Epipolar Sampling for Shadows and Crepuscular Rays in Participating Media with Single Scattering

Thomas Engelhardt Carsten Dachsbacher [*]
VISUS / University of Stuttgart

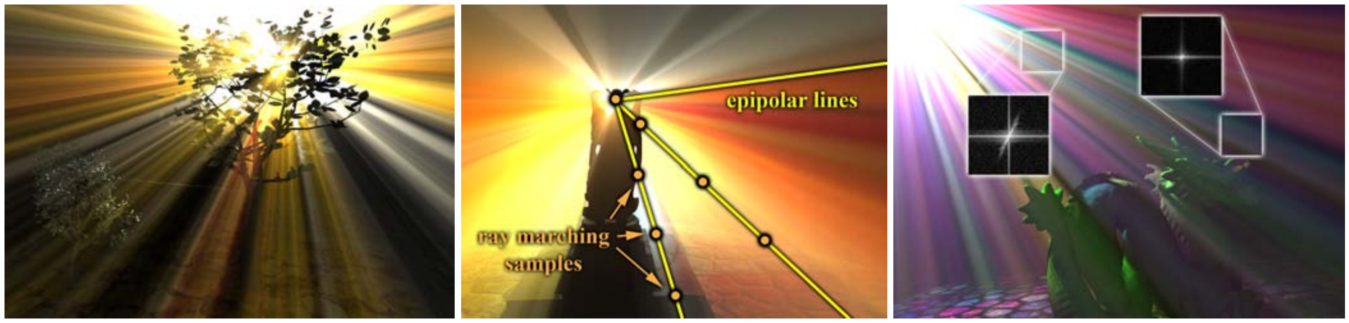


Figure 1: *Volumetric shadows rendered in real-time with our method. We reduce the number of ray marching samples for computing the inscattered light by placing them along epipolar lines and depth discontinuities (center). Our sampling strategy is motivated by the observation that scattering varies mostly smoothly along these lines, and is confirmed by the frequency analysis (right).*

Abstract

Scattering in participating media, such as fog or haze, generates volumetric lighting effects known as crepuscular or god rays. Rendering such effects greatly enhances the realism in virtual scenes, but is inherently costly as scattering events occur at every point in space and thus it requires costly integration of the light scattered towards the observer. This is typically done using ray marching which is too expensive for every pixel on the screen for interactive applications. We propose a rendering technique for textured light sources in single-scattering media, that draws from the concept of epipolar geometry to place samples in image space: the inscattered light varies orthogonally to crepuscular rays, but mostly smoothly along these rays. These are epipolar lines of a plane of light rays that projects onto one line on the image plane. Our method samples sparsely along epipolar lines and interpolates between samples where adequate, but preserves high frequency details that are caused by shadowing of light rays. We show that our method is very simple to implement on the GPU, yields high quality images, and achieves high frame rates.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shading

Keywords: participating media, global illumination, GPU

1 Introduction

Global illumination effects, such as indirect lighting, soft shadows, caustics, or scattering greatly enhance the realism of rendered scenes, but also have a high computational demand that is challenging for interactive applications. In particular, effects that involve participating media are intricate, sometimes even for offline rendering systems. The presence of participating media implicates that light does not only interact with the surfaces of the scene, but requires the evaluation of the scattering integral at each point in space. Simplifying assumptions, such as the restriction to single-scattered light only, are often made to enable higher frame rates. Fortunately, these effects suffice to produce convincing results in many cases.

We follow this direction and focus on single-scattering effects from textured light sources. Note that our sampling strategy is beneficial for non-textured light sources as well. In both cases, the computation requires an integration of the inscattered light along a ray emanating from the camera that is typically realized using ray marching. Consequently, the cost scales with the number of samples along the ray and the number of rays. Previous work limited ray marching to lit regions of the scene (shadowed regions obviously do not contribute inscattered light), but used naïve sub-sampling in image space to reduce the number of rays.

In this paper we show how to determine the locations on the image plane where the evaluation of the scattering integral is important, and where the inscattered light can be interpolated faithfully. Our idea originates from the observation that the inscattered light varies mostly smoothly along the crepuscular or god rays that are visible in participating media (Fig. 1). This observation is also confirmed by the 2-dimensional Fourier analysis of the god rays shown in the teaser insets: close to the light source the image of inscattered light contains high frequency orthogonal to the god rays, and with the distance from the light source these high frequencies diminish. This fact can be explained using epipolar geometry and is exploited by our sampling strategy. We demonstrate a simple yet efficient implementation on GPUs and show high-quality renderings at real-time frame rates.

2 Previous Work

Participating media in offline rendering is a well-studied topic. It is typically encountered using ray marching to accumulate the radiance along a ray, together with Monte Carlo techniques to model the scattering of particles [Pharr and Humphreys 2004], or by volumetric photon mapping [Jensen and Christensen 1998; Jarosz et al.

[*]e-mail: {engelhardt, dachsbacher}@visus.uni-stuttgart.de

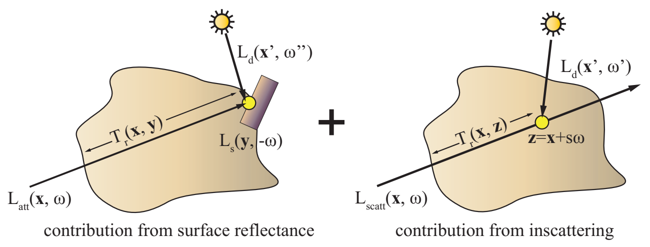


Figure 2: *Left: light reflected at a surface point* $\mathbf{y}$ *towards* $\mathbf{x}$ *is attenuated in participating media. Right: in single-scattering media the attenuated direct light is scattered only once, here at a point* $\mathbf{z}$ *into the path towards* $\mathbf{x}$.

2008]. A recent survey by Cerezo et al. [2005] provides a comprehensive overview of research in this field.

Many approaches in the domain of real-time rendering are based on shadow volumes to provide information on the regions of illuminated and shadowed space in front of visible surfaces. To identify these regions the shadow volume polygons must be sorted back-to-front which introduces additional sorting cost [Biri et al. 2006; Mech 2001; James 2003]. Gautron et al. [2009] compute light cones (instead of shadow volumes) and compute the intersection of an eye ray and a cone to determine the length of the eye path through the light.

Wyman and Ramsey [2008] render inscattering from textured and shadowed spot lights using ray marching, and use shadow volumes to distinguish between lit and potentially shadowed parts of the scene. They also use a naïve image space sub-sampling to reduce the number of rays. Tóth and Umenhoffer [2009] propose to use interleaved sampling in screen space exploiting the fact that nearby pixels cover similar geometry; the inscattered light is computed using ray marching and shadow mapping.

Slicing techniques, known from volume rendering, can be used to emulate ray marching through participating media. Dobashi et al. [2002] and Mitchell [2004] use slicing to render volumetric shadows by rendering quads parallel to the image plane at varying distances from the eye. These slices are accumulated back-to-front using blending, while the shadows on each slice are computed using shadow mapping. Imagire et al. [2007] reduce the number of slices by averaging the illumination over regions near each slice.

Mitchell [2007] presents a simple post-processing technique working in screen space that is very fast, but has inherent limitations including non-textured lights and false light shafts from objects behind the light sources. Similar restrictions apply to the technique presented by Sousa [2008] that uses a radial blur on the scene depth buffer to obtain a blending mask for sun shafts.

Our work relies on straightforward ray marching, but selects the locations in image space for which we ray march. Thus, it could be combined with "per ray" acceleration techniques [Wyman and Ramsey 2008]. However, we omitted this in our implementation as shadow volumes are often not practical for scenes with complex geometry and rarely used. Nevertheless, our method significantly reduces the samples and is capable of producing high frame rates with good quality. In our implementation we used isotropic scattering, and the scattering models of Hoffman and Preetham [2003], and Sun et al. [2005].

Concepts of epipolar geometry have been used in computer graphics in other contexts before: McMillan's image warping algorithm [1997] renders new views from recorded images. Popescu et al. [2006] leverage epipolar constraints to implement ray-geometry intersections for reflection rendering. Moreover, screen space ambient occlusion algorithms [Bavoil et al. 2008] and and relief-mapping methods [Policarpo et al. 2005; Policarpo and Oliveira 2006] benefit from acceleration techniques motivated by epipolar geometry.

3 The Volumetric Rendering Equation

In this section we briefly review the concepts of light transport in the presence of participating media. The radiance transfer equation describes how light travels through a medium. The radiance at a point $\mathbf{x}$ incident from direction ω due to inscattered light is:

$$L_{scatt}(\mathbf{x},\omega) = \int_0^\infty T_r(\mathbf{x},\mathbf{z})L_i(\mathbf{z},-\omega)ds, \text{ with } \mathbf{z}=\mathbf{x}+s\omega. \tag{1}$$

The transmittance $T_r(\mathbf{x},\mathbf{z})$ accounts for both absorption and outscattering and defines the fraction of light that is not attenuated when light travels along the path from $\mathbf{z}$ to $\mathbf{x}$. The extinction coefficient $\sigma_t(\mathbf{x}) = \sigma_a(\mathbf{x}) + \sigma_s(\mathbf{x})$ combines the absorption $\sigma_a(\mathbf{x})$ and scattering $\sigma_s(\mathbf{x})$ coefficients:

$$T_r(\mathbf{x},\mathbf{z}) = \exp\left(\int_0^{\|\mathbf{x}-\mathbf{z}\|} \sigma_t(\mathbf{x}+s'\omega)ds'\right). \tag{2}$$

The radiance at a point $\mathbf{z}$ in direction ω consists of the volume emission L_{ve} and the inscattered radiance (second summand):

$$L_i(\mathbf{z},\omega) = L_{ve}(\mathbf{z},\omega) + \sigma_s(\mathbf{z})\int_{4\pi} p(\mathbf{z},\omega,\omega')L(\mathbf{z},\omega')d\omega'. \tag{3}$$

The *phase function* p is a probability distribution describing light scattering from direction ω' into direction ω [Pharr and Humphreys 2004]. In the simplest case of isotropic scattering $p(\mathbf{z},\omega',\omega) = 1/(4\pi)$. The radiance L_s leaving a surface point $\mathbf{y}$ in direction $-\omega$ is attenuated in the medium and the total incident radiance at $\mathbf{x}$ is (Fig. 2):

$$L(\mathbf{x},\omega) = \underbrace{T_r(\mathbf{x},\mathbf{y})L_s(\mathbf{y},-\omega)}_{\substack{L_{att}(\mathbf{x},\omega)\\ \text{attenuated surface radiance}}} + \underbrace{\int_0^\infty T_r(\mathbf{x},\mathbf{z})L_i(\mathbf{z},\omega)ds}_{\substack{L_{scatt}(\mathbf{x},\omega)\\ \text{inscattered radiance reaching } \mathbf{x}}}. \tag{4}$$

Even for offline rendering systems, the scattering equation is very costly to evaluate when multiple scattering is considered, i.e. if $L(.)$ in Eq. 3 accounts for previously scattered light as well. Fortunately, convincing volumetric effects can be achieved under simplified assumptions. When restricting to *single scattering*, only direct light, i.e. radiance emitted by light sources (and attenuated in the medium), is scattered into an eye path. Throughout this paper we will also ignore volume emission and restrict ourselves to homogeneous media, i.e. the scattering and absorption coefficients are constant and the transmittance can be computed as:

$$T_r(\mathbf{x},\mathbf{z}) = \exp\left(-\sigma_t\,\|\mathbf{x}-\mathbf{z}\|\right). \tag{5}$$

Under these assumptions we compute the integral using ray marching, i.e. by stepping along the ray $\mathbf{x}+s\omega$. The direct lighting L_d (replacing L_i in Eq. 4) is evaluated using regular shadow mapping, taking the absorption from the light source position to, and the emitted radiance towards $\mathbf{z}$ into account.

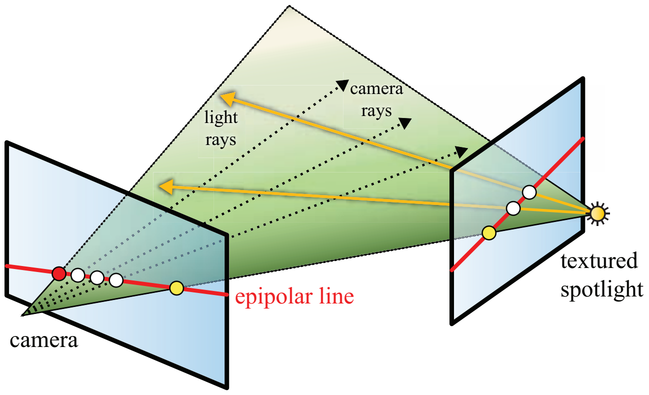


Figure 3: *We render crepuscular rays from textured spot lights. Epipolar geometry explains why we can sample sparsely along epipolar lines (when detecting discontinuities properly).*

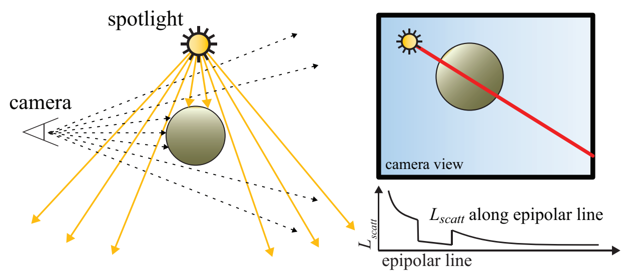


Figure 4: *Left: shadowed regions do not contribute to inscattering. Right: As a consequence, the inscattered light L_{scatt} reaching the camera exhibits abrupt changes (plot).*

4 Epipolar Sampling

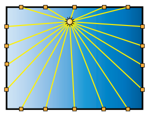
In this section we describe our sampling strategy that determines where the inscattering term L_{scatt} is computed, and where it is interpolated from nearby sample locations. The attenuated surface radiance is computed for every pixel.

Our approach is inspired by the appearance of crepuscular rays that emanate radially from a (textured) light source. The reason for this can be easily explained with epipolar geometry (see e.g. [Hartley and Zisserman 2004]). If we look at a set of rays from the camera going through pixels that lie on one of these crepuscular rays, which are epipolar lines on the screen (Fig. 3, left), then we observe that these rays map to another epipolar line on the “image plane” of the spot light (Fig. 3, right). That is when computing the inscattered light, these camera rays all intersect the same light rays. The only difference is how far these intersections are away from the light source, and thus how much light was attenuated on its way to the intersection. The key point is that the attenuation varies smoothly for camera rays along an epipolar line, while camera rays on another epipolar line intersect different light rays and thus potentially produce different inscattering. In the presence of occluders, however, the inscattered light along an epipolar line can also exhibit discontinuities. This is because of two reasons (see Fig. 4): the distance to the first visible surface along a camera ray changes abruptly from one ray to another, and shadowed regions in space do not contribute to the ray integral.

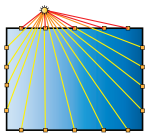
Taking advantage of these observations our method for placing the ray marching samples requires three steps:

- Selection of a user defined number of epipolar lines, and refinement of an initial equidistant sampling along these lines to capture depth discontinuities.
- Ray marching all samples (in parallel on the GPU).
- Interpolation along and between the epipolar line segments.

4.1 Generating Epipolar Lines



The key of our method is the sample placement along epipolar lines, and thus we have to determine these first. Let us first assume that the light source is in front of the camera and we project it onto the image plane. We create the epipolar lines for sampling by connecting the projected position with a user defined number of points equidistantly placed along the border of the screen (figure on the right).



In the case that the light source maps to a point outside the screen some of these lines are completely outside the visible area and are culled prior to the subsequent steps. Epipolar lines that map to outside of the light frustum are omitted for further computation as well. For the remaining lines we compute the location where they enter and exit the visible part of the image plane.

If the light source is behind the camera its projected position is the vanishing point where the (anti-)crepuscular rays converge in infinity. As our sampling does not distinguish whether rays emanate from, or converge to a point, we can proceed exactly as in the aforementioned case. If the camera and the light source coincide then the crepuscular rays and the camera rays are parallel. In this barely happening case the epipolar lines degenerate to points, and our sampling scheme reduces to a naïve screen space sub-sampling. We create a low number of initial samples along the epipolar lines, typically 8 to 32, which is enough the capture the variation of the inscattered light in absence of occluders. This divides the rays into epipolar line segments (Fig. 6, right).

4.2 Sampling Refinement

After the generation of the epipolar lines and the initial samples, we need to detect depth discontinuities along the line segments. The inscattered radiance might change abruptly at discontinuities, and thus we need to place an additional sample directly before and after these locations. For this we use the depth buffer of the scene and linearly search along each epipolar line segment for adjacent locations that exhibit a large threshold. Note that the image needs to be rendered for the $L_{att}(\mathbf{x}, \omega)$ part anyway, and we simply keep the depth buffer for detecting discontinuities.

The resulting sampling then matches the observations that we made from the appearance of the crepuscular rays: we have more samples close to the light source, where the angular variation of the inscattered radiance is stronger, a predefined minimum sampling to capture the attenuation of the inscattered light, and additional samples to account for discontinuities.

On first sight it seems natural to take the difference of inscattered light at the end points of a epipolar line segment as a refinement criterion. However, this has two inherent problems: first, this criterion might miss volumetric shadows if the line segment spans a shadowed region (for example in situations like the one depicted in Fig. 4). Second, this criterion requires the computation of the inscattered light to be interleaved with the sample refinement. Our experiments showed that this is hard to parallelize efficiently on a GPU. We alternatively used a 1-dimensional min-max depth mipmap along epipolar lines to detect discontinuities (similar to [Nichols and Wyman 2009], but in 1D). This obviously produces the same results, but as we need to detect each discontinuity exactly once, the construction of the mip-map did not amortize, presumably as it requires additional render passes.

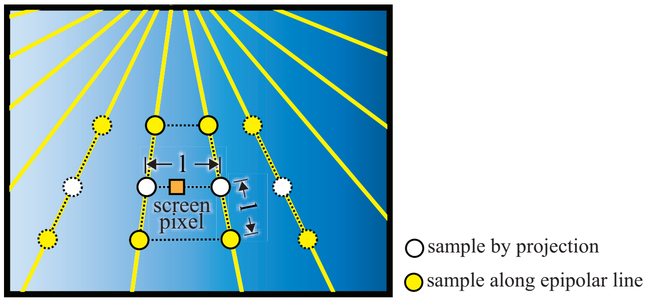


Figure 5: *We interpolate the inscattered radiance for a pixel on the screen using a bilateral filter sampling the closest two epipolar lines. If these taps are invalid, we move to the next outer epipolar lines (sample taps shown with dashed lines).*

4.3 Ray Marching and Upsampling

After the sample placement, we compute the inscattered radiance for all samples on the epipolar lines. From these samples we obtain a piece-wise linear approximation of L_{scatt} along each epipolar line. From this sparse sampling of the image plane, we need to reconstruct the inscattering for each pixel of the final image by interpolation. We found that a bilaterial filter, drawing the filter taps directly from the epipolar lines, is well-suited to interpolate the ray marching samples while preserving edges at the same time.

In order to compute the inscattered radiance for a pixel on the screen we first determine the two closest epipolar lines. We then interpolate bilaterally along, followed by another bilateral interpolation between the epipolar lines. First, we project the pixel's location onto the two closest lines (white dots in Fig. 5) and denote the distance between the two projected points as l. Along each of these two epipolar lines we use a 3 to 5 tap 1D-bilateral filter. The filter taps are taken at distances with a multiple of l along the epipolar lines towards and away from the light source, where L_{scatt} is interpolated linearly from the ray marching samples (that have been placed along the epipolar lines). The filter weight for the i-th tap is determined by its distance (l_i, a multiple of d) and depth difference d_i to the pixel. We compute the weight by a multiplication of two radial basis functions taking the l_i and d_i as input, respectively. We tried weighting similar to Shepard's scattered data interpolation as $w_S(x) = ax^{-2}$, using a Gaussian $w_G(x) = e^{-ax^2}$ (a is a user-defined, scene-dependent constant), and simple thresholding (for d_i only). We found that all three yield comparable results once the appropriate parameter is adjusted.

The interpolation along each epipolar line yields an interpolated color and we also keep track of the sum of weights, denoted as w_{left} and w_{right}. Next we interpolate bilaterally between the two colors, according to the weights. However, it may occur that no suitable sample taps are found along an epipolar line (i.e. all taps have weights close to 0), in particular in situations where object discontinuities are aligned with the epipolar lines. In these cases w_{left} or w_{right} are obviously (close to) zero as well, and we take the respective next outer epipolar line to obtain valid taps for the screen pixel.

This rather simple bilateral filter yields visually pleasing results in most cases. Of course, any such interpolation can cause artifacts due to missing information in some geometric configurations. We also experimented with a bilateral filter that bases its decision on a distinction of taps whose depth is close to the pixel's depth, significantly larger or smaller (similar to Wyman and Ramsey's [2008] implementation). We found that this yields comparable results, however, at increased computation cost. Note that this approach might be more suitable to favor one sort of artifacts over others, e.g. by preferring distant taps to close ones and thus preventing dark pixels.

5 Implementation

In this section we detail the individual steps of the implementation of epipolar sampling. Our method has been designed with modern graphics hardware in mind and implemented using Direct3D10. The rendering of the final image comprises 3 steps: rendering the scene with direct lighting into an off-screen buffer, determining the inscattered light for the entire image using epipolar sampling and interpolation, and additively combining both contributions while accounting for attenuation. The epipolar sampling and the computation of the inscattered light consists of the following steps:

- Determine the epipolar lines and place the initial samples.
- Detect all discontinuities along the epipolar lines, and determine for which samples in image space the inscattered light has to be computed using ray marching, and where interpolation is sufficient.
- Compute the inscattering with ray marching, and interpolate inbetween along the epipolar lines. Finally interpolate between epipolar lines to obtain L_{scatt} for the entire image.

5.1 Epipolar Line and Initial Sample Generation

For efficient computation and interpolation, we store the inscattered light along the epipolar lines in a 2D texture: each texture row corresponds to one epipolar line, and each column corresponds to one location along that line (Figure 6). The number of columns in this texture determines the maximum number of ray marching samples along that line. However, we do not necessarily ray march for every sample as the inscattered radiance might also be interpolated for a texel (typically it is interpolated for most of them).

The first step is to relate the texels in this 2D layout to coordinates on the viewport and store this information in the *coordinate texture*. For this we setup the epipolar lines in a pixel shader as described in Sect. 4.1: each texture row corresponds to the epipolar line from the projected light source position to one location on the screen's border (Fig. 6). Next, these lines are clipped against the light frustum and the viewport to obtain the actually contributing epipolar lines and the respective visible segments thereof. After clipping we know the entry and exit locations of each line, and compute the location of every texel by interpolation between the segment's endpoints. Recall from Sect. 4.1 that line segments that do not overlap the viewport are excluded from subsequent computation. We use the stencil buffer to mask these epipolar lines and rely on the GPU to automatically cull operations in later render passes.

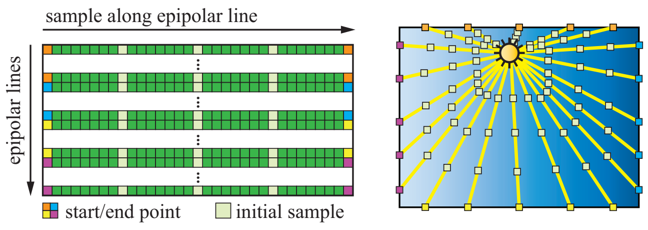


Figure 6: *Each epipolar line is mapped to one row in a 2D texture. Each pixel represents one sample along one of these lines.*

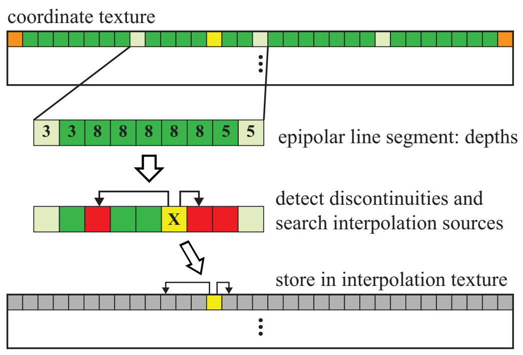

coordinate texture
3 3 8 8 8 8 8 5 5
epipolar line segment: depths
detect discontinuities and
search interpolation sources
X
store in interpolation texture

Figure 7: *For each pixel in the interpolation texture we search depth discontinuities along the epipolar line segment to which it belongs. Samples that require ray marching are marked, and for all other samples we store from which other samples we interpolate the inscattered radiance into the interpolation texture.*

5.2 Discontinuity Detection and Sampling Refinement

After the construction of the epipolar lines, we determine for which samples the inscattered light, L_{scatt}, has to be computed using ray marching (*ray marching samples*), and for which an interpolation is sufficient (*interpolation samples*). For interpolation samples, we also need to determine from which other two samples the inscattered light is interpolated. This information is stored in another 2D texture (*interpolation texture*) with the same layout as the one described above.

Our implementation carries out the classification of texels, and the determination of interpolation sources in a single pixel shader. For each texel in the interpolation texture we regard the entire line segment to which it belongs (line segments are determined by the initial sample placement). For every sample in the segment we copy the depth values from the scene's depth buffer at the corresponding locations (obtained from the coordinate texture) into a local array.

Next we determine from which samples in that segment we interpolate the inscattered light for the currently considered texel. The case that this texel becomes a ray marching sample is detected in the same procedure (see below). The following pseudo-code illustrates the search for the interpolation samples (indexed by `left` and `right`) in the local depth array `depth1d` of size `N`, for the current texel at location `x` (Fig. 7):

```
left = right = x;
while ( left > 0 ) {
  if(abs( depth1d[left-1], depth1d[left] ) > threshold)
    break;
  left --;
}
while ( right < N-1 ) {
  if(abs( depth1d[right], depth1d[right+1] ) > threshold)
    break;
  right ++;
}
```

Note that if no discontinuities are found then the detected interpolation samples are the endpoints of the initial line segment. If the epipolar line crosses a discontinuity at `x`, then the corresponding sample will detect itself as interpolation sample. Samples that interpolate from themselves obviously are the ray marching samples. The number of texture accesses required for the discontinuity search for every epipolar line is: the maximum number of samples per epipolar line times the samples per initial line segment. Note that for typical settings this has negligible cost on contemporary GPUs.

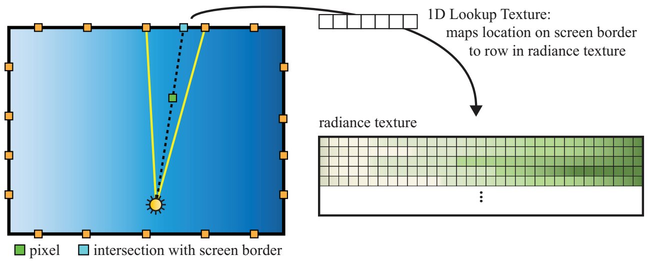

1D Lookup Texture:
maps location on screen border
to row in radiance texture
radiance texture
pixel
intersection with screen border

Figure 8: *For interpolation of the inscattered radiance for a pixel we determine the closest epipolar line using a 1D lookup texture.*

5.3 Ray Marching, Interpolation and Upsampling

The next step is to determine the inscattered radiance for every sample along the epipolar lines. These values will be stored in a third texture (again the same extent as the aforementioned textures) that we name *radiance texture*. After the discontinuity detection some texels have been classified for interpolation, while others are marked for ray marching. For the latter we need to execute the costly ray marching operations, but we do not want the GPU to spend time on the other samples. We experimented with rendering point primitives to trigger ray marching operations for individual samples, but this approach showed bad performance. Again the stencil buffer proved to be the best choice and we mark the texels corresponding to ray marching samples first. That followed we render a full screen quad over the entire radiance texture (which is bound as render target) using a pixel shader that performs the ray marching. The early-z optimization of GPUs takes care that this shader is only executed where required. Next, a simple shader performs the interpolation: for every texel it looks up both interpolation sources, and interpolates according to its relative position. Note that we can use the (already existent) stencil buffer to distinguish between the sample types, but in the discontinuity detection all ray marching samples detected themselves as interpolation source anyway. We always ray march at a fixed step size restricted to the light frustum (i.e. we do not ray march in definitely unlit parts of the scene).

At this point we generated all information that is necessary to determine the inscattered radiance for all pixels in the final image. This procedure is illustrated in Fig. 8. First we compute the ray from the projected light source position to the pixel, and intersect this ray with the border of the screen. Next we determine the epipolar line that is closest to this intersection. To facilitate this computation, we use a small 1D texture that stores the closest epipolar line for every pixel on the screen border. Note that this texture depends on the number of epipolar lines only, and is precomputed and stored once.

Finally, we project the pixel's location onto the two closest epipolar lines to obtain the taps for the bilateral filter. The filtering to obtain the interpolated inscattered radiance is implemented exactly as described in Sect. 4.3: after interpolating bilaterally along each of those epipolar lines, we interpolate inbetween. If no suitable taps, i.e. no taps with large weights, are found on one of the two closest epipolar lines we move to the next outer line. In our experiments this procedure was sufficient to find an appropriate radiance sample except for very rare cases.

6 Results and Conclusions

We implemented our method in Direct3D10 and measured all results using an ATI Radeon HD4850 on a 2.4GHz Core2Quad processor; all timings are measured for rendering at 1920×1080 reso-

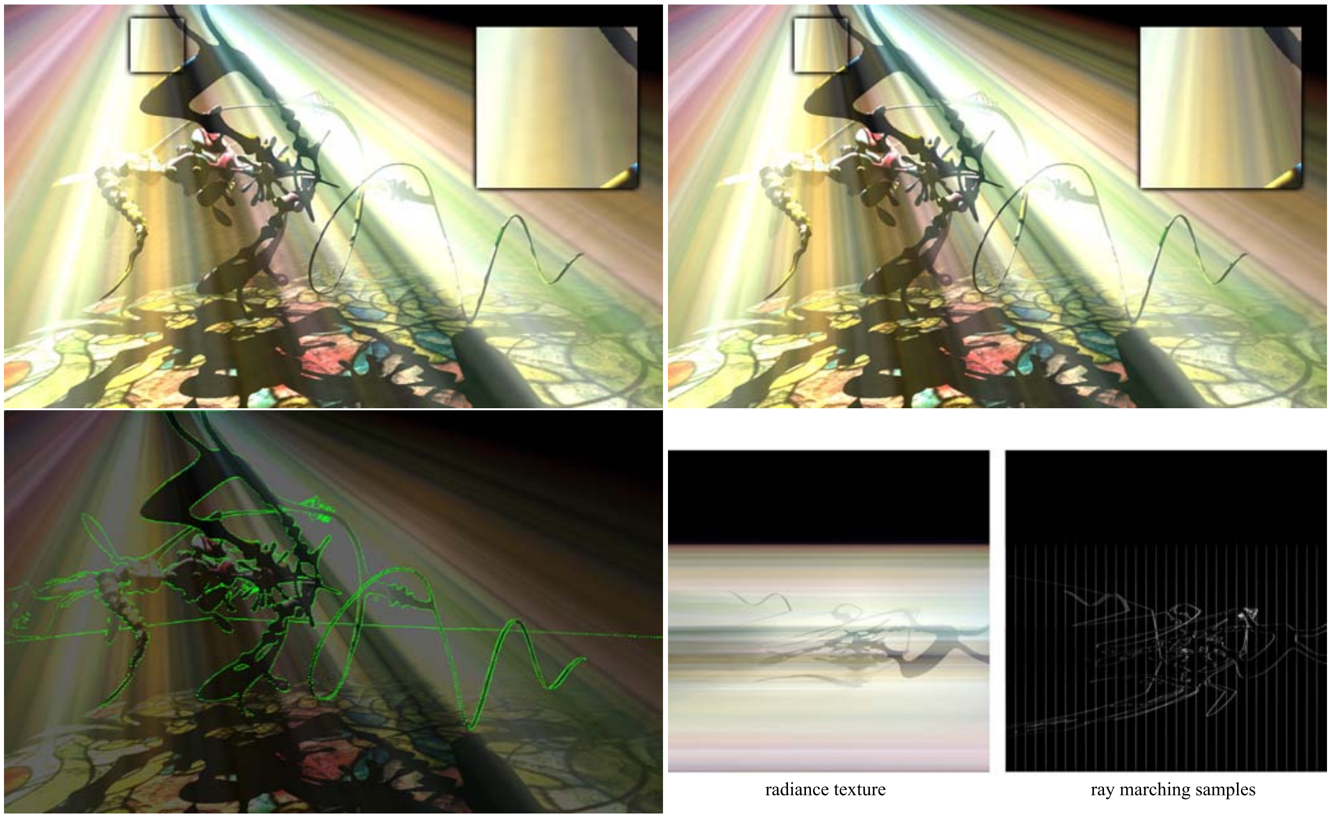


Figure 9: *Top left: naïve subsampling and ray marching (150 steps) with* 256×256 *samples at 40 fps. Top right: epipolar sampling with 1024 lines and 8 initial samples per line yields much better and crisper results at comparable speed (46fps) using only about 17.000 ray marching samples. Bottom: the locations where discontinuities are detected (green), and the radiance texture.*

lution using the scattering model by Hoffman and Preetham [2003]. The major cost in the rendering is the ray marching which indicates that the epipolar sampling (including line generation, depth discontinuity search, and interpolation) introduces a small overhead only. The detailed timings for the left image in the teaser are: epipolar line generation 1.2ms, discontinuity search 6.0ms, ray marching 16.7ms (500 steps per ray), interpolation 4.4ms, for 1024 epipolar lines and 32 initial ray marching samples (recall that the initial segment size influences the cost for searching discontinuities); less then 120.000 ray marching samples have been generated, i.e. ray marching was performed for only 5.8% of the pixels in the image. To our knowledge there is no other method reducing the ray marching samples in a non-naïve manner. To this end, we measure the performance gain over brute-force rendering (ray marching for every pixel) and naïve subsampling, and it naturally increases with the number of steps for the ray marching, as we significantly reduce the number of ray marching samples itself.

Fig. 9 shows renderings of the "yeah right" model comparing epipolar sampling to naïve subsampling. With the settings used for the top-right image in Fig. 9, our method yields results indistinguishable from per-pixel ray marching, while our method ray marches only about 17k rays. Fig. 10 demonstrates the influence of the number of epipolar lines; expectedly, fewer lines yield blurrier images.

Note that the images rendered with epipolar sampling show a crisper appearance which is due to the sample placement that matches the frequency content of the image (see teaser, right). However, our method tends to oversample along crepuscular rays when the light source is close to the screen border and some epipolar lines are short (noticeable in Fig. 6, right). This is because we place a fixed number of initial ray marching samples along each line. Although this can be easily adjusted, we opted for an as simple as possible implementation. It would also be possible to further optimize the discontinuity search by using larger depth thresholds for distant parts of the scene (similar optimizations have been proposed by Wyman and Ramsey [2008]). When using our method in dynamic scenes, e.g. under light and camera movement, we observed stable and flicker-free rendering. When using a lower number of epipolar lines, we have to prefilter the light texture accordingly.

In conclusion, epipolar sampling naturally places ray marching samples where they are required and significantly reduces the cost for rendering single-scattering participating media effects with (textured) light sources in real-time. The implementation is simple and can be easily integrated into existing renderers, and combined with other per-ray optimizations. An extension of our method to support multiple light sources would be to place ray marching samples preferably at the intersection of the epipolar lines of both light sources.

References

BAVOIL, L., SAINZ, M., AND DIMITROV, R. 2008. Image-space horizon-based ambient occlusion. In *SIGGRAPH '08: ACM SIGGRAPH 2008 talks*, ACM, New York, NY, USA, 1–1.

BIRI, V., ARQUÈS, D., AND MICHELIN, S. 2006. Real time rendering of atmospheric scattering and volumetric shadows. *Journal of WSCG'06 14(1)*, 65–72.

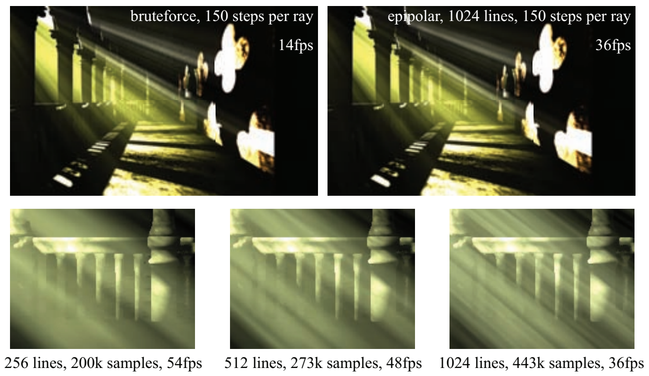

bruteforce, 150 steps per ray
14fps
epipolar, 1024 lines, 150 steps per ray
36fps
256 lines, 200k samples, 54fps
512 lines, 273k samples, 48fps
1024 lines, 443k samples, 36fps

Figure 10: *Top left: brute force rendering with 150 ray marching steps per pixe. 150 steps are still too few in this case (please zoom in to see the artifacts). Top right: our method yields smoother results with the same number of steps, due to the interpolation along the epipolar lines. Bottom line: close ups with increased contrast of renderings with 256, 512, and 1024 epipolar lines, and the respective number of ray marching events.*

CEREZO, E., PEREZ-CAZORLA, F., PUEYO, X., SERON, F., AND SILLION, F. 2005. A survey on participating media rendering techniques. *The Visual Computer*.

DOBASHI, Y., YAMAMOTO, T., AND NISHITA, T. 2002. Interactive rendering of atmospheric scattering effects using graphics hardware. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, 99–107.

GAUTRON, P., MARVIE, J.-E., AND FRANÇOIS, G., 2009. Volumetric shadow mapping. ACM SIGGRAPH 2009 Sketches.

HARTLEY, R. I., AND ZISSERMAN, A. 2004. *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, ISBN: 0521540518.

HOFFMAN, N., AND PREETHAM, A. J. 2003. Real-time light-atmosphere interactions for outdoor scenes. 337–352.

IMAGIRE, T., JOHAN, H., TAMURA, N., AND NISHITA, T. 2007. Anti-aliased and real-time rendering of scenes with light scattering effects. *The Visual Computer 23*, 9, 935–944.

JAMES, R. 2003. True volumetric shadows. In *Graphics programming methods*, Charles River Media, Inc., 353–366.

JAROSZ, W., ZWICKER, M., AND JENSEN, H. W. 2008. The beam radiance estimate for volumetric photon mapping. *Computer Graphics Forum (Proc. Eurographics EG'08) 27*, 2 (4), 557–566.

JENSEN, H. W., AND CHRISTENSEN, P. H. 1998. Efficient simulation of light transport in scences with participating media using photon maps. In *SIGGRAPH '98*, ACM, 311–320.

MCMILLAN, JR., L. 1997. *An image-based approach to three-dimensional computer graphics*. PhD thesis, Chapel Hill, NC, USA.

MECH, R. 2001. Hardware-accelerated real-time rendering of gaseous phenomena. In *Journal of Graphics Tools, 6(3)*, 1–16.

MITCHELL, J., 2004. Light shafts: Rendering shadows in participating media. Game Developers Conference 2004 Presentations.

MITCHELL, K. 2007. Volumetric light scattering as a post-process. *GPU Gems 3*, 275–285.

NICHOLS, G., AND WYMAN, C. 2009. Multiresolution splatting for indirect illumination. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, 83–90.

PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann.

POLICARPO, F., AND OLIVEIRA, M. M. 2006. Relief mapping of non-height-field surface details. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 55–62.

POLICARPO, F., OLIVEIRA, M. M., AND COMBA, JO A. L. D. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 155–162.

POPESCU, V., MEI, C., DAUBLE, J., AND SACKS, E. 2006. Reflected-scene impostors for realistic reflections at interactive rates. *Comput. Graph. Forum 25*, 3, 313–322.

SOUSA, T., 2008. Crysis next gen effects. Game Developers Conference 2008 Presentations.

SUN, B., RAMAMOORTHI, R., NARASIMHAN, S. G., AND NAYAR, S. K. 2005. A practical analytic single scattering model for real time rendering. *ACM Trans. Graph. 24*, 3, 1040–1049.

TÓTH, B., AND UMENHOFFER, T. 2009. Real-time volumetric lighting in participating media. In *EUROGRAPHICS 2009 Short Papers*.

WYMAN, C., AND RAMSEY, S. 2008. Interactive volumetric shadows in participating media with single scattering. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, 87–92.