light source and check if it can be reached without encountering an occlusion. The result of this occlusion test is then stored as a binary bitmask in the underlying texture of the shadow grid. What this ultimately represents is a bitmask for every (visible) voxel, which contains visibility information for each relevant light source. Since the shadow grid is fixed in place, we found that an alternating update strategy can be utilized without causing any noticeable artifacts. By alternating the updated voxels based on the parity of their z-axis index, we can reduce the total number of traced rays per frame by roughly half. Since the shadow grid itself is fixed in the scene, changes to the lighting and geometry are easily handled by the ray tracing of the shadow rays during the update process. We did not notice any issues regarding temporal inconsistencies even in dynamic scenes, which is likely attributed to the temporal accumulation of the evaluated frustum in a later step in the pipeline (Section 3.1.3).

### 3.1.2. Shadow Frustum Update

The shadow grid is supplemented by an additional local frustum shadow volume for every active viewer in the scene. These froxel shadow volumes store the ray-traced shadows in the same manner as the shadow grid, but each of them is bound to a camera frustum. We modified these shadow volumes in such a way that each froxel spans 16 screen-space pixels in the x- and y-axes, instead of the usual 8, thus increasing the size of the individual froxels. We also only store about a third of the depth slices, as this local shadow volume is only used to capture effects close to the camera. This decrease in total froxel count reduces the update cost per viewer and thus allows for better scaling with the number of viewers at the price of slightly lower visual fidelity. Increasing the size further than the span of 16 pixels leads to a noticeable degradation in quality and has only a negligible impact on performance. To better visualize these froxel volumes, Figure 4 shows a full-sized froxel volume in a scene, as it is used by most froxel-based volumetric rendering methods for evaluating the scattering and density. We also use it in a later pipeline stage (Section 3.1.3). It should be noted, that these are the volumes used in the subsequent parts of the pipeline, and the local shadow volumes feature fewer depth slices and larger individual froxels.

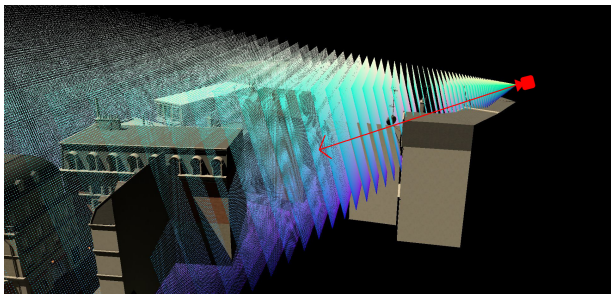


**Figure 4:** *Visualization of the froxel distribution viewed from a third-person perspective. The froxel centers are marked in color. The camera and the view direction are marked in red. These froxel volumes are used in the lighting computation and accumulation steps of the pipeline.*

Although these shadow volumes are not shared across multiple viewpoints, their small resolution allows for very reasonable costs in terms of memory and computations. To further decrease the costs, we combine all the shadow frustum textures into a single large texture by stacking them along the depth axis. This allows us to update them all in a single shader call, saving some additional overhead.

### 3.1.3. Lighting Computation

Now that the bitmasks in the shadow grid texture, and the shadow frustum texture are updated, we closely follow the single-camera procedure from Hillaire [Hil15] to compute the scattered light and estimate the media density of each froxel. As mentioned before, we do not perform a shadow map lookup during the scattering computation but rather sample the shadows from our shadow data structures.

The results of the lighting computation are stored in a per-camera froxel volume, that stores the in-scattered lighting and media density of the participating media. In contrast to our local shadow volume, this froxel volume is the same as it is in the reference solution, so the froxels span 8 pixels in the x- and y-axes and the volume contains the full 128 depth slices. To compute the in-scattered lighting and media density, we sample the local shadow frustum for the first third of the froxels in this new volume, while we sample from the closest world-space voxel in the shadow grid for the remainder of the froxels. This is done for all cameras and the result of this step is the same as in the reference implementation. This volume is also integrated temporally by jittering the sample locations along the individual view rays to reduce noise and aliasing. A simplified top-down visualization of how the shadow frustum and the shadow grid interact with each other can be seen in Figure 5.

Optionally, instead of computing the scattering at just one sample position, we can use a trilinear interpolation technique to reduce the blockiness found with low-resolution grids and sharp light shafts. We implemented this by sampling the 8 closest shadow grid voxels, computing the scattering at their respective center positions, and interpolating the computed values according to the position of the evaluated froxel. This helps to mask the borders between in-

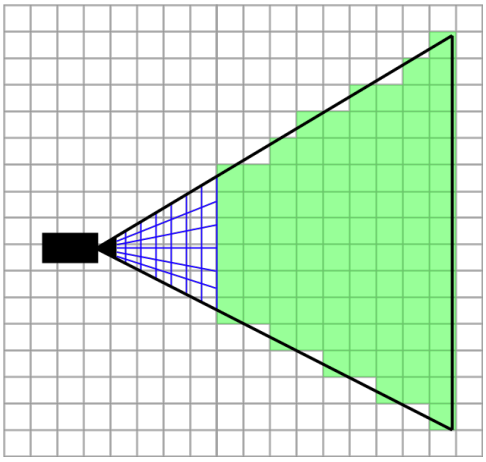


**Figure 5:** *A simplified visualization of the hybrid solution from a top-down view. The camera and its frustum are outlined in black. The shared shadow grid is represented by the grey grid, with the voxels to be updated marked in green. The local shadow frustum volume is represented by the blue grid.*

dividual voxels when larger voxel sizes are used but also slightly increases computational cost.

#### 3.1.4. Volume Ray March

Following the lighting evaluations, the generated frustum volume needs to be accumulated in a final pass. This procedure also follows Hillaire [Hil15], where we perform a 2D ray march inside the frustum texture from front to back, accumulating the scattering and transmittance into a new texture. This is performed for every viewer and provides us with a 3D texture that can be used to sample the accumulated light and transmittance at any point in the corresponding camera's view frustum. In a deferred rendering context we can then use this texture by transforming the world-space position of each pixel into the UV space of the texture and sampling the values at that position. To apply the effect the pixel color is then multiplied with the transmittance and is added to the in-scattered light.

## 4. Evaluation

In this section, we will be comparing our method to a reference implementation based on Hillaire [Hil15] as it is widely used in production and provides results close to what can be achieved with path tracing. The setup is mostly identical to their proposed approach but as mentioned before we use ray tracing instead of shadow mapping to compute the shadow term inside the froxels. This means that our approach can model the same effects of participating media as the reference implementation does. While newer methods like Bauer [Bau19] exist they fundamentally use the same general idea with slight extensions to fit their specific needs. We built our approach on the Falcor [KCK*21] framework by NVIDIA and we evaluate the performance of our approach using a test machine configured with an NVIDIA Geforce RTX 3090 GPU, an AMD Ryzen 9 3900X Processor, and 32GB of RAM. The resolution of every viewer was set to 1920x1080. For the reference implementation, we used the typical span of 8 pixels per froxel for the frustum volume with a depth of 128. At 1080p this amounts to a total frustum resolution of 240x135x128.

In the following sections, we will talk about the shadow grid in terms of actual voxel size. In particular, this voxel size refers to the side length of the voxel in each of the three world-space axes. Meaning a shadow grid with a voxel size of 0.5 meters will have eight times the amount of voxels as a shadow grid with a voxel size of 1 meter. This is done so that it is easier to compare shadow grids in different scenes, as the size differences across multiple scenes can yield radically different grid resolutions.

Section 4.1 contains qualitative comparisons of different voxel sizes for the shadow grid attempting to find an optimal tradeoff for multiple scenes concerning performance and quality. In Section 4.2 we analyze and compare the run time of our approach in detail using various camera setups as well as inspect the scaling with regards to camera overlap. Finally, we provide an in-depth look at the memory consumption of our approach in Section 4.3.

### 4.1. Performance-Quality Tradeoff

The memory consumption and the run time of our approach are mainly determined by the chosen voxel size of the shadow grid. To get a better idea of what kind of quality we can achieve by varying the voxel size, we ran an ablation study for a range of different values and compared the rendered results to our reference solution. To highlight the differences to the reference solution, we used the FLIP error metric [ANAM*20], and these results can be found in Figure 6.

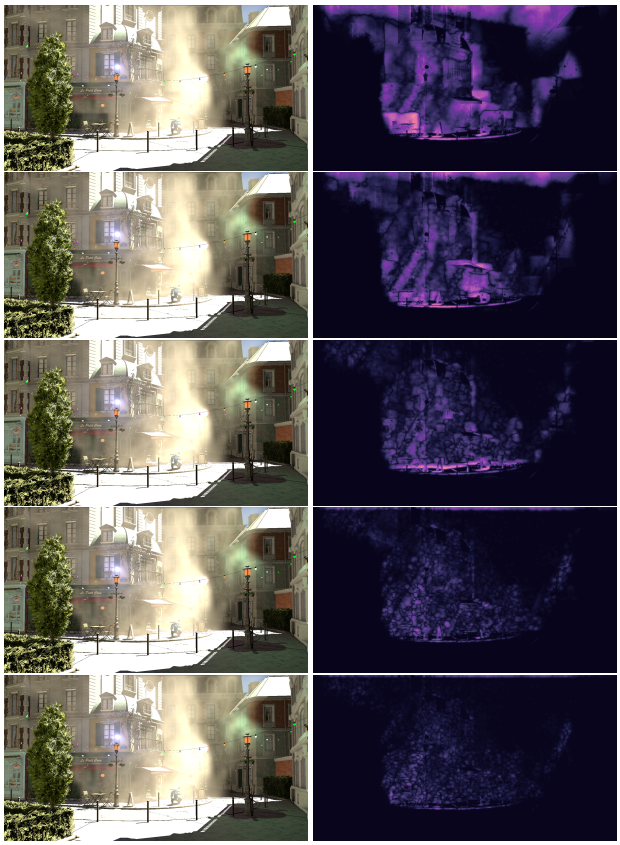
By analyzing the results we obtained during our ablation study, we found that there is no optimal voxel size shared across all tested scenes. Still, some general guidelines can be learned from the results: For larger-scale participating media effects, a voxel size of around 0.5 to 0.333 meters seems to be sufficient to capture adequate detail for rendering purposes. To replicate sharp light shafts, like those that are present in the *Sponza* scene (Figure 7), a smaller size of about 0.1 meters is necessary. This large discrepancy stems from the fact that effects such as light shafts are more visible close to the viewer, and their small size requires an equally fine data structure to properly capture the details for an adequate result.

### 4.2. Run-Time Performance

To ensure a fair comparison between our approach and the reference solution, we carefully selected the voxel size for our shadow grid to achieve results that closely approximate the quality of the reference solution. Additionally, since our shadow grid covers just the extent of the scene, we also modified the reference implementation to only trace rays within the scene bounds. The performance comparison between the reference implementation and our approach can be seen in Tables 1, 2, and 3, as well as in Figures 8, 9, and 10.

These results show us that by using a large voxel size, we can usually achieve an improvement in performance over the reference solution, as soon as a second viewpoint needs to be rendered. When using a small voxel size, the amount of viewers required to see

**(a)** *Result of rendering the scene with the reference implementation.*



**(b)** *Left: Render results using shadow grids with varying voxel sizes. Right: FLIP result compared to the reference implementation. Shadow grid voxel sizes from top to bottom: 4, 2, 1, 0.5, 0.25 meters. Grid resolutions from top to bottom: 44x48x28, 88x94x54, 174x186x106, 348x370x212, 689x740x424.*

**Figure 6:** *Bistro (Exterior) render results.*

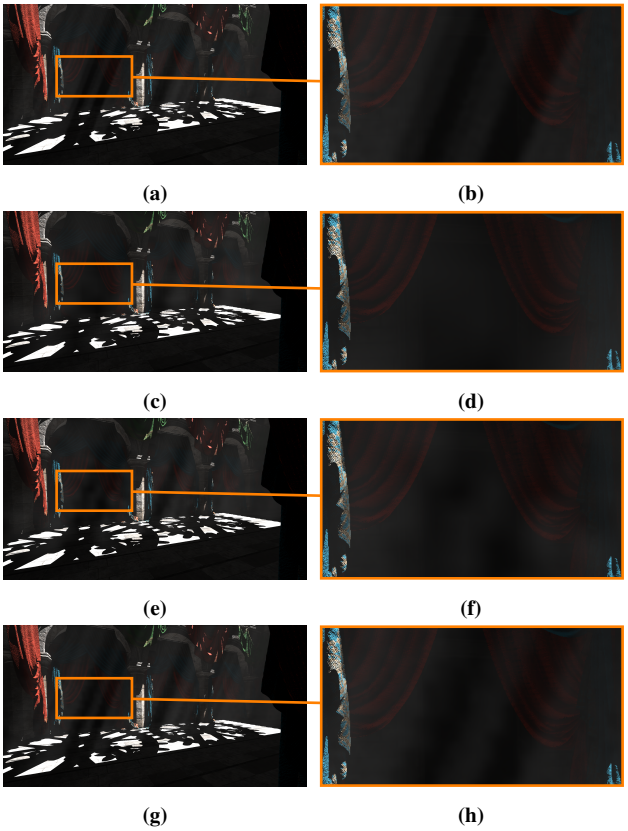



**Figure 7:** *Render results for the Sponza scene. Image a) depicts the result rendered with the reference implementation. The images c), e), g) are rendered using our approach with the following voxel sizes: c) 0.5 meters (74x46x40 grid resolution), e) 0.25 meters (148x92x80 grid resolution), g) 0.1 meters (364x238x200 grid resolution)*

| Cameras | Reference | Shared Shadow Grid |
|---|---|---|
| 1 | 1.10 ms | 0.91 ms |
| 2 | 2.13 ms | 1.57 ms |
| 3 | 3.17 ms | 2.17 ms |
| 4 | 4.06 ms | 2.73 ms |
| 6 | 5.54 ms | 3.57 ms |
| 8 | 6.94 ms | 4.57 ms |
| 12 | 10.68 ms | 6.54 ms |
| 16 | 14.39 ms | 8.57 ms |

**Table 1:** *San Miguel performance comparison using realistic camera positioning and movement. The used voxel size is 0.125 meters, which yields a grid resolution of 554x216x122.*

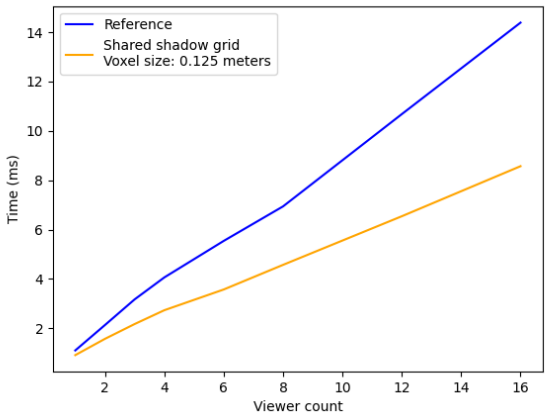



**Figure 8:** *Performance results for the San Miguel scene with different viewer counts. The used voxel size yields a grid resolution of 554x216x122.*

| | | Shared Shadow Grid | |
|---|---|---|---|
| Cameras | Reference | 0.5 meters | 0.333 meters |
| 1 | 4.58 ms | 4.37 ms | 7.82 ms |
| 2 | 9.44 ms | 6.56 ms | 13.73 ms |
| 3 | 13.77 ms | 8.96 ms | 18.41 ms |
| 4 | 17.84 ms | 11.20 ms | 22.28 ms |
| 6 | 26.53 ms | 14.98 ms | 27.48 ms |
| 8 | 34.89 ms | 17.88 ms | 32.28 ms |
| 12 | 52.02 ms | 22.27 ms | 37.60 ms |
| 16 | 68.86 ms | 26.92 ms | 44.37 ms |

**Table 2:** *Bistro (Exterior) performance comparison using realistic camera positioning and movement. The voxel size of 0.5 meters gives us a grid resolution of 348x370x212, while the 0.333 meters version uses a grid with a resolution of 522x554x318.*

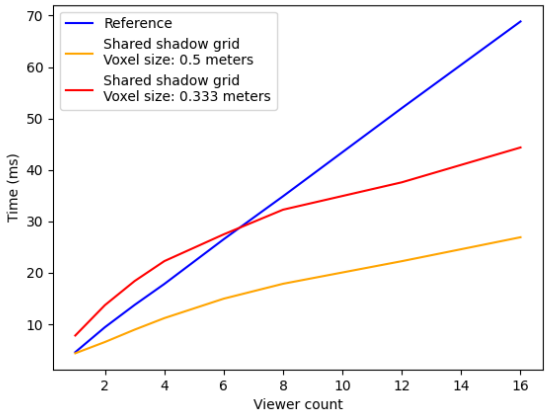



**Figure 9:** *Performance results for the Bistro (Exterior) scene with different viewer counts. The voxel size of 0.5 meters gives us a grid resolution of 348x370x212, while the 0.333 meters version uses a grid with a resolution of 522x554x318.*

| | | Shared Shadow Grid | |
|---|---|---|---|
| Cameras | Reference | 0.333 meters | 0.25 meters |
| 1 | 2.41 ms | 3.64 ms | 7.12 ms |
| 2 | 6.17 ms | 5.91 ms | 10.69 ms |
| 3 | 9.53 ms | 8.27 ms | 14.52 ms |
| 4 | 19.97 ms | 12.88 ms | 21.11 ms |
| 6 | 25.43 ms | 15.63 ms | 24.30 ms |
| 8 | 31.87 ms | 18.57 ms | 27.83 ms |
| 12 | 45.68 ms | 23.70 ms | 32.96 ms |
| 16 | 67.53 ms | 30.49 ms | 39.88 ms |

**Table 3:** *Battle of the Trash God performance comparison using realistic camera positioning and movement. The voxel size of 0.333 gives us a grid resolution of 286x276x278, while the 0.25 meters version uses a grid with a resolution of 380x368x370.*

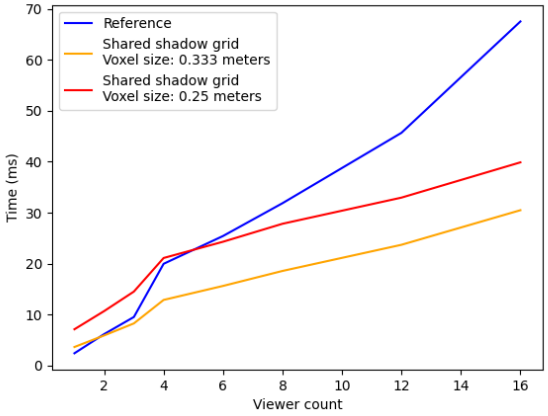



**Figure 10:** *Performance results for the Battle of the Trash God scene with different viewer counts. The voxel size of 0.333 gives us a grid resolution of 286x276x278, while the 0.25 meters version uses a grid with a resolution of 380x368x370.*

a noticeable performance increase over the reference implementation is noticeably higher. Typically, around eight viewpoints need to be rendered for our approach to beat the reference implementation in performance in this case. The reason for the subpar performance with few viewers is in part due to the overhead caused by the additional write and read operations to the shadow grid texture. Additionally, our update procedure is overestimating the voxels required for sampling as we update all voxels within the frustums even if some of the updated voxels are never the closest targets for a sample.

The biggest tested difference in performance between our approach and the reference solution was achieved by having 16 active viewers in the shared scene. In the *San Miguel* scene, this increase amounted to about 40%, while for the *Bistro* scene, the improvement reached 60%, and 35% with the larger, and smaller voxel sizes respectively. The increase in performance remained the same for the remaining scene, *Battle of the Trash God*, where we saw improvements of 55% and 41% for the larger, and smaller voxel sizes respectively. Depending on the chosen voxel size, and the number

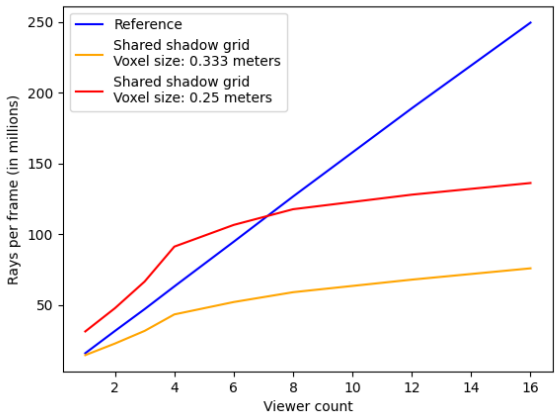



**Figure 11:** *Number of traced rays for the Battle of the Trash God scene with different viewer counts. The voxel size of 0.333 gives us a grid resolution of 286x276x278, while the 0.25 meters version uses a grid with a resolution of 380x368x370.*

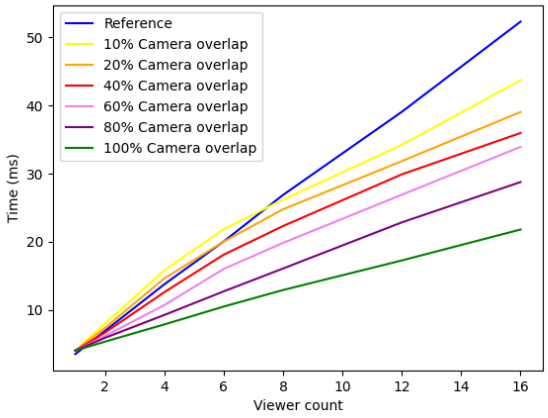



**Figure 12:** *Performance in Bistro (Exterior) in fixed overlap scenarios with a voxel size of 0.333 meters, and grid resolution of 522x554x318.*

of viewers, the number of traced rays can be reduced significantly, which can be seen in Figure 11.

For us to be able to quantify the performance benefits of our approach, we also measured the run time using artificially constructed overlap scenarios at various camera orientations. To this end, we fixed the amount of overlap between the cameras, by duplicating one camera and rotating it in place until the desired overlap in frustums has been achieved. The results of these experiments can be seen in Figure 12.

Even in the worst case of 10% overlap between the cameras, as soon as 8 active viewers are present in the scene, our approach beats the reference implementation. For VR applications, where the two cameras for the eyes are very close together, and the overlap is also very high, we can achieve better performance with a low amount of viewers in the scene.

To better showcase the performance characteristics of our approach, we measured the performance impact of each stage of our approach. The performance impact of each stage can be seen in Figure 13.

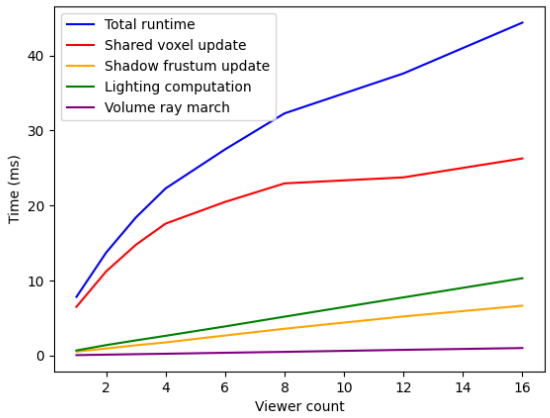



**Figure 13:** *Run time distribution in Bistro (Exterior) with a voxel size of 0.333 meters, and grid resolution of 522x554x318.*

As expected, shadow tracing largely dominates the run time of our approach. With low viewer counts this also leads to a decrease in performance when compared to the reference implementation, depending on the chosen voxel size. By increasing the number of viewers, our approach can only win against the reference implementation, as our approach scales sub-linearly with the number of viewers.

### 4.3. Memory Consumption

The memory footprint of our approach is made up of five different components. Three of these five components are the fully frustum-aligned textures used for storing the (accumulated) in-scattering and participating media density. Next is the additional frustum-aligned texture for the local shadow evaluations, and lastly, we have our shadow grid. The fully frustum-aligned textures are the same used by Hillaire [Hil15], and they span 8 screen-space pixels in both the x- and y-axes and have 128 depth slices, thus each of them requires about 33.2 MB of memory. The texture for the local shadow evaluations only uses a third of the depth slices, just half of the resolution in each x- and y-direction, and also uses a single 8-bit channel, which comes to about 0.4 MB. The memory requirement of these 4 textures scales per viewer in contrast to our shadow grid, which requires only a constant amount of memory.

As was mentioned before, the memory requirements of the shadow grid depend on the size of the scene and the chosen voxel size. With a relatively large voxel size of 0.333 meters, which we used in the *Bistro* scene, the shadow grid would need around 27 MB of memory for a scene the size of 100x100x100 meters. In the case of a scene where smaller voxel sizes are necessary, as was the case to capture enough detail in the *Sponza* scene, with 0.167 meters per voxel, this value would increase to about 216 MB.

In total, we need roughly 100 MB of video memory per viewer in addition to a constant amount for our shadow grid, which depends

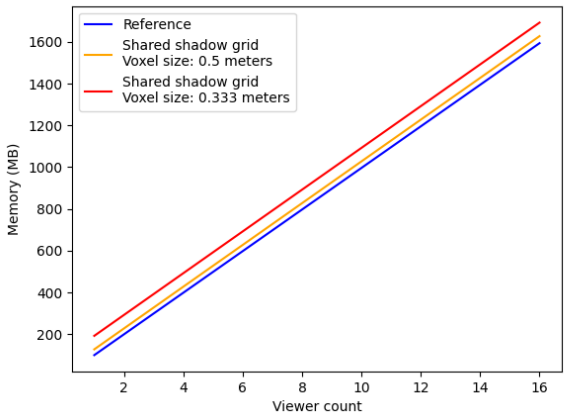



**Figure 14:** *Memory requirements in Bistro (Exterior) at 1080p. The voxel size of 0.5 meters gives us a grid resolution of 348x370x212, while the 0.333 meters version uses a grid with a resolution of 522x554x318.*

on the chosen voxel and scene size. The memory needed for the reference implementation and 2 voxel sizes in the *Bistro* scene is visualized in Figure 14. Compared to the reference implementation, our approach uses 0.4 MB more memory per viewer for the single texture for the local shadow evaluations, which is negligible, and additionally also only uses the constant memory of the shadow grid.

## 5. Limitations & Future Work

This section deals with issues and limitations we face in our current approach as well as ideas on how to remedy them.

### 5.1. Large-scale Scenes

Our approach assumes the scenes to be limited to small or medium sizes, as we span our shadow grid across the whole scene. For large open-world type environments the required resolution of the shadow grid texture would be very large and, because of that, storing the whole grid in memory would be impractical. To handle the memory issues we could divide the scene into multiple cells which each contain their own shadow grid and then load those cells on demand as the viewers reach them.

### 5.2. Hierarchical Approach

Although we use a single-level fixed-resolution shadow grid in our final approach we can use a hierarchical approach as well. During our investigation, we performed some experiments on a basic hierarchical approach based on the shadow grid. Instead of sampling the per viewer froxel shadow frustum, a shared higher-resolution shadow grid is sampled. This higher-resolution shadow grid is created similarly to the previously mentioned shared shadow grid but has a higher resolution to capture finer details. The high-resolution grid is sampled by the froxels close to the camera while the other froxels are sampled from the lower-resolution grid. This approach improves the resulting quality, but the increase in memory consumption, and the poor sharing of the high-resolution shadow grid, which led to poor scaling in the number of viewers, caused us to switch to the local shadow frustum per viewer. The additional memory consumption can be mitigated by having sparse layers or dynamically creating them where required, similar to an octree. Nevertheless, this may be an interesting application for virtual reality in particular, as the two cameras might be able to share a large percentage of the shared higher-resolution shadow grid.

### 5.3. Clouds & Long-Distance Fog

State-of-the-art volumetric rendering approaches like Bauer [Bau19] typically feature a unified rendering approach that includes support for short- as well as long-distance participating media like clouds. Our approach focuses on effects close to the viewer, so the rendering of far-away phenomena such as clouds is currently not well supported with our approach. A quick fix would be to render them using a traditional ray marching approach as suggested in [Bau19] but a future solution could use additional strategies to share computations to better scale with many viewers as explored by Weinrauch et al. [WLT*23].

### 5.4. Surface lights

Our approach is currently limited to analytical lights due to storing one binary visibility term per light source. To extend to surface lights we could approximate and accumulate incoming lighting information with more sophisticated data structures. Examples are explored in the work of Stadlbauer et al. [SWTS23] for caching direct illumination with spherical harmonics or cones spanning incoming light.

## 6. Conclusion

We showcase an efficient method of computing volumetric effects from participating media in scenarios with multiple rendered viewpoints. Our approach is based on existing state-of-the-art volumetric fog methods but additionally uses a world-space-aligned shadow grid, and a shadow frustum per viewer, which both store ray-traced shadow information. While the shadow grid shares its shadow information across all viewers, the shadow frustums only contain shadow information for each viewer. The grid is updated in a manner that avoids duplicated computations and can be sampled from any point in the scene. The split into shadow grid and local shadow frustums allows them to be updated independently of each other. Using our method, we manage to achieve similar quality compared to traditional volumetric fog approaches but with a performance increase of up to 60%.

## Acknowledgements

Figure 1 uses the *San Miguel* scene from McGuire's Computer Graphics Archive [McG17]. Figures 4, 6 use the *Bistro* scene provided by Amazon Lumberyard [Lum17]. The *Sponza* scene used in Figure 7 is originally from Intel [MPS*22] and was adapted to suit our needs. Figure 3 shows the *Battle of the Trash God* scene by Burunduk [Bur22].

## References


[AMHH*18] AKENINE-MÖLLER T., HAINES E., HOFFMAN N., PESCE A., IWANICKI M., HILLAIRE S.: *Real-Time Rendering*, fourth ed. A K Peters/CRC Press, 2018. 2

[ANAM*20] ANDERSSON P., NILSSON J., AKENINE-MÖLLER T., OSKARSSON M., ÅSTRÖM K., FAIRCHILD M. D.: Flip: A difference evaluator for alternating images. *Proc. ACM Comput. Graph. Interact. Tech. 3*, 2 (Aug. 2020). 5

[Bau19] BAUER F.: Creating the atmospheric world of red dead redemption 2: a complete and integrated solution. In *46th International Conference and Exhibition on Computer Graphics & Interactive Techniques, SIGGRAPH 2019* (2019). 2, 5, 9

[BCRK*10] BARAN I., CHEN J., RAGAN-KELLEY J., DURAND F., LEHTINEN J.: A hierarchical volumetric shadow algorithm for single scattering. In *ACM SIGGRAPH Asia 2010 Papers* (New York, NY, USA, 2010), SIGGRAPH ASIA '10, Association for Computing Machinery. URL: https://doi.org/10.1145/1866158.1866200, doi:10.1145/1866158.1866200. 2

[Bur22] BURUNDUK: Flying world - battle of the trash god, May 2022. URL: https://sketchfab.com/3d-models/flying-world-battle-of-the-trash-god-350a9b2fac4c4430b883898e7d3c431f. 9

[DGMF11] DELALANDRE C., GAUTRON P., MARVIE J.-E., FRANÇOIS G.: Transmittance function mapping. In *Symposium on Interactive 3D Graphics and Games* (2011), pp. 31–38. 2

[GDM11] GAUTRON P., DELALANDRE C., MARVIE J.-E.: Extinction transmittance maps. In *SIGGRAPH Asia 2011 Sketches*, SA '11. Association for Computing Machinery, New York, NY, USA, 2011, pp. 1–2. URL: https://doi.org/10.1145/2077378.2077387, doi:10.1145/2077378.2077387. 2

[HHCM21] HOFMANN N., HASSELGREN J., CLARBERG P., MUNKBERG J.: Interactive path tracing and reconstruction of sparse volumes. *Proc. ACM Comput. Graph. Interact. Tech. 4*, 1 (apr 2021). URL: https://doi.org/10.1145/3451256, doi:10.1145/3451256. 2

[Hil15] HILLAIRE S.: Physically based and unified volumetric rendering in frostbite. *SIGGRAPH Advances in Real-Time Rendering course (2015)* (2015). 2, 3, 4, 5, 8

[KCK*21] KALLWEIT S., CLARBERG P., KOLB C., YAO K.-H., FOLEY T., WU L., CHEN L., AKENINE-MOLLER T., WYMAN C., CRASSIN C., BENTY N.: The Falcor Rendering Framework, 2021. URL: https://github.com/NVIDIAGameWorks/Falcor. 5

[Lum17] LUMBERYARD A.: Amazon lumberyard bistro, open research content archive (orca), July 2017. URL: http://developer.nvidia.com/orca/amazon-lumberyard-bistro. 9

[LWY21] LIN D., WYMAN C., YUKSEL C.: Fast volume rendering with spatiotemporal reservoir resampling. *ACM Trans. Graph. 40*, 6 (dec 2021). URL: https://doi.org/10.1145/3478513.3480499, doi:10.1145/3478513.3480499. 2

[McG17] MCGUIRE M.: Computer graphics archive, July 2017. URL: https://casual-effects.com/data. 9

[MPS*22] MEINL F., PUTICA K., SIQUERIA C., HEATH T., PRAZEN J., HERHOLZ S., CHERNIAK B., KAPLANYAN A.: Intel sample library, 2022. URL: https://www.intel.com/content/www/us/en/developer/topic-technology/graphics-processing-research/samples.html. 9

[NBD*23] NEFF T., BUDGE B., DONG Z., SCHMALSTIEG D., STEINBERGER M.: PSAO: Point-Based Split Rendering for Ambient Occlusion. In *High-Performance Graphics - Symposium Papers* (2023), Bikker JaccoGribble C., (Ed.), The Eurographics Association, pp. 1–1111 pages. doi:10.2312/hpg.20231131. 3

[NGHJ18] NOVÁK J., GEORGIEV I., HANIKA J., JAROSZ W.: Monte carlo methods for volumetric light transport simulation. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 551–576. 2

[OBA12] OLSSON O., BILLETER M., ASSARSSON U.: Clustered Deferred and Forward Shading. In *Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics* (2012), Dachsbacher C., Munkberg J., Pantaleoni J., (Eds.), The Eurographics Association. doi:10.2312/EGGH/HPG12/087-096. 3

[SWTS23] STADLBAUER P., WEINRAUCH A., TATZGERN W., STEINBERGER M.: Surface Light Cones: Sharing Direct Illumination for Efficient Multi-viewer Rendering. In *High-Performance Graphics - Symposium Papers* (2023), Bikker JaccoGribble C., (Ed.), The Eurographics Association, pp. 65–7511 pages. doi:10.2312/hpg.20231137. 9

[WLT*23] WEINRAUCH A., LORBEK S., TATZGERN W., STADLBAUER P., STEINBERGER M.: Clouds in the Cloud: Efficient Cloud-Based Rendering of Real-Time Volumetric Clouds. In *High-Performance Graphics - Symposium Papers* (2023), Bikker JaccoGribble C., (Ed.), The Eurographics Association, pp. 77–8812 pages. doi:10.2312/hpg.20231138. 9

[WR08] WYMAN C., RAMSEY S.: Interactive volumetric shadows in participating media with single-scattering. In *2008 IEEE Symposium on Interactive Ray Tracing* (2008), IEEE, pp. 87–92. 2

[Wro14] WRONSKI B.: Volumetric fog: Unified compute shader-based solution to atmospheric scattering. In *ACM SIGGRAPH* (2014), vol. 1, p. 3. 2

[WTS*23] WEINRAUCH A., TATZGERN W., STADLBAUER P., CRICKX A., HLADKY J., COOMANS A., WINTER M., MUELLER J. H., STEINBERGER M.: Effect-based multi-viewer caching for cloud-native rendering. *ACM Trans. Graph. 42*, 4 (jul 2023). URL: https://doi.org/10.1145/3592431, doi:10.1145/3592431. 3