Filter-based Real-time Single Scattering using Rectified Shadow Maps

Oliver Klehm
MPI Informatik

Hans-Peter Seidel
MPI Informatik

Elmar Eisemann
Delft University of Technology

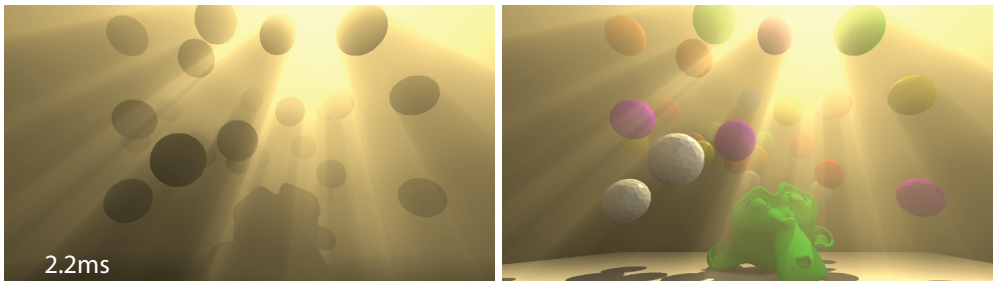


Figure 1. Our method needs only 2.2ms for single scattering using a 1024^2 shadow map and a 1920×1080 full-HD screen resolution. It has basically a constant cost per screen pixel, achieving speedups of a magnitude for similar quality compared to state-of-the-art methods whose performance is scene-dependent [Chen et al. 2011].

Abstract

Light scattering due to participating media is a complex process and difficult to simulate in real time because light can be scattered towards the camera from everywhere in space. In this work, we replace the usually-employed ray-marching with an efficient ray-independent texture filtering process, which leads to a significant acceleration. Our algorithm assumes single scattering media and takes a rectified shadow map as input, which can also be efficiently rectified by a new scheme, which we propose in this work. Our method is fast and almost resolution independent, while producing near-reference results. For this reason, it is a good candidate for performance-critical applications, such as games.

1. Introduction

Participating media influence the appearance of a scene drastically. A careful simulation can add realism, as well as spatial cues regarding scene layout. When light interacts with thin participating media, crepuscular rays (or so-called god rays) appear, which are a strongly visible phenomenon and they often serve even artistic purposes.

 ISSN 2331-7418

In contrast to surface-light interaction, which is evaluated at surface points, participating media require solving for the in-scattering of light along entire view rays. Recently, several real-time methods have been proposed to approximate light interaction in homogeneous participating media via a single scattering model for fully-dynamic scenes. Single scattering limits the light-media interaction to one event, which for a single point light means that the light is redirected along the view ray. This methodology is similar to direct illumination for surfaces. Although this simplifying assumption led to several methods that are faster than an accurate multiple-scattering model, these are usually scene-dependent and struggle when employed in real time applications. Usually, they would launch a ray-marching per screen pixel along the view ray. For each location along the ray, a lookup in a shadow map is performed to determine if light can be in-scattered at this location. To accelerate the algorithm, a rectified shadow map can be used, in which view rays always stay within a single row of the shadow map. Using a tree structure [Baran et al. 2010] or a GPU-friendly min-max hierarchy [Chen et al. 2011], longer marching distances can be used employing only a single shadow test. Nonetheless, for complex scenes, these structures can become relatively inefficient, making many steps along the view ray necessary.

We [2014] previously showed how to compute single scattering with lower complexity than previous methods for fully-dynamic scenes, practically causing a speed-up of up to a magnitude. The key insight of our solution is to replace the ray marching with a filtering process, in the spirit of Annen et al.'s work [2007]. This *prefiltering*, independent of actual view rays of the camera, is applied to a rectified shadow map and results in a simple computation to evaluate the scattering contribution for each screen pixel. The benefit of this is that we compute visibility for all screen pixels that lie on so-called epipolar line at once. Thus, the algorithm is mostly scene independent, as first only a simple shadow map needs to be produced and filtered. Second, the evaluation per pixel is a single dot product, which is also independent of the scene complexity and cheap. In consequence, even high-resolution renderings can be achieved in milliseconds. Additionally, we presented a rectification method for the shadow map, which can be produced with a linear matrix transform, hereby, avoiding pixel-shader scattering operations as used in previous reprojection schemes [Chen et al. 2011]. Further, our method is mostly independent of scene-complexity, resolution, and viewpoint, using a constant amount of operations per pixel. Its quality is high (Fig. 1), but the method delivers a speed up of over a magnitude compared to competing solutions [Chen et al. 2011]. In this paper, we present this algorithm, as well as additional implementation details, evaluations, pseudo code, and new results compared to our previous work [2014].

This paper contributes:

- A GPU-friendly single scattering method using prefiltering;
- A single-step shadow-map rectification using a matrix multiplication;
- GPU-oriented optimizations.

In the following, we first discuss the background of scattering (Sec. 2), before presenting the core of our scattering method (Sec. 3), extensions to it (Sec. 3.2), and give implementation details (Sec. 3.3). As our method relies on a rectified shadow map, we present the existing method and our new rectification (Sec. 4). Finally, we present our results (Sec. 5) and conclude the paper (Sec. 6).

2. Background

In this section, we explain scattering model and the rectified shadow map. This background knowledge makes it easier to follow the core of our algorithm. In particular, a good understanding of the properties of rectified shadow maps is helpful. We will introduce our own rectification method later in this article.

2.1. Scattering Model

We are concerned with volumetric single scattering, for which all contributing scattering events for a given pixel happen along its corresponding view ray from the camera. We further assume a homogeneous medium, which allows us to compute medium transmittance and scattering probability analytically. In turn, radiance towards a camera at $\mathbf{x}$ from direction ω_i due to a single directional light source with direction ω_l is computed by integrating the contributions along the *view ray*:

$$L_\mathrm{scat}(\mathbf{x},\omega_\mathrm{i}) = \rho\,\sigma_\mathrm{t}\,f(\omega_\mathrm{l},\omega_\mathrm{i})\,\widetilde{L}_\mathrm{i}(\omega_\mathrm{l}) \int_0^s T(\mathbf{x},\mathbf{x}_t)\,V(\mathbf{x}_t)\,dt \tag{1}$$

with $\mathbf{x}_t = \mathbf{x} - t\,\omega_\mathrm{i}$. The integration is limited by s, the distance to the first visible surface, the closest surface along a view ray. Further, throughout the paper, we will consider a single light source. As light transport is linear, one can compute a solution per light source and accumulate the results.

A homogeneous medium has a number of spatially-invariant properties; scattering albedo ρ, denoting the relative scattering capability, extinction coefficient σ_t, and phase function f. $T(\mathbf{x},\mathbf{x}_t) = e^{-t\sigma_\mathrm{t}}$ denotes medium transmittance for a ray segment of length t and describes the out-scattering behavior of the medium. $\widetilde{L}_\mathrm{i}(\omega_\mathrm{l})$ is the unblocked, incoming light and $V(\mathbf{x}_t)$ the corresponding light visibility at the scattering point (being one if the light is not occluded as seen from $\mathbf{x}_t$ and zero otherwise). Throughout the paper we will use the shorthand of $C := \rho\,\sigma_\mathrm{t}\,f(\omega_\mathrm{l},\omega_\mathrm{i})\,\widetilde{L}_\mathrm{i}(\omega_\mathrm{l})$ to denote all ray-constant factors. A common approximation of Eq. 1 is based on factoring out

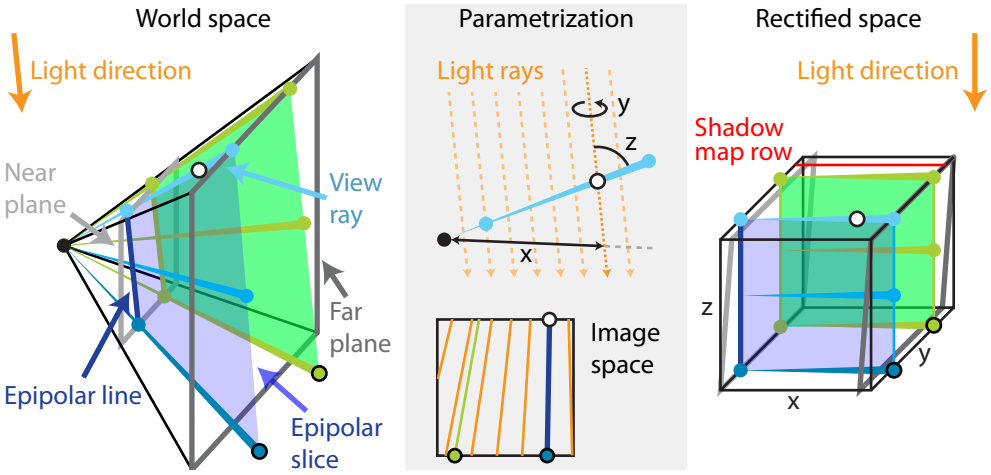


Figure 2. A rectification transforms camera view rays in such a way that they become parallel to each other; View rays in the same epipolar slice share the same y coordinate and the light direction is aligned with the z axis.

the visibility term [Baran et al. 2010; Chen et al. 2011]:

$$L_{\text{scat}}(\mathbf{x},\omega_{\text{i}}) \approx C\overline{V}(\mathbf{x},\omega_{\text{i}}) \int_0^s T(\mathbf{x},\mathbf{x}_t)\, dt,$$
$$\overline{V}(\mathbf{x},\omega_{\text{i}}) = s^{-1} \int_0^s V(\mathbf{x}_t)\, dt. \tag{2}$$

The first integral can be computed analytically:

$$\int_0^s T(\mathbf{x},\mathbf{x}_t)\, dt = \frac{1}{\sigma_{\text{t}}}(1 - e^{-s\sigma_{\text{t}}}). \tag{3}$$

Thus, the challenge lies in the computation of the average visibility $\overline{V}$ for a given view ray. In Sec. 3.2, we will show how to lift this approximation and move the transmittance weighting back into the visibility integral.

2.2. Rectified Shadow Map and Epipolar Geometry

In a rectified shadow map, view rays share the same y,z-coordinates and are in turn parallel to the x-axis, as well as to each other (Fig. 2). Consequently, the view rays can be parameterized by two parameters; y to indicate the shadow map row and its depth z. View rays go from left to right, hence, it is possible to map all view rays formed by the camera model to a range of $[0;1]$ for y and z. In contrast, light rays are indexed by (x,y) similar to usual shadow mapping. With epipolar coordinates, z denotes the angle spanned by the view-ray direction and the light direction. All view rays with the same y-coordinate lie in a so-called epipolar slice. In turn, an epipolar slice projects on an epipolar line in the camera view. The advantage of such a rectification is that one can accelerate the ray marching along view rays, e.g., by

a 1D min-max mipmap [Chen et al. 2011]. Our approach will also make use of the constant z of each view ray. Details on the rectified shadow map construction via reprojection and our novel projective transformation will be presented in Sec. 4.

3. Method

Our main observation is that solving average visibility in Eq. 2 is similar to filtering hard shadows in surface shadow mapping. In fact, percentage-closer filtering (PCF) performs visibility tests for a single surface point against multiple nearby shadow-map texels in a 2D kernel (*one-to-many*). In our case, for single scattering with a standard shadow map, the filter kernel becomes a line, corresponding to the projection of the view ray into the shadow map. However, one major difference is that the depth along the view ray changes. Hence, different sample points are tested against their corresponding shadow-map texels (*many-to-many*). This issue can be resolved by relying on a rectified shadow map [Chen et al. 2011] (Sec. 4). The constant z-coordinate along a view ray ensures that all sample points will have equal depth. Hence, we are back to testing multiple shadow map entries to a single depth value (view ray) — only within a 1D kernel, derived by the projected view ray.

This conceptional similarity to PCF does not yet result in any performance advantage, as we still need to evaluate each shadow-map texel, which is identical to ray marching. Our goal is to enable a filter process on the shadow map, which allows us to query the response efficiently. We take inspiration from acceleration schemes for PCF. Annen et al. [2007] approximate the visibility function by a linear combination of basis functions, which enables *prefiltering*, i.e., the shadow map is filtered *before* visibility is evaluated for a specific point. The idea is to decompose the shadow-test functions (each corresponding to a shadow-map texel) into a linear combination of basis functions. This allows to filter the basis functions independent of the coefficients. In consequence, it is also possible to remove all ray-related terms from the single scattering integral. The performance advantage stems from the fact that filtering a single row in the shadow map results in the filtered response for all rays in the corresponding epipolar slice. Because walking along a ray means traversing the shadow map from left to right, the filtering is a (weighted) prefix summation. The result of the scattering operation for a ray is then obtained by first deriving a few easy-to-compute ray coefficients depending on the ray's depth and then performing a dot product with the prefiltered basis functions. This leads to a four step algorithm: computing the rectified shadow map, deriving basis functions, performing prefiltering in form of a prefix sum, and evaluating per pixel the in-scattering along the ray by computing its coefficients.

In the following section, we detail the core of our method: the definition of the basis functions, the prefiltering, and the final ray evaluation. We first focus on the

simpler case of computing an average visibility along a view ray, before giving a more accurate approximation of Eq.1 in Sec. 3.2.

3.1. Average Visibility

To compute the average visibility for a view ray, we need to determine:

$$\overline{V}(r) = s^{-1} \int_0^s V(\mathbf{x}_t)\, dt, \tag{4}$$

which allows us to solve Eq. 2. We assume a rectified shadow map as input, hence, the view ray $r := (\mathbf{x}, \omega_i)$, transformed into the rectified shadow map, is parallel to the x-axis and starts at $x = 0$. Therefore, all points on the view ray have equal y (epipolar slice) and z (depth). In the following, we will denote z_r as the depth of the view ray in the rectified shadow map. Due to the discrete nature of a shadow map, the binary function V is piecewise constant along the view ray with equally sized steps and its integral can therefore be represented as a sum of functions $\mathcal{V}_{z_i}(z_\mathbf{x})$, which describe the visibility function as for shadow mapping:

$$\overline{V}(r) \approx s^{-1} \sum_{i=1}^{s} \mathcal{V}_{z_i}(z_{\mathbf{x}_i}) \tag{5}$$

with $\mathbf{x}_i$ describing a sample point on the view ray and s denoting the texel index of the first visible surface $\mathbf{x}_s$, hit by ray r. $\mathcal{V}$ is a step function with $\mathcal{V}_{z_i}(z_\mathbf{x}) = 1$, if $z_\mathbf{x} - z_i \leq 0$, else $V_{z_i}(z_\mathbf{x}) = 0$. Note that Eq. 5 still corresponds to the usual ray-marching procedure with a visibility tests for each sample.

Next, we apply the linearization method of Annen et al. [2007], which allows us to remove the ray-dependent $z_{\mathbf{x}_i}$ from the sum. This enables us to perform a 1D prefiltering along the x-axis on the rectified shadow map, independent of the view rays within an epipolar slice. In detail, we represent each visibility function $\mathcal{V}_{z_i}$ by a linear combination of M basis functions $B_k(z_i)$, such that $\mathcal{V}_{z_i}(z_{\mathbf{x}_i}) \approx \sum_k^M a_{k,i}(z_{\mathbf{x}_i}) B_k(z_i)$. $B_k(z_i)$ can be stored as *basis textures*; a 2D texture array indexed by k. The values are computed directly from the shadow map and each pixel's depth serves as an input to derive the values at the corresponding location in these basis textures.

Key of the previous step is that the coefficients $a_{k,i}$ depend only on the sample points $\mathbf{x}_i$ of r. Further, due to the rectification, the z-values are constant along the ray; $z_r = z_{\mathbf{x}_i} = z_{\mathbf{x}_j} \forall i, j$ and, consequently, the coefficients $a_{1,i} \ldots a_{M,i}$ are independent of i, and we simply write $a_k(z_r)$:

$$\overline{V}(r) \approx s^{-1} \sum_{i=1}^{s} \left(\sum_{k}^{M} a_k(z_r) B_k(z_i) \right). \tag{6}$$

n consequence, we can swap the order of the sums and remove $a_k(z_r)$ from the sum over the texels:

$$\overline{V}(r) \approx s^{-1} \sum_{k}^{M} a_k(z_r) \left(\sum_{i=1}^{s} B_k(z_i) \right). \tag{7}$$

Now, the inner sum only depends on the depth values stored in the shadow map, which makes it possible to filter the basis functions $B_k(z_i)$ independently of the ray. However, while Annen et al. could directly apply a fixed filter kernel to the basis textures, our situation is different. The kernel size is not constant, but requires a ray-dependent upper limit s, which determines an upper limit for the summation over the shadow-map texels. To compute s, we need to project the first surface hit along the ray into the shadow map, indicating where the ray stops.

Adapting the Filter Kernel The solution to the varying kernel size is to compute and store the 1D prefix sum for all possible view ray distances up to s, which is the number of columns of the rectified shadow map. Consequently, the resulting *filtered-basis textures* share the resolution of the rectified shadow map and each row still corresponds to an epipolar slice. For each texel k we then have access to the sum of the first k basis functions to its left. Specifically, for each k the sum of each basis function is stored in the channels and layers of the filtered-basis textures, resulting in a *filtered-basis vector* for each 2D texel. To obtain the sum of the functions between two locations u, v on the ray, it is sufficient to retrieve the *filtered-basis vector* for these two locations and to compute a difference between the two.

Evaluating a Ray Given the filtered-basis textures, we can evaluate the scattering along a ray r via a dot product in a constant-time evaluation. The first vector corresponds to the coefficient vector $\vec{a}(z_r) := a(a_1(z_r), \ldots, a_M(z_r))^T$ that we compute from r. For the second, we transform the first hit point $\mathbf{x}_s$ along r into the rectified shadow map, and retrieve the filtered-basis vector from the corresponding texel in the filtered-basis textures. Optionally, we can also retrieve the filtered-basis vector for the ray's origin on the near plane projected into the rectified shadow map and subtract this contribution to get in-scattering only within the near and far plane. Nonetheless, in practice, as this correction only affects the part of the ray from the camera to the near plane, it can usually be safely neglected.

Fourier Series as Basis Functions We follow Annen et al. and use the Fourier series of $\mathcal{V}$. Choosing the Fourier transformation implies that the function to be transformed needs to be periodic. To satisfy this requirement, we observe that $\mathcal{V}_{z_i}(z_{\mathbf{x}_i}) = H(z_{\mathbf{x}_i} - z_i)$, where H is the Heavyside step function ($H(x) = 1$ for $x < 0$, else $H(x) = 0$). As z_i and z_r only take values in the range of $[0;1]$, H will only be used with values in $[-1;1]$. Hence, Annen et al. suggest to use $H(\pi x) - 0.5$, which is a stretched and shifted version, leading to the full use the interval $[-\pi, \pi]$ and point symmetry. The latter is helpful when approximating via its Fourier transformation (see [Annen et al.

2007] for details), resulting in relatively simple coefficients:

$$\begin{aligned} a_{(2k-1)}(z_r) &= 2c_k^{-1}\cos(c_k z_r), & a_{(2k)}(z_r) &= -2c_k^{-1}\sin(c_k z_r) \\ B_{(2k-1)}(z_i) &= \sin(c_k z_i), & B_{(2k)}(z_i) &= \cos(c_k z_i) \end{aligned} \tag{8}$$

with $c_k = \pi(2k-1)$. We thus obtain:

$$\begin{aligned} \mathcal{V}_{z_i}(z_r) &= (H(\pi(z_r - z_i)) - 0.5) + 0.5 \\ &\approx \sum_{k=0}^{M} a_k(z_r) B_k(z_k) + 0.5 \qquad (9) \\ &= (a_1(z_r), \ldots, a_M(z_r))(B_1(z_i), \ldots, B_M(z_i))^T + 0.5, \\ &= \vec{a}(z_r)^T (B_1(z_i), \ldots, B_M(z_i))^T + 0.5, \qquad (10) \end{aligned}$$

which implies

$$\sum_{i=1}^{s} \mathcal{V}_{z_i}(z_r) = \vec{a}(z_r)^T \left(\sum_{i=1}^{s} B_1(z_i), \ldots, \sum_{i=0}^{s} B_M(z_i)\right)^T + 0.5.$$

Alternative basis functions to approximate $\mathcal{V}$ are discussed by Annen et al. [2007] and comparisons to different linearization methods that are based on the distributions of $z_0, \ldots, z_s$ by Klehm et al. [2014].

3.2. Extensions

The method described in Sec. 3.1 allows us to compute an average visibility, but it omits weighting by transmittance (cf. Eq. 2). Correctly applied, the weights have a exponential distribution just as transmittance along a ray. Hence, average visibility mostly works well for optically-thin media, where transmittance has low impact with a flat exponential distribution. For thicker media, we show how an approximate weighting can be employed to increase accuracy. Further, we explain the use of textured and spotlight sources.

Transmittance Weighting Along the View Rays In Eq. 1 visibility V is weighted by the transmittance term $T(\mathbf{x}, \mathbf{x}_t) = e^{-t\sigma_t}$, which can be evaluated analytically for a piecewise constant V based on a shadow map. Starting with the discrete space of the shadow map, we denote as w_i as weighting factor for the i-th segment along the view ray. Then, we obtain:

$$\begin{aligned} L_{\text{scat}}(\mathbf{x}, \omega_i) &\approx C \sum_{i=1}^{s} \left(\int_{(i-1)\Delta}^{i\Delta} e^{-t\sigma_t}\, dt \right) \mathcal{V}_{z_i}(z_{\mathbf{x}_i}) \\ &\approx C \frac{1}{\sigma_t} \sum_{i=1}^{s} w_i \mathcal{V}_{z_i}(z_{\mathbf{x}_i}), \\ w_i &:= \sigma_t \int_{(i-1)\Delta}^{i\Delta} e^{-t\sigma_t}\, dt = e^{-(i-1)\Delta\sigma_t} - e^{-i\Delta\sigma_t} \end{aligned} \tag{11}$$

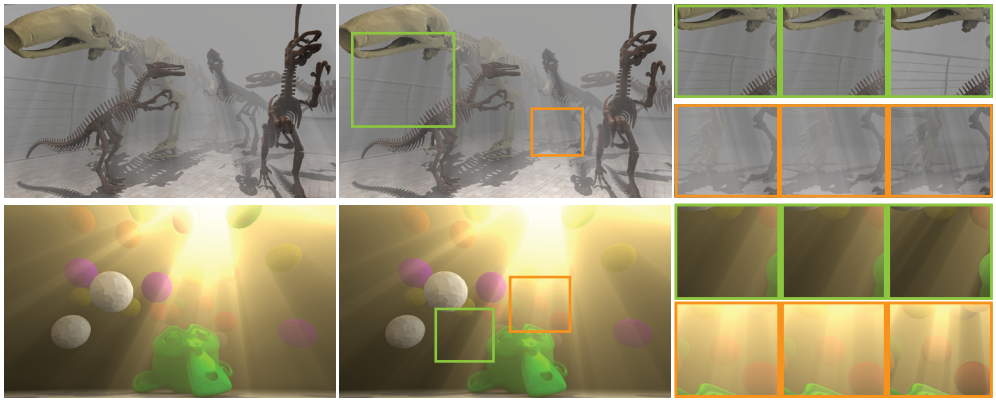
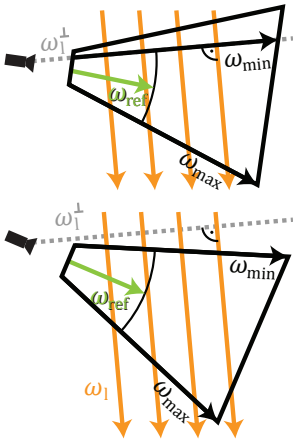


Figure 3. Left: our method; center: ray-marched in-scattering; The insets show from left to right: our method applying transmittance weights; ray-marching reference with correct attenuation; average visibility. Average visibility gives plausible results, but many significant differences occur. Shadows are missing or are overly dark. The difference increases with the thickness of the participating media.

Along the ray r, each texel in the rectified shadow map implies an associated distance Δ in world-space. However, the magnitude of Δ is different for view rays in the same epipolar slice, as it is defined by the angle between the light direction and view direction. Therefore, we cannot turn Eq. 11 into a ray-independent, weighted filtering.

As a compromise, we opt for finding a good constant for all rays inside the same epipolar slice. To this extent, we pick a reference ray in the epipolar slice whose world step size corresponding to a shadow-map texel Δ_{ref} will be used as a representative for all rays when computing w_i.



There are several ways such a ray could be defined depending on the optimization strategy one chooses. The approach that we use throughout the paper is as follows. We start by relating the different Δ's to each other. The minimal distance $\Delta^\perp$ to cross a shadow map texel is given by the ray that is orthogonal to the light direction $\omega_l^\perp$. The Δ of another ray in the same epipolar slice with direction ω_i can be expressed in dependence of its angle to $\omega_l^\perp$: $\Delta = \frac{1}{k(\omega_i)}\Delta^\perp$ with $k(\omega_i) := \omega_i \cdot \omega_l^\perp$ being the cosine of the angle between both. Thus, the relationship between any two rays with directions ω_i and ω_i' is given by $c = \dfrac{\omega_i \cdot \omega_l^\perp}{\omega_i' \cdot \omega_l^\perp} = \dfrac{k(\omega_i)}{k(\omega_i')}$. Therefore, we seek a reference ray, for which this ratio is close to one for all rays in the epipolar slice. The first task is to find the rays with directions $\omega_{\text{min}}, \omega_{\text{max}}$ that have $\Delta^{\text{min}}, \Delta^{\text{max}}$ (the minimum and maximum Delta values) within the epipolar slice inside the camera frustum. If $\omega_l^\perp$ is part of the frustum, it

directly gives $\omega_i^{\min}$ as it has the minimal distance $\Delta^{\perp}$ to cross a shadow map texel. If this is not the case one of the limiting rays must be $\omega_{\min}$, basically, the angularly closest direction to $\omega_l^{\perp}$ within the camera frustum. $\omega_{\max}$ has a similar relationship; if any of the view rays is parallel to the light direction ω_l, we get $\omega_{\max} = \omega_l$, because it forms the largest angle of 90 degrees with $\omega_l^{\perp}$; otherwise, $\omega_{\max}$ is found as one of the limiting rays, angularly farthest away from $\omega_l^{\perp}$. Any reference view ray should now lie between $\omega_{\min}$ and $\omega_{\max}$. A simple and robust approach is to use the bisectrix between $\omega_{\min}, \omega_{\max}$. This, however, is not necessarily the optimal choice. To get an optimal ω_{ref}, we seek to minimize the maximal errors that occur for $\omega_{\min}$ and $\omega_{\max}$:

$$c_{\min} = \frac{\omega_{\text{ref}} \cdot \omega_l^{\perp}}{\omega_{\min} \cdot \omega_l^{\perp}} = \frac{k(\omega_{\text{ref}})}{k(\omega_{\min})} \leq 1, \qquad c_{\max} = \frac{\omega_{\text{ref}} \cdot \omega_l^{\perp}}{\omega_{\max} \cdot \omega_l^{\perp}} = \frac{k(\omega_{\text{ref}})}{k(\omega_{\max})} \geq 1.$$

We use the shorthand $k_{\text{ref}} := k(\omega_{\text{ref}})$ in the following. As we care about the relative scale of $c_{\min}$ and $c_{\max}$, we need to invert one to compare them. Now, $c_{\min}^{-1}$ and $c_{\max}$ have a range of $[1;\infty)$ and k_{ref} can be optimized to minimize both. Writing both as functions of k_{ref}: $c_{\min}(k_{\text{ref}}), c_{\max}(k_{\text{ref}})$, we can formalize the optimization as:

$$\min_{k_{\text{ref}}} E(k_{\text{ref}}); \;\; E(k_{\text{ref}}) := \max(c_{\min}(k_{\text{ref}}), c_{\max}(k_{\text{ref}})).$$

Due to the inversion $c_{\min}^{-1}$, increasing k_{ref} monotonically decreases $c_{\min}^{-1}(k_{\text{ref}})$, but also monotonically increases $c_{\max}(k_{\text{ref}})$ and vice versa. Hence, the intersection of both functions $c_{\min}^{-1}(k_{\text{ref}}) = c_{\max}(k_{\text{ref}})$ yields the minimum of $E(k_{\text{ref}})$ at point:

$$k_{\text{ref}} = \sqrt{k(\omega_{\min})\, k(\omega_{\max})} = \sqrt{(\omega_{\min} \cdot \omega_l^{\perp})(\omega_{\max} \cdot \omega_l^{\perp})}.$$

We optimized for $k_{\text{ref}} = \omega_{\text{ref}} \cdot \omega_l^{\perp}$, which is the cosine of the angle of the optimal reference ray to the orthogonal light direction and is used to derive ω_{ref}. Note that $k(\omega_i)$ approaches zero for ω_l. Hence, we cannot compute the optimum, iff $\omega_{\max} = \omega_l$ as in this case $c_{\max} = \infty$. Due to this practical limitation, we generally use the bisectrix as the reference ray, which already delivers a very good approximation (Fig. 3).

It is noteworthy that previous work completely factors out visibility or uses low-rank approximations [Baran et al. 2010; Chen et al. 2011]. Our w_i weights are a cheap, but good approximation with a positive impact on the visual quality (Fig. 3). As our weighting has practically no impact on performance, we generally recommend using this weighted variant over average visibility.

Local Lights A challenging task is to integrate the distance falloff along the light rays, which, in contrast to directional sources, is visually important for local light sources. Factoring visibility and accounting for the distance falloff in an analytical solution for unoccluded scattering [Pegoraro and Parker 2009] as in [Wyman 2011] is trivially supported by our approach. The difficulty lies in the combined evaluation

and other approaches face this issue as well; we could follow the suggestion of [Baran et al. 2010] who rely on additional basis functions for the falloff. Nonetheless, such a solution is a bit cumbersome and combining all falloff and shadow basis functions would result in an increased memory consumption. A better alternative would be to directly choose basis functions which directly expand fractional visibility along light rays and support the falloff [Jansen and Bavoil 2010], but such a choice would decrease the shadow accuracy.

For omnidirectional sources, a single shadow map is insufficient, as it cannot cover an entire sphere. This case can be treated by defining six independent spotlights for each directional sector, corresponding to the face of a cube map. The different contributions of these sources are simply added.

Angular-dependency for Sources Integrating sources with changing angular contribution is straightforward; during the prefiltering step, each visibility function is simply weighted by the light's contribution just as is done for the transmittance weighting. In particular, this weighting allows us to simulate an angular falloff. If the light has an associated color texture, we need to store one value for each color component. Unfortunately, this step triples storage amount and computation costs.

3.3. Implementation

Here, we take a careful look at the implementation of the algorithm given a rectified shadow map as input, which can be built with our technique presented in the next section, or one of the existing previous solutions [Chen et al. 2011].

Prefiltering The first task is to compute the basis textures from the shadow map, which is a direct implementation of Eq. 8. The results of the sine and cosine are additionally mapped to the range of $[0;1]$, which allows us to store the basis textures as a RGBA8 `Texture2DArray` (usually 8 layers for $M = 32$ coefficients).

The next step is to filter the basis textures in a prefix-sum-like manner. Interestingly, the simplest implementation of the prefix sum also turned out to be the fastest and we present it in the following. We implemented a compute shader that launches one thread per row of the rectified shadow map. Due to the rectification, each row execution corresponds to a parallel execution over an epipolar slice. As each thread starts on the left and proceeds to the right, the values are written into each texel of the filtered-basis texture. In consequence, only very few registers are needed and the execution is fast.

The above computation addresses average visibility, however, as suggested in Sec. 3.2 more accuracy can be achieved by performing a weighted filtering. To compute ray-independent weights, in the last section we suggested to first choose a reference ray per epipolar slice. Computing this ray can be done in a 1D compute shader implementing the previously sketched procedure. The ray's travelled world-space dis-

```
#define NUM_BASES 32
// ...

void main() {
  // ...
  int y = gl_GlobalInvocationID.y; // epipolarSlice
  // in practice:  pack 4 bases in vec4
  float filtB[NUM_BASES];
  for(int i=0; i<textureSize(texInBasis,0).x; ++i) {
    for(int k=0; k<NUM_BASES; ++k) {
      float basis = texelFetch(texInBasis, ivec3(x,y,k), 0).r;
      // accumulate bases == increasing filter kernel
      // == (sum_{i=1}^{s} B_k(z_i)) / s
      filtB[k] = mix(filtB[k], basis, 1.f / (i+1));
      imageStore(imgOutFilteredBasis, ivec3(i,y,k), filtB[k].rrrr);
    }
  }
}
```

Listing 1. Basis-texture filtering.

tance Δ_{ref} for a shadow-map texel will determine the weighting factors. Δ_{ref} is simply the distance between two points on the ray, projecting to two neighboring texels of the recitified shadow map. Listing 2 shows the adaptations to the filtering to account for the non-uniform filter kernels.

Instead of storing $\sum_{i=1}^{s} w_i B_k(z_i)$ for each k, we normalize by the sum of the filter weights: $\frac{\sum_{i=1}^{s} w_i B_k(z_i)}{\sum_{i=1}^{s} w_i}$, which gives values that use the full range of $[0;1]$ (cf. the blend of the filter kernel up to $i-1$ and new value at i via mix in listing 1, 2). In consequence, low-precision filtered-basis textures can be used — no improvement was visible for textures with more than 8bit — leading to a significant reduction in memory and an increase in performance. It is easy to recover the correct value during the scattering evaluation for a specific view ray by multiplying the filtered basis-vector with $\sum_{i}^{s} w_i$. A more detailed derivation is shown in Appendix A.

A possibility to reduce bandwidth costs is to merge the shader to compute the basis transformation with the shader to perform the filtering. This effectively saves reading and writing of the entire basis texture once, which would otherwise happen in the filtering and transformation shader. The marginal downside of this is the added computational workload to the filtering shader as it needs to read shadow map values and transform those to the basis-vector:

The parallel execution over epipolar slices results in a process of low parallelization, potentially not fully utilizing the GPU during the prefiltering. We present two strategies to split the sequential filtering into smaller tasks that add a small amount of

```
uniform vec3 fPlaneItsWRefRay; // far plane intersection point
uniform float Δref;

void main() {
  // ...
  float stepSize = Δref;
  float opticalDepth = stepSize * sigma_t; // Δref σt
  // transmittance of a single step
  float stepTransmittance = exp(-opticalDepth);
  float totalTransmittance = 1;
  float sumWeights = 0;
  for(int i = 0; i < textureSize(texInBasis,0).x; ++i) {
    // directly use transmittance as weight ui = e^{-i Δref σt} = ∏_{j=0}^{i} e^{-Δref σt}
    // it has the same distribution as wi, but is faster to compute
    // hence, ui = c wi, while c does not matter
    totalTransmittance *= stepTransmittance;
    float weight = totalTransmittance;
    sumWeights += weight;
    for(int k = 0; k < NUM_BASES; ++k) {
      // ...
      // == (Σ_{i=1}^{s} ui Bk(zi)) / (Σ_{i=1}^{s} ui) = (c Σ_{i=1}^{s} wi Bk(zi)) / (c Σ_{i=1}^{s} wi) = (Σ_{i=1}^{s} wi Bk(zi)) / (Σ_{i=1}^{s} wi)
      filtB[k] = mix(filtB[k], basis, weight / sumWeights);
      // ...
    }
  }
}
```

Listing 2. Weighted basis-texture filtering.

```
void main() {
  for(int i = 0; i < textureSize(texInDepth, 0).x; ++i) {
    float depth = texelFetch(texInDepth, ivec2(x, y), 0).r;
    // ...
    for(int k = 0; k < NUM_BASES; ++k) {
      float basis = transformBasis(depth, k);
      // ...
    }
  }
}
```

Listing 3. Live transformation of the shadow map into basis functions.

bandwidth costs but increase parallelism in the prefiltering stage. The first choice is to parallelize over the k bases, i.e., multiple threads compute the filtering for the same epipolar slice, but for different bases (as bases are packed in `vec4`, the maximum parallelization is achieved with 4 bases per thread). However, this causes each thread

# Regions per Slice	# Bases per Thread	Performance in ms			
		Prefix Sum	+ Scattering	= Sum	+%
1	4	1.85	0.62	2.47	0%
1	8	2.49	0.62	3.11	-21%
1	16	4.05	0.62	4.67	-47%
2	4	1.19	0.96	2.15	+6%
2	8	1.37	0.96	2.33	-21%
2	16	2.03	0.96	2.99	-17%
3	4	1.50	1.04	2.54	-3%
3	8	0.93	1.04	1.97	+25%
3	16	1.32	1.04	2.36	+5%
4	4	1.28	1.33	2.61	-5%
4	8	1.16	1.33	2.49	-1%
4	16	1.10	1.33	2.43	+2%

Table 1. Computation time of our method with different parallelization parameters at the prefiltering stage for 32 basis functions. Note that we use the naive parallelization over epipolar slices and the 32 bases as the performance reference. As test scene we used *San Miguel* with a screen resolution of 1920×1080 and a shadow map resolution of 1024×1024, which is sufficiently high to produce artifact-free results in all the scenes shown in this paper.

to access the shadow map and compute the filtering weights although these tasks are basis-independent. The second strategy shifts work from the prefiltering stage to the final scattering evaluation. Instead of performing the prefix sum for an entire row, we split the row uniformly, e.g., into three regions, and compute the filtering independently, hereby increasing parallelism. During rendering, to evaluate a ray r, we consequently need several lookups; one from the region that contains the first visible surface along r, and one more for each filtered-basis texture region in front of it — with three regions, maximally three lookups. In combination, these two parallelization strategies increase the amount of parallel threads by an order of magnitude (e.g., $3\times$ by uniform splits and $4\times$ by each thread filtering only 8 of the 32 bases, we use per default). While performance can be improved in this way, we found it difficult to provide exact suggestions on how many splits to perform and bases to use per thread. The reason is that these numbers are highly hardware dependent. Furthermore, the speedup is mostly influenced by how efficient the initial parallelization over epipolar slices already performs, which, in turn, also depends on the shadow map resolution. For our hardware (an NVIDIA Titan), three regions and eight bases per thread led to a speed up of about 20% compared to a naive parallelization over all bases (in addition to the parallelization over epipolar slices); see Tab. 1.

We also tested a GPU-optimized prefix sum [Harris et al. 2007] designed for sums

of complete arrays (while we only require a weighted prefix sum per row and per basis). It makes use of shared memory and avoids some synchronization by performing work in pre-defined warp sizes. We implemented the algorithm with OpenGL compute shaders (applying the same optimizations) and adopted the version to perform filtering instead of computing sums. However, this variant is two times slower than the previously-described simpler approach. Probably, remaining synchronization steps and shared memory create a bottleneck, but this may depend again on the used hardware.

Evaluating a Ray The scattering evaluation for the view rays is performed in a simple fragment/compute shader, executed per screen pixel (see Lst. 4). The view ray is easily computed in the shader, e.g., by linearly blending the frustum vertices at the near / far plane.

```
void main() {
  Ray viewRay;
  vec3 sPosWS;
  float disToSurface;
  getViewRaySurfacePos(depthMap, pixelCoord, viewRay,
    sPosWS, disToSurface);
  vec3 sPosRS = toRectifiedSpace(sPosWS);
  vec4 coeff[NUM_BASES / 4], basis[NUM_BASES / 4];
  computeCoeff(sPosRS.z, coeff);
  fetchBases(basisFilteredTex, basis);
  float avgVis = 0.5;
  for(int k = 0; k < NUM_BASES / 4; ++k) {
    avgVis += dot(coeff[k], basis[k]);
  }

  vec3 C = getRayConstantScatteringParam(viewRay);
  vec3 inScat = C / σt * (1 - exp(disToSurface*σt)) * avgVis;
}
```

Listing 4. Scattering evaluation per view ray.

Assuming a deferred-shading pipeline, we retrieve the depth component from the G-Buffer to compute the surface point up to which the single scattering integral has to be computed. We map it to the space of the rectified shadow map (Sec. 4) to query the filtered-basis vector — on a side note, in the case of our novel rectification the mapping is a simple projective transformation just as with usual shadow mapping. Next, we compute the coefficients of the view ray using Eqs. 8 and the z-component of the transformed surface point, which results in the z value of the view ray, as the latter is constant along the ray, hence, also at the hit point. We evaluate the dot product between coefficients and filtered-basis functions (mapped to range $[-1;1]$ after the

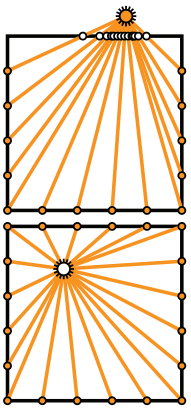
texture access) in a loop and add a constant of 0.5 as required by the Fourier series approximation. Note that the pseudo-code is the same for average visibility as well as transmittance-weighted visibility as in the latter case the filtered-basis vectors are stored normalized (by $\sum_{i=1}^{s} w_i$) and are restored by the transmittance to the surface point as by definition $\sum_{i=1}^{s} w_i = \sigma_t \int_0^s T(\mathbf{x}, \mathbf{x}_t)\, dt = 1 - e^{-s\sigma_t}$.

4. Rectification

In this section, we discuss the creation of a rectified shadow map, in which all view rays have a constant z_r value along the rays as required by Eq. 7, run from left to right in the shadow map and are all parallel to each other. We first recapitulate the standard resampling approach of Baran et al. [2010], which is based on epipolar coordinates. Then, we introduce a novel linear rectification scheme that uses a projective transformation and allows to directly rasterize a scene into a rectified shadow map.

4.1. Epipolar Geometry

The method of Baran et al. [2010] uses epipolar geometry to obtain a rectified shadow map. In principle, the mapping of a point from world to the rectified space is defined as follows: z is the angle spanned by the view ray and the light direction in a range of $[0;\pi]$, y is the angle, which in conjunction with the camera position and the light direction defines the epipolar slice, which has a range of $[0;2\pi]$, and x is the length of the vector from the camera to the projection of a point onto a reference ray in the epipolar slice, see Fig. 2. Conveniently, we can choose the optimal reference ray (defined in Sec. 3.2) as a reference ray.



However, x still needs to be bounded tightly and view-dependent bounds need to be found for y, z. These tight bounds are important to avoid undersampling and to increase precision. Given the bounds, we can then define the mapping from world to rectified space.

To find the bounds for y, indicating the epipolar slice, it is possible to sample the screen border pixels and derive their epipolar slices based on their view ray directions respectively. Hereby, we implicitly define the y resolution of the rectified shadow map (up to $2\times$ width and height of the screen) and ensure that any screen pixel is at most half a pixel away from the closest epipolar slice that is captured in the rectified shadow map. All other pixels on the screen can be associated to their corresponding slice by projecting its position along the epipolar line to the border of the image. Engelhardt and Dachsbacher [2010] as well as Baran et al. [2010] discuss optimal epipolar sampling strategies that avoid some of the unnecessary oversampling of the screen border approach.

Bounds for the angle z between view ray and light direction can be found by examining the limiting view rays of an epipolar slice. The first limiting ray is already given by the epipolar slice sampling approach (border pixel, yellow dots in figure), the second is either the epipole (light source), if it is within the screen area or the projection of the first border pixel into the direction of the epipole to the screen border (white dots).

Finding an upper bound for x, the length of the vector from the camera to the projection of a point within the epipolar slice onto a reference ray is also simple. For each epipolar slice, we take its reference ray and proceed as follows; we project the far plane points of the limiting rays of the epipolar slice onto the reference ray, compute the distance to the camera, and take the maximum.

The transformation of the scene into the rectified shadow map based on the bounds derived for each epipolar slice does not preserve straight lines, which is a challenge for the graphics card. Hence, it is more efficient to first create a standard shadow map, which is subsequently point-resampled into a rectified shadow map. The reprojection is performed in a compute shader over all texel in the rectified shadow map. Each light ray, defined by the texel position in the rectified shadow map, can be mapped to world space and then to the shadow map to recover the necessary input shadow map texel. Given its value, we can then compute the necessary depth value for the rectified shadow map.

One catch are blockers between the light and the camera frustum. They would not necessarily fall within the ranges defined by the bounds of the rectified shadow map. This problem can be solved easily by clamping their original shadow-map depth to the camera frustum. Hereby, their values become compatible with the range, but they cast the same shadows into the visible scene.

Although the performance of the resampling is high (1024×1024: 0.1ms, 4096×1536: 0.6ms, 4096×4096: 1.5ms), it is preferable to avoid resampling in general because it can lead to severe under or oversampling. Further, this mapping does not allow us to easily perform efficient lookups, e.g., via a simple matrix multiplication, as for shadow mapping. Instead a projection of the pixel onto the screen border is needed and more involved. In the following, we will present a new technique to address these issues.

4.2. Our Projective Rectification

Here, we introduce our linear rectification, which avoids reprojection completely by constructing a special projection matrix to replace the standard light view-projection matrix during the rendering of the shadow map. In consequence, points can be mapped easily into this space by applying a homogeneous transformation. Hence, not just the creation of the rectified shadow map is simplified, but also lookups into it. To find the corresponding transformation, we construct a frustum with an associated

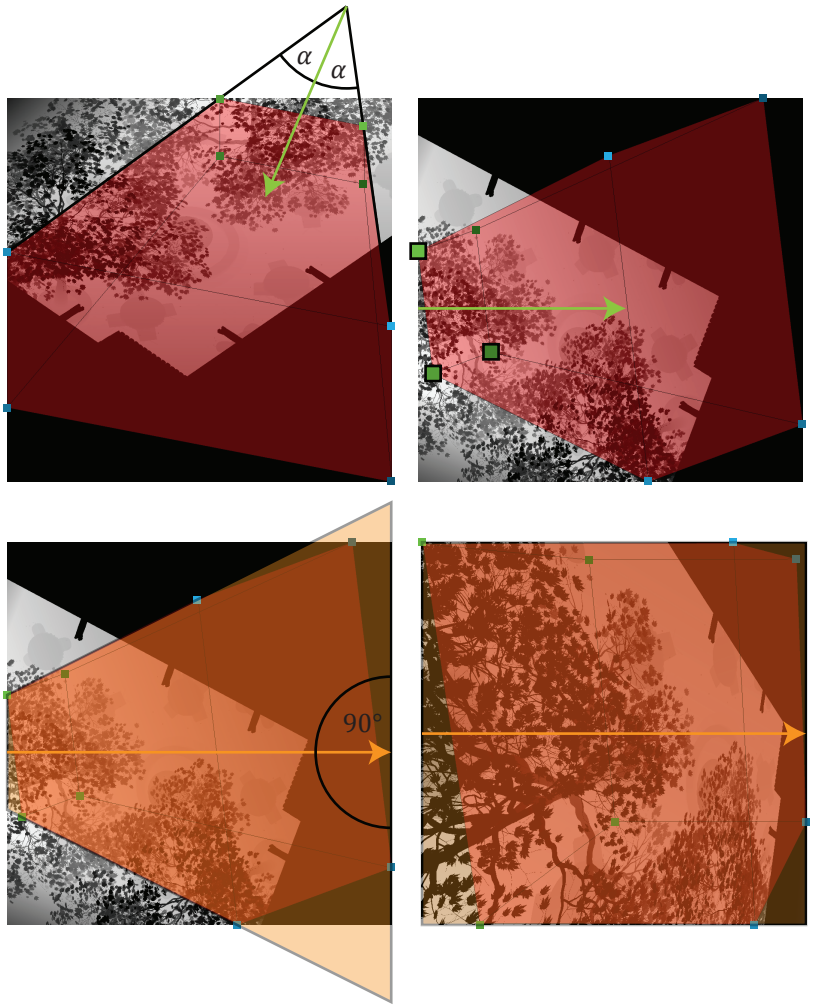


Figure 4. Camera frustum overlayed on shadow map. Rectification (top left to bottom right): standard light view transformed allows to find 2D bisectrix; new view transform such that bisectrix goes from left to right; light frustum with near/far planes parallel to the light direction; rectification via perspective projection.

matrix that maps the scene in such a way that it fulfills the constraints of our scattering approach, i.e., all view rays in an epipolar slice map to a row of the shadow map, the depth along a view ray is constant, rays in this light space go from left to right, and the shadow map tightly encompasses the camera frustum.

Constructing the Linear Rectification Matrix Fig. 4 gives an overview of the construction steps for a directional light source. While they could be grouped, this step-by-step construction is easier to follow. We first determine the 3D bisectrix of the camera frustum projected along the light direction (Fig. 4, top-left), which is achieved in multiple small steps as follows. We apply a standard light space *view* transform from the cam-

era, in direction of the light using any possible up vector. Next, we project the camera frustum corners into this space and find the 2D vectors from the projected camera position to projected corners that span the biggest angle. From these two vectors, we compute the 2D bisectrix, which is turned into a 3D vector by setting $z = 0$ in the light space. In consequence, this 3D bisectrix is orthogonal to the light direction. The bisectrix is used to find a new up vector for the light space *view* transform, which is a rotated version around the light direction, such that the bisectrix now goes from left to right (Fig. 4, top-right).

Next we seek a transformation with the camera as the center of projection to get view rays that are parallel to each other. We need to find a light frustum that is wrapped around the camera's frustum. Choosing the y and z axis to define the parallel planes of the frustum allows the use of the common projection matrix form with the parameters near/far, left/right, and top/bottom offsets from the center of projection. However, different from the common use of perspective projection along z, we perform a projective transformation along x, hence, near and far correspond to the minimal and maximal x value, respectively, of the camera frustum corners (Fig. 4, bottom-left). The frustum's left/right (side planes, y axis) and top/bottom (top planes, z axis) offsets are also found by the minimal and maximal y, z coordinates of the corner points (marked dots in Fig. 4, top-right). Filling the projection matrix with the computed parameters yields the final transformation. Note that the bisectrix causes that parameters left/right to have the same absolute value, which not necessarily holds for top/bottom. Hence, the light frustum can be skewed, but this has no negative effect on the projection. The final matrix can be used to generate a rectified shadow map with all needed properties (Fig. 4, bottom-right). Additionally, using depth clamping (NV_DEPTH_CLAMP) when rendering the shadow map ensures that objects shadowing the camera frustum, but lying outside of it, are rendered in the shadow map as well.

It is important to note that the construction is not possible if any of the view rays is parallel to the light direction. This is the case, when the epipole is within the screen area, i.e., when looking directly into the light source resp. into the same direction as the light. In this situation, we need to switch back to a texel-wise reprojection.

Non-linearity of the Rectification Our frustum-based rectification results in a non-linearity along the view rays (x-axis in the shadow map), similar to the non-linearity of the depth buffer. The effect increases the more view and light rays are parallel to each other. Thus, we also return to texel-wise reprojection for close-to-degenerated cases (easily detectable as distance of the epipole to screen border). However, the increased shadow map resolution near the observer can be an advantage, as, due to transmittance, the in-scattered light near the observer receives more precision.

We reduce the non-linearity by using a method similar to split shadow maps [Zhang

et al. 2006]. Here, we split the light view into different regions (from left to right), which increase in size. If each region is then rendered into a shadow map of the same resolution, the higher resolution near the observer is counteracted. Similar to epipolar geometry, straight lines are not preserved anymore over all splits, hence, we need to rasterize the scene multiple times, once for each split. The splitting here is conceptually equal to the splitting for increased parallelization at the prefiltering stage, described in Sec. 3.3 and can be combined with this step.

Using the rectification has an impact on the weights w_i that modulate the prefix sum in the scattering approach (cf. Eq. 11). In our rectification, the travelled distance along a view ray increases by a constant factor d from one shadow-map texel to the next. This result stems from the fact that a perspective transformation, such as our rectification, preserves the cross ratio of points [Heckbert 1989]. The weight becomes

$$w_i' := e^{-\Delta\sigma_t \Sigma_{j=1}^{i} d^{j-1}} (1 - e^{-\Delta\sigma_t d^i}). \tag{12}$$

To compute d, we transform the first three texel centers of a shadow-map row into world space (the depth does not matter, but need to be the same for all three) and compute the cross ratio of the distances. For a given shadow-map row, the factor d only needs to be determined once before launching its prefiltering. Please note that this ratio relation along the ray is not the same as the previously established ratio relation between the different rays of an epipolar slice.

Discussion Our novel rectification method has several advantages. First, no resampling is needed. Second, mappings from world to rectified space boil down to a matrix-vector multiplication similar to common shadow mapping. This operation is faster to execute and easier to implement than the projection of a pixel to the screen border to compute the epipolar slice. Third, the shadow map resolution can be chosen independently of the transformation; complex epipolar slice sampling is avoided. Due to the regular sampling of the epipolar slices, the transformation provides near optimal sample distribution under any resolution. While we selected the shadow map resolution manually, it remains future work to find the minimum y resolution under the constraint that each pixel should be at most half a pixel away from the closest epipolar slice as has been done for epipolar geometry [Baran et al. 2010]. Nonetheless, given that our approach also supports texture filtering, this half-pixel distance does not have to be as accurate as in previous solutions.

As indicated in Sec. 3.1, the camera's near plane is usually not parallel to the light direction, therefore, different view rays have a different coverage of the camera to near plane (cf. Fig. 4, bottom-right, near plane corners in green). While we already described a solution to this imprecision, we want to underline again that, practically, this didn't cause any artifacts in our test cases because the near plane is usually close to the camera and the uncovered area is thus very small.

The projection warps a light frustum placed around the camera frustum by using the camera as center of projection. Consequently, the camera itself should not lie in the light frustum and the resulting rectified shadow map misses visibility information close to the camera. However, by construction, it contains the entire camera frustum, thus, at most the region from the camera to the camera's near plane is not accessible in the rectified shadow map. Consequently, our approach cannot replace the epipolar resampling scheme in all cases. Although, while highly impractical, one could avoid view rays being parallel to the light direction by splitting the camera frustum in multiple smaller frusta (such that the corners of the sub-frustum span an angle less than 180 degrees, cf. the first step of the matrix construction) and omit pixels on the epipole to fix this situation.

5. Results

We tested our prototype on an Intel Core i7 system with an NVIDIA Titan card. The method was implemented in OpenGL using compute shaders. In this section, we present our results and compare to previous work. We also discuss benefits and limitations of our solution.

All results in this section were computed at full-HD screen resolution (1920×1080). Per default, we use $M = 32$ coefficients (sine and cosine alternating) per texel, which proves sufficient in practice. Note that in our previous work [2014], we suggested the use of 16 *coefficients*, but count sine *and* cosine only once as it responds to one order of the Fourier expansion of the Heavyside step function. Nonetheless, the same amount of data is used, stored in an eight-layer RGBA8 `Texture2DArray`. Fig. 5 shows a comparison between different methods. We evaluated quality and performance on scenes with differing complexity. Our method performed in all cases below 3 ms, while still resulting in a high-quality rendering. Note that our timings exclude rendering time for the creation of the shadow map, and we only state the overhead due to the additional effect of single scattering.

The upper row in each scene set shows the result with optimal quality settings of each method, i.e., the parameters were chosen as suggested by the authors of the previously-published techniques, although, when not leading to a lower quality, we reduced the shadow-map resolution more to enable a fairer comparison. Our approach reached the highest performance in all cases. For the San Miguel model (top rows and also shown in Fig. 6), our solution was over $20\times$ faster than the min-max-mipmap approach [Chen et al. 2011]. The low performance is a direct consequence of the complex depth map; the many leaves of the tree lead to an overhead on the min-max hierarchy (as also observed in [Baboud et al. 2012]). For simple scenes, such as the terrain, the min-max structure reaches optimal performance, but our method is still 1.8 times faster (due to lower resolution shadow maps with hardware filtering).

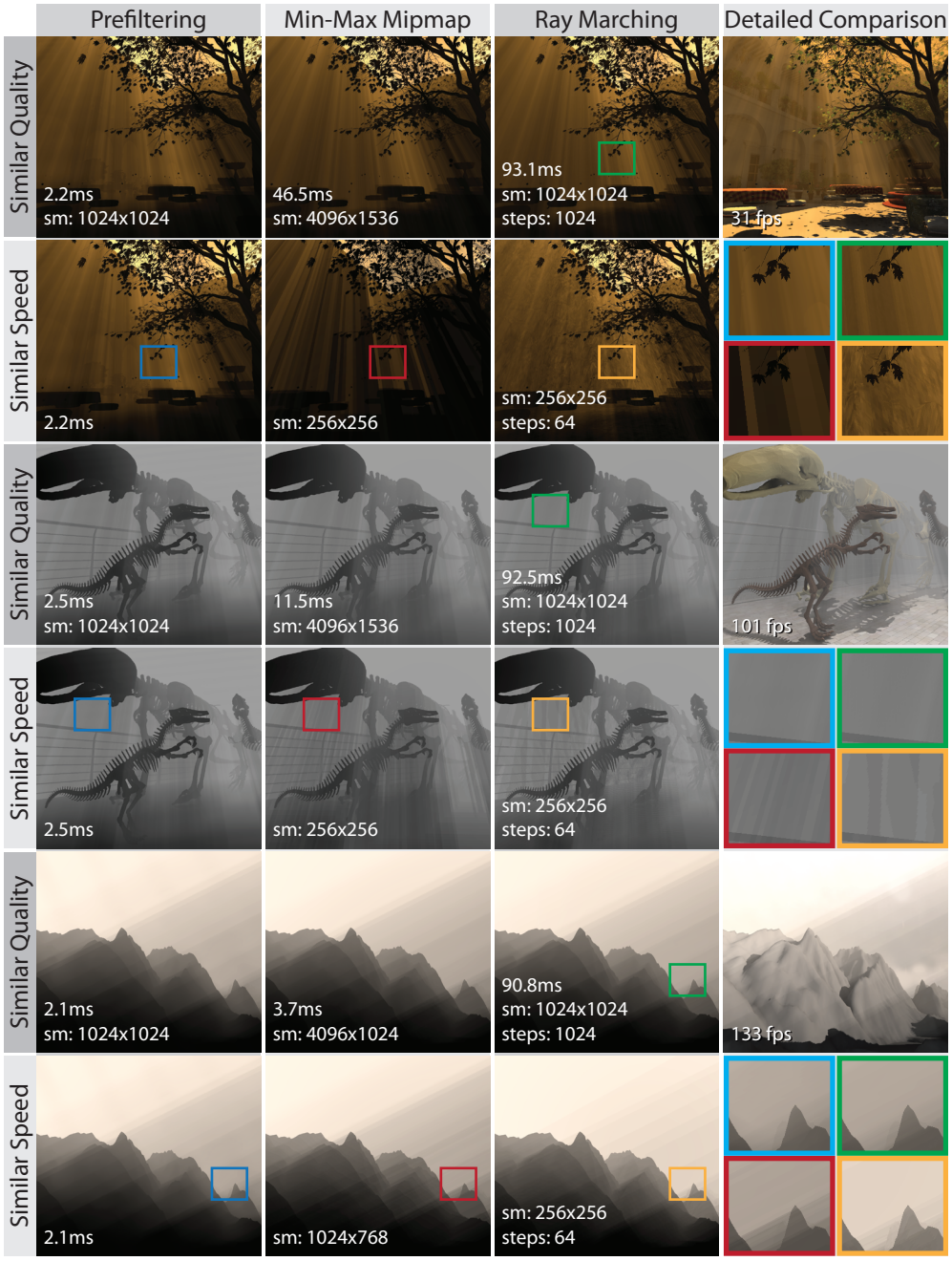


Figure 5. Quality/Performance comparison at 1920×1080. To compare quality, we chose the suggested settings of all methods. For same time, we reduced the shadow-map resolution. Our result compares favorably to reference quality, even for low-resolution shadow maps.

These comparisons show the main benefit of our technique: it is independent of the scene complexity and configuration, while the cost of other methods is not easy to

Figure 6. Single scattering contributes significantly to the appearance of the scene and, using our method, its computation time is low and predictable. Making it possible to meet a fixed target framerate, which makes it a good candidate to be used together with other lighting effects that share this property (here ambient occlusion, surface shadows, alpha matting).

predict. In this regard, our solution is very different from other approaches whose run time is more strongly scene dependent [Billeter et al. 2010; Engelhardt and Dachsbacher 2010; Baran et al. 2010]. Our cost is $\mathcal{O}(wh + ad)$ for a screen resolution of $w \times h$, a maximum of d view ray integration steps, and the number of epipolar slices a. Brute-force ray marching leads to d integration steps per pixel, yielding $\mathcal{O}(whd)$. Acceleration structures [Baran et al. 2010; Chen et al. 2011] can lead to an improvement of, potentially, $\mathcal{O}(wh \log d + ad)$. However, while $\log d$ is the optimal case, it can become d in the worst case. Our method shows constant performance.

The lower row in each scene set shows an equal time comparison, where we adapted the shadow map resolution of the competitors until our performance was roughly matched. The resulting images clearly exhibit artifacts stemming from discretization problems. Further, our method makes it possible to enable hardware filtering, while other approaches do not have this option. In consequence, our results remain visually pleasing, even with lower resolution shadow maps.

A performance breakdown of our solution for San Miguel is shown in Tab. 2. The scattering cost is constant, while the prefix sum scales linearly with resolution, as expected. Changing the amount of coefficients behaves sub-linearly. This result is probably linked to the cache and texture mechanisms; as 32-bit floating point computations are standard and on newer cards even 64, it is likely that the bandwidth has been increased correspondingly. Hence, our solution also seems well-suited for future hardware developments.

In theory, compared to the min-max tree of [Chen et al. 2011], we require 2.6× (32-bit floats for depth, min-max) or 5.3× (16-bit floats) more memory. Nevertheless, we can rely on hardware filtering when accessing the prefiltered basis functions, better reducing potential undersampling artifacts. This possibility allows us to reduce the resolution significantly (usually by a factor of six; cf. Fig. 5). Using this additional bilinear filtering is beneficial along the x-axis, which averages discrete integration

Shadow Map Resolution	Count Basis	Performance in ms			Memory in MB
		Prefix Sum	+ Scattering	= Sum	
512×1024	32	0.54	1.47	2.01	16
1024×1024	32	0.94	1.30	2.24	32
2048×1024	32	2.17	1.31	3.48	64
4096×1024	32	4.32	1.35	5.66	128
1024×512	32	0.97	1.47	2.44	16
1024×768	32	1.03	1.50	2.53	24
1024×1536	32	1.78	1.52	3.31	48
1024×1024	16	0.96	0.92	1.88	16
1024×1024	64	3.07	2.63	5.71	64
1024×1024	128	4.61	5.00	9.61	128

Table 2. Computation time and memory consumption of our method with different parameters. As test scene we used *San Miguel* for a screen resolution of 1920×1080, where the method achieves near ground truth results with the green highlighted parameters. Note that the basis counting deviates from Klehm et al. [2014] as we count the sine *and* cosine here.

steps, as well as along the y-axis, which blends neighboring epipolar slices. The latter is often desired as it avoids unnaturally sharp edges without requiring an extensive oversampling. Due to hardware filtering our method constantly achieves high performance and quality, while previous work relies on higher resolution textures [Chen et al. 2011] or customized upsampling strategies [Baran et al. 2010; Wyman 2011].

Ringing is a potential issue, but is almost mostly hidden with 32 bases (Fig. 7 third column). For more complex scenes that create depth variation in the shadow map the ringing artifact is even completely masked when integrating along the view ray. This can be observed for the San Miguel scene (Fig. 7 top row), where our method achieves good quality even with a very low number of 4 basis functions. Hence, our method works well even with complex geometry as can be found in today's computer games. In contrast, the masking does not occur for planar regions in the shadow map, hence, the shown scene with a directional light from the top is a particularly difficult case. Note that also scene details (textures, materials, surface lighting, etc.) tend to hide the artifact even further (usually completely). Our choice of 32 basis functions is conservative to avoid also any temporal artifacts.

Similar to Baran et al. [2010], we need to perform a brute-force ray marching in a small block of pixels around the light source, if it is directly visible. Epipolar geometry has low precision in these areas as view rays and light rays are nearly parallel. As result, the light source *bleeds* through the occluders or their occlusion is overestimated. This effect can be reduced if a depth warp is applied to the rectified shadow map, which basically stretches the rays. However, the effect cannot be

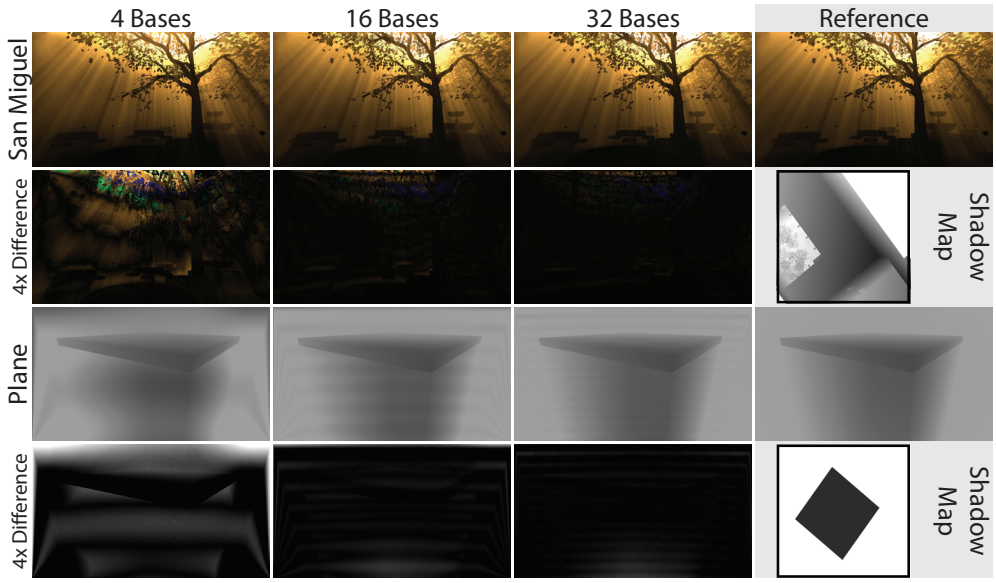


Figure 7. The effect of ringing dependent on the number of basis functions (scattering component of Fig. 6). Right column: reference (ray marching) and the shadow map for the scene. Ringing is less prominent for shadow maps with increased depth variation.

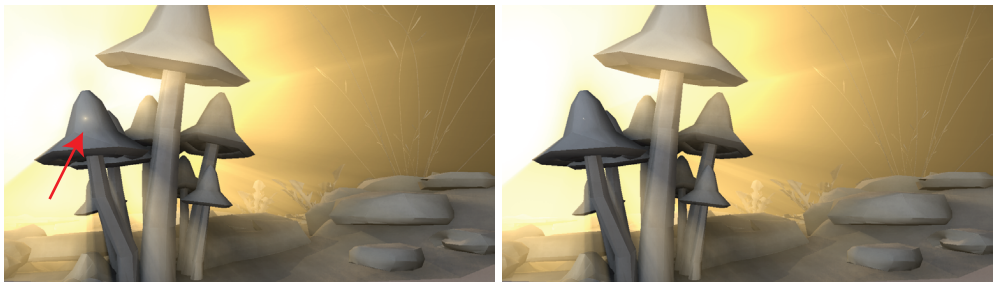


Figure 8. Left: Our method causes light bleeding around the epipole as the precision in the epipolar geomtry is low. Right: Reference using ray marching. This can be practically fixed by ray-marching a small block of pixels around the epipole.

avoided completely as shown in Fig. 8.

6. Conclusion

We present an efficient single-scattering approach, with a linear, but decoupled complexity $\mathcal{O}(wh+ad)$ of screen pixels wh and shadow map resolution ad. The prefiltering process is fast and our method is compatible to hardware interpolation. The latter allows us to use smaller resolutions for the shadow maps than in previous approaches, which results in lower memory requirements. While our method can exhibit ringing artifacts, those partially cancel out in complex scenes, due to the increased depth

variation in the shadow map. Textures and other scene details mask the effect further, making 32 bases a safe choice. In contrast to other solutions, our method delivers predictable and generally high performance combined with convincing image quality, and is independent of the scene size, complexity, or detail level. Hence, our approach is a good candidate for time-critical applications.

We also propose a new rectification method, which makes use of view and projection matrices similar to shadow-map approaches. Applied to the scene, the matrices lead to a rectified shadow map where view rays are parallel to each other and which can be used for our filtering to compute single scattering. This rectification avoids resampling of a standard shadow map, however, the procedure is only possible if the view and light rays are not parallel to each other, i.e., the light source is not directly visible. In this case, we need to resort to resampling approach as previous methods.

In the future, we will investigate whether our rectification can be coupled more closely to the actual ray marching process to increase performance and quality.

Acknowledgements

San Miguel was made by Guillermo M. Leal Llaguno. This work was partly supported by the Intel Visual Computing Institute at Saarland University.

References

ANNEN, T., MERTENS, T., BEKAERT, P., SEIDEL, H.-P., AND KAUTZ, J. 2007. Convolution shadow maps. In *Rendering Techniques (Proc. of Eurographics Symposium on Rendering)*, 51–60. URL: http://dx.doi.org/10.2312/EGWR/EGSR07/051-060. 8, 11, 12, 14

BABOUD, L., EISEMANN, E., AND SEIDEL, H.-P. 2012. Precomputed safety shapes for efficient and accurate height-field rendering. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 18*, 11, 1811–1823. URL: http://doi.ieeecomputersociety.org/10.1109/TVCG.2011.281. 27

BARAN, I., CHEN, J., RAGAN-KELLEY, J., DURAND, F., AND LEHTINEN, J. 2010. A hierarchical volumetric shadow algorithm for single scattering. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia) 29*, 6, 178:1–178:10. URL: http://doi.acm.org/10.1145/1882261.1866200. 8, 10, 16, 17, 22, 26, 29, 30

BILLETER, M., SINTORN, E., AND ASSARSSON, U. 2010. Real time volumetric shadows using polygonal light volumes. In *Proc. of High-Performance Graphics (HPG)*, 39–45. URL: http://dl.acm.org/citation.cfm?id=1921479.1921487. 29

CHEN, J., BARAN, I., DURAND, F., AND JAROSZ, W. 2011. Real-time volumetric shadows using 1d min-max mipmaps. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 39–46. URL: http://doi.acm.org/10.1145/1944745.1944752. 7, 8, 10, 11, 16, 17, 27, 29, 30

ENGELHARDT, T., AND DACHSBACHER, C. 2010. Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 119–125. URL: http://doi.acm.org/10.1145/1730804.1730823. 22, 29

HARRIS, M., SENGUPTA, S., AND OWENS, J. D. 2007. *GPU Gems 3*. ch. 39. Parallel Prefix Sum (Scan) with CUDA, 851–876. 20

HECKBERT, P. S. 1989. Fundamentals of texture mapping and image warping. Tech. rep. 26

JANSEN, J., AND BAVOIL, L. 2010. Fourier opacity mapping. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 165–172. URL: http://doi.acm.org/10.1145/1730804.1730831. 17

KLEHM, O., SEIDEL, H.-P., AND EISEMANN, E. 2014. Prefiltered single scattering. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 71–78. URL: http://doi.acm.org/10.1145/2556700.2556704. 8, 14, 27, 30

PEGORARO, V., AND PARKER, S. G. 2009. An Analytical Solution to Single Scattering in Homogeneous Participating Media. *Computer Graphics Forum (Proc. of Eurographics) 28*, 2, 329–335. 16

WYMAN, C. 2011. Voxelized shadow volumes. In *Proc. of High-Performance Graphics (HPG)*, 33–40. URL: http://doi.acm.org/10.1145/2018323.2018329. 16, 30

ZHANG, F., SUN, H., XU, L., AND LUN, L. K. 2006. Parallel-split shadow maps for large-scale virtual environments. In *Proc. of ACM International Conference on Virtual Reality Continuum and Its Applications*, 311–318. URL: http://doi.acm.org/10.1145/1128923.1128975. 26

A. Normalized Filtered-Basis Functions

The following shows that we can store normalized filtered-basis functions B_k to better make use of fixed-point 8bit textures (cf. Sec. 3.3):

$$\begin{aligned} L_{\text{scat}}(\mathbf{x},\omega_{\text{i}}) &\approx C\left[0.5\sum_{i=1}^{s} w_i + \frac{\sum_{i=1}^{s} w_i}{\sum_{i=1}^{s} w_i}\sum_{k}^{M} a_k(z_r)\left(\sum_{i=1}^{s} w_i B_k(z_i)\right)\right] \\ &= C\sum_{i=1}^{s} w_i\left[0.5+\sum_{k}^{M} a_k(z_r)\left((\sum_{i=1}^{s} w_i)^{-1}\sum_{i=1}^{s} w_i B_k(z_i)\right)\right] \end{aligned}$$

As $B_k(z_i)$ is by definition in the range of $[-1;1]$ (cf. Eq. 8), $(\sum_{i=1}^{s} w_i)^{-1}\sum_{i=1}^{s} w_i B_k(z_i)$ is as well and can be mapped to $[0;1]$ before writing to the texture. We can also make another observation; the truncation of the Fourier series causes unwanted ringing artifacts. Consequently, the expression within the brackets $[0.5+\ldots]$ is not necessarily in the range of $[0;1]$. Nonetheless, it can be safely clamped, which reduces the artifacts.

Author Contact Information

O.Klehm and H-P. Seidel
MPI Informatik
Campus E1 4
66123 Saarbrücken, Germany
{oklehm, hpseidel}@mpi-inf.mpg.de
http://people.mpi-inf.mpg.de/~{oklehm, hpseidel}

Elmar Eisemann
Delft University of Technology
Mekelweg 4
2628 CD, Delft, Netherlands
e.eisemann@tudelft.nl
http://graphics.tudelft.nl/~eisemann

O. Klehm, H.-P. Seidel, E. Eisemann, Filter-based Real-time Single Scattering using Rectified Shadow Maps, *Journal of Computer Graphics Techniques (JCGT)*, vol. 3, no. 3, 7–34, 2014
http://jcgt.org/published/0003/03/02/

Received:	2013-10-22		
Recommended:	2013-12-09	Corresponding Editor:	Carsten Dachsbacher
Published:	2014-08-16	Editor-in-Chief:	Morgan McGuire