Volumetric Radial Shadow Maps

Martin Jakab*
Supervised by: Dr. László Szécsi†

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Műegyetem rkp. 3., H-1111
Budapest / Hungary

Abstract

This paper presents a novel approach to volumetric shadow mapping aimed at reducing the computational redundancy inherent in conventional methods. Traditional volumetric shadow maps evaluate the same depth samples along all primary rays that coincide in the depth map space. Our method pre-processes the shadow map and gathers discrete shadow segments —key intervals that capture the essential shadow-casting characteristics of objects— along possible primary ray images in the depth map. This requires accurately determining the angular and radial ranges from the light source's perspective, ensuring that only the relevant portions of the scene are processed during shadow calculation. A major benefit of this approach is that the computational cost scales with the scene's depth complexity rather than the number of ray casting samples required for a given quality level. Preliminary evaluations indicate that this approach not only streamlines the shadow computation process but also maintains high visual fidelity.

Keywords: Shadow Map, Light Shaft, Real-time Rendering, Participating Media

1 Introduction

In this paper, we introduce the fundamental principles and an initial implementation of radial volumetric shadow maps. This new technique is intended to accelerate shadow computations when the effect of a solid shadow caster on participating media with single scattering is considered. The lighting effect that arises in this situation is known as *light shafts* or *crepuscular rays*, and are staples of real-time rendering. We offer an improvement over the traditional solution of ray marching along *primary rays* originating form the camera, while looking up a *shadow map* rendered from the light source. In particular we reduce the number of samples required in the ray marching step by pre-processing the shadow map.

*martinjakab01@gmail.com
†szecsi@iit.bme.hu

The first step in understanding the pre-processing of the shadow map is a reparametrization using *epipolar geometry*. This gives the name *radial* to our method, as isoparametric points lie along radial (epipolar) lines in the shadow map space. The values stored in the map are also not depth values any more, but angular coordinates. The second step is substituting the shadow caster geometry along these lines with isoparametric segments with shadow casting capability identical to that of the original objects. During ray casting, primary rays can be tested against these segments instead of sampling the original shadow map, leading to significant reduction of required bandwidth.

This approach enables faster processing of shadow map data, as the number of segments in radial shadow maps is, on average, substantially lower than the number of steps required in traditional per-ray shadow computations. However, the method necessitates additional processing steps during shadow map generation.

In this study, we present the algorithmic foundation for constructing and processing radial volumetric shadow maps. Furthermore, we discuss the details of our implementation and explore the potential applications of this technique in real-time rendering and other relevant domains.

2 Related work

Employing ray marching to render volumetric effects is a classic method in graphics, as is using shadow maps to determine light source visibility at the sample points along the camera rays. Full ray marching, however, over all camera rays independently, is extremely expensive compared to other shading effects, as a high number of sample points in every pixel is required. Several approaches have been proposed to exploit coherence, speeding up the process, often with the goal of real-time performance.

The work of Engelhardt and Dachsbacher [1] is an example of an approach where ray marching is not performed for all screen-space pixels, but instead only for cleverly selected ones, and interpolation is used otherwise. This work uses epipolar geometry based on the light source and the camera as optical centers, exactly like our

work presented in our current paper. They locate the ray-marched samples along epipolar lines in camera space. Our work, instead, focuses on the epipolar lines in the shadow map space, and pre-processes the shadow map for faster ray marching. Thus, Engelhardt and Dachsbacher's work reduces the number of rays to be marched, whereas our work speeds up ray marching itself. Although both methods rely on the same epipolar geometry setup, they are orthogonal, and could be used together in theory.

The work of Hanika et al [2] introduces camera-space shadows. The idea is similar to our work: pre-processing the shadow caster geometry to produce an acceleration structure for faster inscattering evaluation along a ray instead of full ray marching. The authors propose to build a quadtree, and thus construct shadow volumes identical in effect to the original shadow casting geometry. Our work produces a data structure with a similar function, but we exploit epipolar geometry to work on two dimensional epipolar planes instead of three dimensional shadow volumes.

Lin et al. [3] achieved a speed-up by reducing sampling in the temporal domain when rendering dynamic scenes. Adaptive Volumetric Shadow Maps [5] and Deep Shadow Maps [4] go further than handling solid shadow casters, and instead capture the volumetric partial shadowing of participating media, layered or semi-transparent geometry. They are powerful and generic methods as opposed to our focus on the specific case of solid shadow casters.

3 Radial Volumetric Shadow Map Algorithm

We assume a pinhole camera and a point-like light source directionally confined to a frustum. These define an epipolar system. Planes that contain both the camera and the light source are the *epipolar planes*. Figure 1 provides an illustration of the concept. The key observation is that all primary rays originating from the camera that are in the same epipolar plane map to the same line in the shadow map. Thus, during ray marching these rays, the same depth samples are read to determine if the sample point is in shadow or not. In fact, rays are composed of shadowed and lit segments. We are going to show (in section 3.4) that rays in the same epipolar plane, when limited to the outside of solid shadow-caster geometry, can be composed of identical potential segments when using epipolar parametrization. Light visibity over these potential segments can be evaluated for any particular primary rays to identify lit parts.

The epipolar planes appear in the shadow map as epipolar lines. These lines start from the epipolar point corresponding to the camera, and proceed in radial directions. We refer to these epipolar lines as *spokes*. The spokes are shown in Figure 2.

An epipolar plane is identified by a *spoke angle*, which gives the direction of the spoke in the shadow map space. A light ray on a spoke is identified by its distance from the camera's epipolar point in the shadow map space. A point on a light ray may be identified by its distance from the light source, but also by picking the camera ray going through it. The camera rays in the epipolar plane are identified by the angle between them and the line connecting the light and the camera. We will refer to these angles as *secondary angles*.

The volumetric shadow mapping algorithm consists of multiple steps, which are detailed below. The final output of the algorithm is an image with single-scattering participating media rendered with shadows cast by solid geometry.

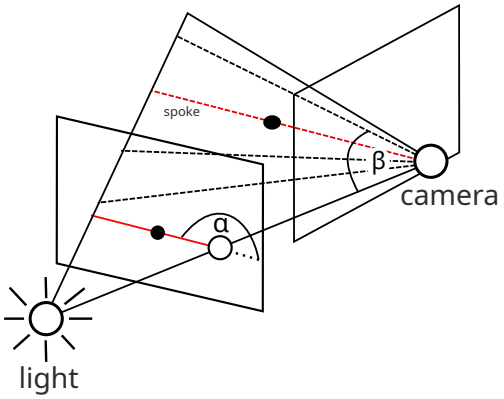

Figure 1: α denotes the spoke angle, β denotes the secondary angle. The black circle represents an object point, and the red line is the ray on which the object point lie.

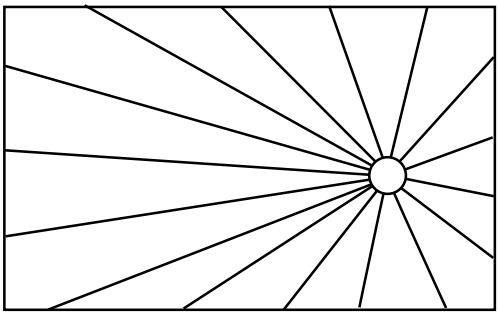

Figure 2: The spokes from the light source's perspective.

3.1 Angular range

We first describe a re-sampling of the shadow map into different texture that can be addressed using the epipolar coordinates. This step would not be strictly necessary, as lookups could be mapped to the original shadow map on-the-fly. However, this explicit re-sampling allows us to demonstrate and visualize our radial shadow map.

The first step in computing the radial shadow map is identifying the angular ranges relevant to shadow calcula-

tion. The camera's view frustum, as seen in the shadow map, may only cover a part of the spoke directions. Restricting the angular domain to regions visible to the camera from the light's perspective allows us to optimize texture usage by excluding unnecessary angles. Since these angles remain fixed for a given light source for each ray, this computation is performed only once per light. To accomplish this, the angles corresponding to the edges of the camera's view frustum must be determined from the light's viewpoint.

After calculating these angles, we need to sort them. Once the values are arranged in increasing order, the algorithm calculates the largest gap between consecutive values in the sorted array, considering the circular nature of the arrangement by connecting the last value to the first.

The largest gap is identified by computing the difference between each value and its successor. The position of the largest gap is tracked as well.

If the largest gap is smaller than π, it means we need to consider every angle on the circle, since the camera's viewport covers all angles in the range $[-\pi, \pi]$.

Otherwise, the boundaries are determined by the two values surrounding the largest gap. This method efficiently identifies the largest empty region in a circular sequence of values and returns the corresponding boundary angles.

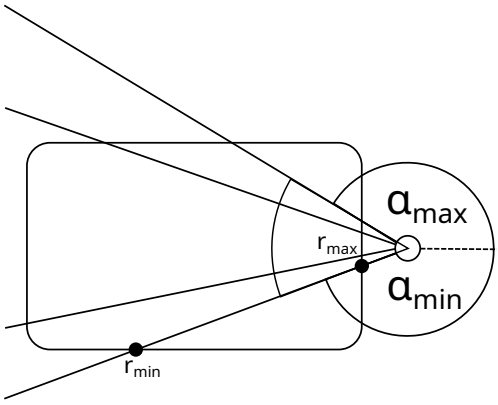

Figure 3: The angular range is defined as $[\alpha_{\min}, \alpha_{\max}]$, and the radial range is defined as $[r_{\min}, r_{\max}]$.

3.2 Radial Ranges

We also calculate the visible range for each spoke from the light source's perspective, i.e. which part of the epipolar line is within the frustum of the lights. Since the spoke is already in the light's clipping space, we only need to check where it intersects the $[-1, 1]$ box. This eliminates the need for complex calculations involving the light's frustum, making the procedure straightforward as shown in Figure 3.

3.3 Radial Shadow Map

The next step of the algorithm requires generating traditional shadow maps from the light's viewpoint, which store depth values in world-space units for simplicity. Two shadow maps are required: one for the front faces and another for the back faces, with the rationale for this explained in section 3.4.

In a radial shadow map, the spoke angles correspond to the y-coordinates, while the x-coordinates represent the distance along the ray in texture space. The specific angles and distances for each pixel are computed, considering both the angular and radial ranges. Once the texture coordinates for a given ray point are determined, they can be used to sample the shadow maps. The resulting depth sample is then utilized to calculate corresponding secondary angles. These secondary angles are subsequently aggregated into shadow segments, as described in section 3.4. To optimize computational efficiency, it is sufficient to store the dot product between the sample direction and the light-camera direction.

Due to the preliminary range calculations, the algorithm efficiently processes only the relevant portions of the shadow maps, reducing unused texture space and thereby enhancing segment resolution.

3.4 Shadow Segment Calculation

Segment encoding can be seen as a kind of run-length encoding (RLE) along shadow map spokes. This method is particularly effective for compressing data with long sequences of identical values. While shadow map depth data is almost never composed of constant-value runs, we aim to replace the values with such runs without compromosing the resulting shadows outside of shadow caster objects. By storing the shadow segments in this manner, we can significantly reduce the memory footprint of the shadow map.

To generate the segments, we process each row of the radial shadow map (i.e. a shadow map spoke) simultaneously using a compute shader. Remember that values in the radial shadow map are secondary angles, which are constant along a primary ray. In this section, for simplicity, we will simply refer to the secondary angle as *depth*, which is valid in the radial shadow map's space. We aim to generate segments of constant depth. In order for the segment to have the shadowing capability as the original solid shadow caster geometry, segment depth has to remain between front face and back face depths.

We scan the values sequentially, while maintaining a possible minimum and maximum depth for the segment. Whenever the minimum and maximum values would cross, a new segment is started. This ensures that the section of the geometry that the segment represents has the same shadow casting properties as the segment, as shown in figure 6.

The algorithm follows these steps:

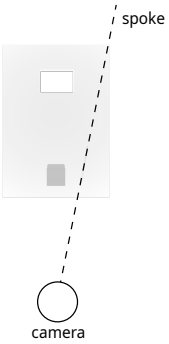

Figure 4: Traditional shadow map

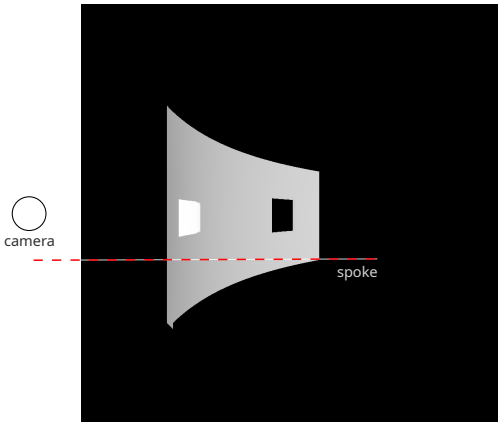

Figure 5: Radial shadow map

1. Initialize Variables:
   - Set up indices and texture dimensions.
   - Define variables for tracking the minimum and maximum, and last secondary angle values for the front face and the back face, and segment properties.
   - Initialize a state variable to track whether the current pixel belongs to a potentially shadowed or completely lit region (where no shadow casters at any depth are present). We call these *object state* and *lit state*, respectively.
2. Iterate Over Each Pixel in the Row:
   - Read shadow depth values (front and back) from the texture.
   - Determine the current state based on depth values:
     - If the depth values are not the background values, but valid front and back face depth, we are in the object state.
     - Otherwise, we are in the lit state.
   - A segment is created when:
     - A shadow-casting segment is found, meaning that the minimum depth would exceed the maximum depth.
     - Transitioning from an object region to a lit region or vice versa.
   - When creating a segment, we store its depth (i.e. secondary angle), and its length in texture space.
   - If the pixel remains in the same state, update minimum and maximum depths accordingly.
3. Finalize the Last Segment:
   - If a segment was still being recorded at the end of the row, store it.

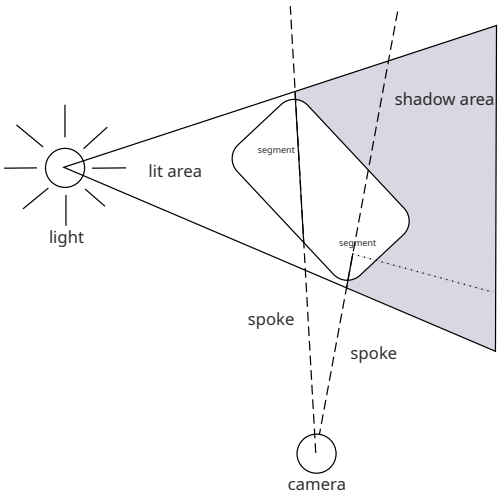

Figure 6: The segments along a spoke. In this example, the shadow casting object is partitioned into two shadow casting segments.

3.5 Shadow segment processing

When computing the light's contribution to the scene, we determine the camera ray direction for each pixel, following the same approach used in traditional volumetric shadow implementation. Once the ray direction is established, deriving the spoke angle and the corresponding secondary angle becomes straightforward. Since the spoke angle falls within the precomputed angular range, we can obtain the interpolation parameter, which, when multiplied by the total number of spoke angles, yields the corresponding spoke angle index.

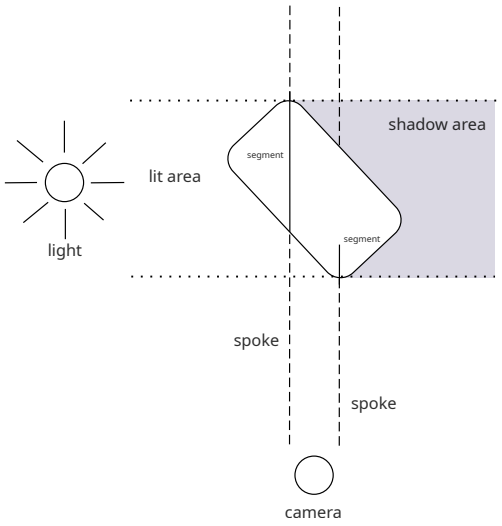

Figure 7: The segments along the spoke in an epipolar viewpoint.

The shadow contribution calculation involves iterating through the shadow segments and evaluating each segment's secondary angle relative to the camera ray's secondary angle. This comparison enables the determination of whether an entire ray segment is occluded or remains unshadowed, as illustrated in Figure 8.

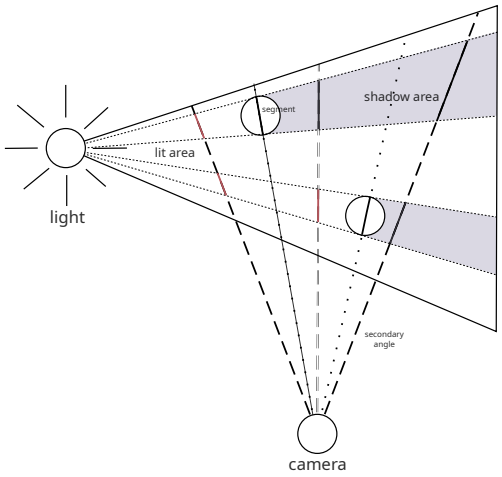

Figure 8: These are three cases of secondary angles for a given spoke. In the leftmost case, both segment comparisons fail. In the center case, one comparison fails while the other succeeds. In the rightmost case, both comparisons are successful. Segment comparisons that failed are highlighted in red, while successful ones are shown in black.

Secondary angles may exist where iterative evaluation results in unintended light contributions. These contributions should be properly constrained by the light source's far clipping plane. Without enforcing this far-plane constraint, light rays will continue to propagate indefinitely, converging toward a singularity due to the absence of proper clipping.

The far clip distance, d_{far}, is computed as follows:

The far clip radius is given by

$$r_{\text{far}} = \frac{w_{\text{eye}} - z_{\text{eye}}}{z_{\text{ray}} - w_{\text{ray}}}$$

where $w_{\text{eye}}, z_{\text{eye}}$ are the homogeneous and depth components of the transformed eye position, and $w_{\text{ray}}, z_{\text{ray}}$ are those of the ray direction.

The far clip point is computed as

$$\mathbf{p}_{\text{far}} = \mathbf{e} + r_{\text{far}}\mathbf{d}$$

where $\mathbf{e}$ is the eye position and $\mathbf{d}$ is the normalized ray direction.

Projecting the far clip point into clip space gives

$$\mathbf{p}_{\text{far,clip}} = P\mathbf{p}_{\text{far}}$$

which is then normalized by its homogeneous component:

$$\mathbf{p}_{\text{far,ndc}} = \frac{\mathbf{p}_{\text{far,clip}}}{w_{\text{far,clip}}}$$

The radial distance of the far clip point in normalized device coordinates (NDC) is

$$r_{\text{far,ndc}} = \|\mathbf{p}^{xy}_{\text{far,ndc}} - \mathbf{p}^{xy}_{\text{eye,ndc}}\|$$

Finally, the far clip distance is determined by

$$d_{\text{far}} = \frac{r_{\text{far,ndc}} - 2r_{\text{min}}}{r_{\text{max}} - r_{\text{min}}}$$

with a conditional adjustment ensuring $d_{\text{far}} = 1$ when $r_{\text{far}} < 0$.

Another constraint for the light contribution is the object's distance along the camera ray, which must be transformed into its radial distance in texture space.

$$\mathbf{e}_{\text{light}} = P_{\text{light}} \cdot \left(\mathbf{e} + \mathbf{d} \cdot d_{\text{depth}}\right)$$

where $\mathbf{e}$ is the eye position, $\mathbf{d}$ is the ray direction, and P_{light} is the projection matrix for the light. The eye space depth is then transformed by the light's projection matrix into light space coordinates.

Next, we compute the light's coordinate depth:

$$\mathbf{r}_{\text{light}} = \frac{\mathbf{e}_{\text{light}}}{w_{\text{light}}} - \mathbf{r}_{\text{eye}}$$

where w_{light} is the homogeneous component of the light's projected depth, and $\mathbf{r}_{\text{eye}}$ is the eye position in NDC space.

The radial distance in texture space is then given by:

$$r_{\text{light}} = \frac{\|\mathbf{r}_{\text{light}}\| - 2r_{\text{min}}}{r_{\text{max}} - r_{\text{min}}}$$

where r_{min} is the minimum radius, and r_{max} is the maximum radius.

Algorithm 1 Shadow Segment Light Contribution

$\mathscr{S} \leftarrow$ segments at angle index θ
$\ell_{\text{curr}} \leftarrow 0$
$\rho_{\text{max}} \leftarrow \min(d_{\text{light}}, d_{\text{far}})$
$\mathbf{L} \leftarrow \mathbf{0}$
for $i = 0$ **to** $n-1$ **do**
  if $\ell_{\text{curr}} \geq \rho_{\text{max}}$ **then**
    break
  end if
  $s \leftarrow \mathscr{S}[i]$
  if $\ell_s \leq 0$ **then**
    break
  end if
  $\ell \leftarrow \ell_s$
  $\ell_{\text{new}} \leftarrow \ell_{\text{curr}} + \ell$
  if $\ell_{\text{new}} > \rho_{\text{max}}$ **then**
    $\ell \leftarrow \rho_{\text{max}} - \ell_{\text{curr}}$
    $\alpha_{\text{obj}} \leftarrow 0$
  end if
  $\ell_{\text{curr}} \leftarrow \ell_{\text{new}}$
  if $\alpha_s < \alpha_{\text{ray}}$ **then**
    $\mathbf{L} \leftarrow \mathbf{L} + \mathbf{C}_{\text{light}} \cdot I_{\text{light}} \cdot \ell \cdot r_{\text{diff}} \cdot 2$
    if $\ell_{\text{new}} > \rho_{\text{max}}$ **then**
      break
    end if
    continue
  end if
end for
return $\mathbf{L}$

Finally, we ensure the calculated depth is constrained to the appropriate value.

Once the constraint values have been established, the minimum value among them will determine the final distance. The implementation is detailed in Algorithm 1.

This algorithm efficiently accumulates light contributions by traversing precomputed shadow segments, using angle-based occlusion tests and respecting the spatial constraints established by the far clip distance and object depth.

4 Results

For the implementation we used a custom renderer built using the wgpu Rust library. The following results were obtained on a machine with an Intel i7-12650H CPU and an NVIDIA RTX 3050Ti Mobile GPU.

We compare our solution with the traditional volumetric shadowing method in a simple scenario. The presented scene consists of a cube and a rectangular cuboid with a hole.

In the traditional approach, we make a slight optimization compared to a completely naive implementation: we compute the intersection segment of the ray with the light source's frustum and perform calculations only within this segment.

To ensure comparable results between the two methods, we use the same step resolution, so the number of steps taken along each ray is the same. The results are shown in Table 1.

Our tests indicate that even at relatively low shadow map resolutions, the radial method can be advantageous. This is due to the significantly lower number of segments required compared to the number of samples needed in the traditional approach.

An example result is shown in Figure 9. The scene is rendered with a resolution of 1920×1080 pixels, and the shadow map resolution is 1024×1024 pixels. The maximum number of segments is set to 32.

In certain cases, as illustrated in Figure 10, the number of segments can increase significantly, potentially exceeding the predefined maximum threshold. This may lead to artifacts, as the entire radial domain is not fully covered. An example of this is shown in Figure 10.

Near the camera, the spokes are typically densely packed because the rays converge toward the camera. As a result, shadow quality tends to degrade along the direction of the rays.

5 Conclusions

In our work, we explored a novel approach to generating volumetric shadows, achieving significant performance improvements in several test cases. However, in certain

Step resolution	Traditional	Radial
128	12.5 ms	2.0 ms
256	21.7 ms	2.7 ms
512	27.6 ms	3.8 ms
1024	40.5 ms	5.1 ms

Table 1: Comparison of the two methods with different step resolutions (1920×1080 render resolution)

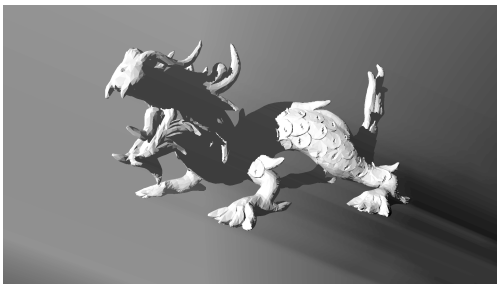

Figure 9: An example result of the Stanford Dragon model with volumetric shadows.

scenarios, our method proved less efficient than traditional techniques.

We implemented a solution that represents an object's shadow-casting capability using segmented intervals, effectively eliminating redundant computations along the shadow rays.

Our method can be further enhanced by investigating the following directions:

Reducing the number of segments can not only lower computational costs but also decrease the likelihood of exceeding the segment buffer's capacity. This can be achieved by allowing segments to be stored with two secondary angles instead of one, enabling the merging of sections that remain entirely within the object's projected shadow volume. During processing, the true secondary angles can be derived through interpolation between segment endpoints.

The per-row processing of the radial shadow map texture can be parallelized using parallel reduction techniques, potentially improving performance.

In many cases, rendering passes can be consolidated to minimize unnecessary texture accesses. Moreover, alternative, non-traditional rendering techniques could allow for direct rendering of the radial shadow map texture, further improving efficiency.

6 Acknowledgements

This work was supported by OTKA K-145970 and by the Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program.

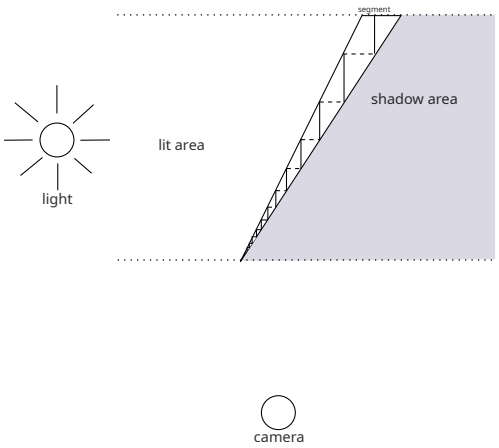

Figure 10: An example case where too many segments need to be inserted.

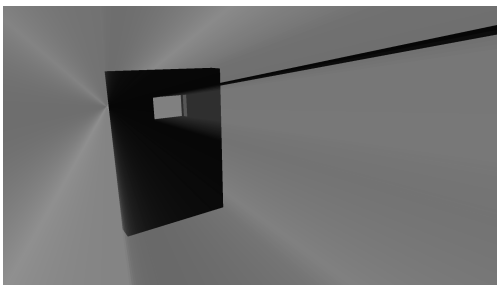

Figure 11: An example scenario in which an artifact is generated due to the insertion of an excessive number of segments.

References

[1] Thomas Engelhardt and Carsten Dachsbacher. Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 119–125, 2010.

[2] Johannes Hanika, Peter Hillman, Martin Hill, and Luca Fascione. Camera space volumetric shadows. In *Proceedings of the Digital Production Symposium*, pages 7–14, 2012.

[3] Hsiang-Yu Lin, Chin-Chen Chang, Yu-Ting Tsai, Der-Lor Way, and Zen-Chung Shih. Adaptive sampling approach for volumetric shadows in dynamic scenes. *IET Image Processing*, 7(8):762–767, 2013.

[4] Tom Lokovic and Eric Veach. Deep shadow maps. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 311–318. 2023.

[5] Marco Salvi, Kiril Vidimče, Andrew Lauritzen, Aaron Lefohn, and Matt Pharr. Adaptive volumetric shadow maps. *GPU Pro 360 Guide to Shadows*, pages 97–113, 2018.