Vision, Modeling, and Visualization (2014)
J. Bender, A. Kuijper, T. von Landesberger, H. Theisel and P. Urban (Eds.)

Deep Screen Space for Indirect Lighting of Volumes

Oliver Nalbach[1] Tobias Ritschel[1,2] Hans-Peter Seidel[1]

[1]MPI Informatik [2]Saarland University / MMCI

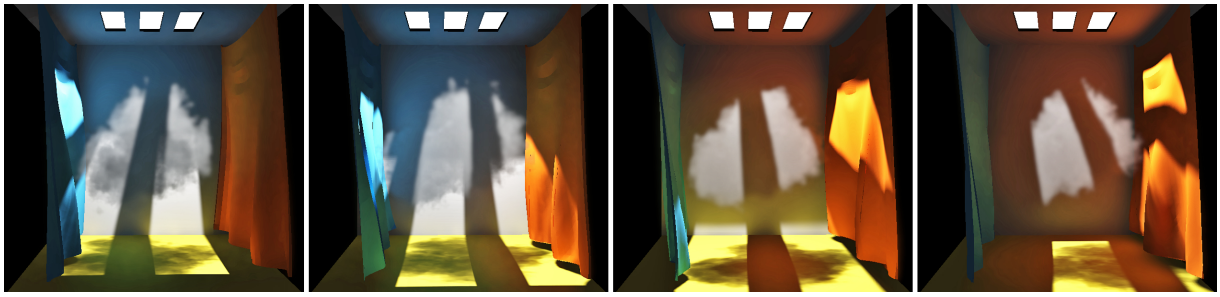

Figure 1: *Cropped frames from an animated scene with dynamic light, changing participating medium and deforming surfaces, rendered with our surface-to-volume indirect lighting at* 25 *fps,* 800×600 *pixels.*

Abstract

We present a method to render approximate indirect light transport from surfaces to volumes which is fully dynamic with respect to geometry, the medium and the main light sources, running at interactive speed. This is achieved in a three-step procedure. First, the scene is turned into a view-dependent level-of-detail surfel cloud using fast hardware tessellation. These surfels are lit and represent the senders of indirect light. Second, the current view of the volume is converted into a transmittance interval map, containing depth intervals in which the transmittance to the camera is reduced by the same fraction of the total extinction. These intervals will receive indirect illumination. Finally, surfels and intervals are linked by splatting the effect of the surfels into a hierarchical framebuffer. This linking delivers high precision between surfel-interval pairs that exchange much light and is coarser for pairs exchanging little, without constructing any explicit hierarchical data structure.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity

1. Introduction

Plausible simulation of indirect lighting constitutes a major step towards the realistic appearance of virtual scenes. This is not exclusive to the light transport between surfaces but also applies to participating media. Rendering of participating media is still an important challenge even for offline rendering. While offline rendering of participating media, e. g., by means of radiosity [RT87], discrete ordinates [Fat09], diffusion [Sta95], photons [JC98], beams [NNDJ12] or Metropolis light transport [PKK00] can produce stunning imagery, these methods usually do not deliver interactive results, in particular for interactive scenes. In particular, the interaction of surfaces and volumes in lighting has until now only received little attention. We present a method to compute approximate indirect light transport from surfaces to volumes which is fully dynamic with respect to geometry, the medium and the main light sources, running at interactive speed.

Our method is based on the idea of screen space shading [Mit07], where fast rasterization is used to create a framebuffer on which shading is computed. In particular, we build on the idea of a deep screen space [NRS14], where a scene is tessellated on-the-fly into a surfel-based representation instead of rasterizing it into pixels. The resulting point cloud is a discrete but complete scene description that overcomes

DOI: 10.2312/vmv.20141287

important limitations of screen space, while retaining most of its efficiency. The surfel cloud allows to compute detailed shading for the near range and approximate shading over larger distances by splatting to a hierarchical framebuffer.

Deep screen space was used to compute surface shading, such as ambient occlusion [Mit07], directional occlusion and bounces [RGS09] or subsurface scattering [JSG09]. In this work, we extend deep screen space to indirect light transport from surfaces to volumes. The main challenge is to find a representation that is suitable for splatting onto. A simple solution would be to splat cubes to a 3D texture in analogy to splatting quads into a 2D texture. This however, would consume prohibitive amounts of storage and compute time to retain sufficient detail. Instead, we propose to first cache transmittance for the ray from the camera into the volume at every pixel. This is similar to Deep Shadow Maps [LV00], but taking the sensor's point of view. The ray is marched once, and we store the depths, at which the accumulated change of transmittance exceeds certain values. Effectively, the ray under each pixel is discretized into ray intervals with likely similar contribution to the sensor. The usual splatting of the surfels' contribution onto pixels representing pieces of surface is then replaced by splatting onto ray intervals.

2. Previous work

Screen space shading is popular, e. g., for ambient occlusion [Mit07], but suffers from the fact that the information about the scene is typically limited to very few or only one depth layer, producing bias. Deep screen space [NRS14] overcomes some of the screen space limitations by computing a scene representation which is view-dependent but complete.

For indirect lighting of volumes, many offline methods based on the idea of instant radiosity [Kel97] exist. Virtual point lights (VPLs) are distributed in the scene from which light is gathered at sampling points inside the medium [RSK08]. One shortcoming of those methods is that the discretization to points leads to intensity singularities close to the VPLs. This problem has been overcome by means of intelligent bias compensation [ENSD12] or exchanging the point lights by ray lights [NED11]. Weber et al. [WKSD13] show how to compensate for changes in the rendered volume in an efficient way by progressively readjusting the VPLs.

Volume rendering based on diffusion [Fat09, KD10, ERDS14] is a general solution to render different types of illumination, also supporting multiple scattering. This sometimes includes indirect light from surfaces [ERDS14] by using VPLs. The rendered volumes however typically have a low spatial resolution and the interaction between surfaces and volumes is limited to a low number of VPLs. Our technique resolves fine details between surface pieces and their nearby rays on a resolution close to the pixel resolution, at similar speed, and uses thousands of virtual lights.

For the case of single scattering from direct light in homogeneous media, recently highly efficient methods emerged, e. g., by making use of prefiltering and rectification [KSE14]. In contrast, our method deals with indirect single scattering in heterogeneous media.

Regarding indirect single scattering from specular surfaces (volume caustics) in heterogeneous media, the method by Hu et al. [HDI*10] interactively produces plausible results. Leveraging that specular surfaces reflect light in only few directions, pixels affected by caustics are bounded by lines which are cheap to splat. However, our method deals with reflections from diffuse surfaces, where the indirect light is spread over a much larger region of space.

Our transmittance interval map is similar to Deep Shadow Maps [LV00] but created from the sensor's perspective. Different from the original, our map is built in linear time while avoiding sorting or iteration over the transmittance values which do not fit GPUs. Opacity shadow maps [KN01] seek to improve the efficiency of deep shadow maps by representing the volume using geometric primitives and rasterizing them to planar maps which perpendicular to the light's direction, integrating the density along rays from the light. In contrast to that, we don't use planar slices as we cannot expect them to divide the medium into intervals of similar importance, yielding good sampling intervals. Deep opacity maps [YK08] iterate the idea by using slices of same depth with respect to the depth at which the medium is entered in each pixel. However, the approach targets hair rendering and, using regular intervals, implicitly assumes a largely homogeneous medium. Again for rendering hair, Mertens et al. [MKBVR04] construct a compact representation of 1D visibility functions along rays, too, however, it works by rasterizing individual hairs and cannot be directly transferred to continuous volumes. Inverse to all these methods, we store a mapping from transmittance values to depths, from the perspective of the camera, instead of from the light's point of view.

3. Background

As our approach method computes single scattering (Sec. 3.1) using a deep screen space pipeline (Sec. 3.2) we will first recall the definitions and notations regarding both.

3.1. Single scattering

In computer graphics, the interaction of light with a participating medium is often described by the equation of radiative transfer [Cha50, PH10]:

$$L_\mathrm{i}(\mathbf{x},\omega_\mathrm{i}) = \underbrace{\tau(\mathbf{x}\leftarrow\mathbf{x}_\mathrm{s})\,L_\mathrm{o}(\mathbf{x}_\mathrm{s},-\omega_\mathrm{i})}_{\text{Attenuated light from first surface}} + \underbrace{\int_0^t \tau(\mathbf{x}\leftarrow\mathbf{x}')\,S(\mathbf{x}',-\omega_\mathrm{i})\,\mathrm{d}t'}_{\text{Attenuated in-scattered or emitted light}}$$

Here, $\mathbf{x}_s$ is the location of the first surface in direction ω_i when starting at $\mathbf{x}$, $\tau(\mathbf{a} \leftarrow \mathbf{b})$ is the *transmittance* function, describing the amount of light surviving *absorption* and *out-scattering* between points $\mathbf{b}$ and $\mathbf{a}$, $\mathbf{x}' = \mathbf{x} + t'\omega_i$ is a position along the ray at distance t', with $\mathbf{x}_s = \mathbf{x} + t\omega_i$ and S the *source term*, accounting for *in-scattered* or *emitted* light.

We base our computations on the assumption of *single scattering*, i. e., we only consider light paths starting at the main light source and being scattered towards the camera at some point inside the medium after none or exactly one prior bounce from a surface. In the following, we refer to the zero and one bounce cases by *direct* and *indirect single scattering*, respectively. Also, we assume that the medium itself does not emit light. Our simplified source term then becomes

$$S(\mathbf{x}', -\omega_i) = \sigma_s(\mathbf{x}') \int_{\mathcal{S}^2} \rho(\mathbf{x}', -\omega', -\omega_i) L_{d/1}(\mathbf{x}', \omega') \, d\omega',$$

where $L_{d/1}$ only considers direct or one-bounce indirect light and we are integrating over all directions ω' of the unit sphere $\mathcal{S}^2$. Here, σ_s is the *scattering coefficient*, determing the probability of scattering and ρ is the so-called *phase function*, describing the angular distribution of the scattering depending on the angle between $-\omega'$ and $-\omega_i$. We assume that absorption and scattering probability are both linearly dependent on the *density* of the medium which is given as a scalar field over a three-dimensional domain.

For a more extensive theoretical background, we refer to survey papers like the one by Max [Max95] or the book chapter by Pharr [PH10] and will only describe the actual computations we perform (Sec. 4.2).

3.2. Deep screen space

In the following, we outline the deep screen space pipeline which we use to transport light from the surfaces in the scene into the medium. For a detailed description, we refer to [NRS14]. The scene primitives are first tessellated into a view-dependent surfel representation which is then splatted onto a multi-resolution deferred shading buffer to compute the effect of the surfels on their surroundings (e. g., transport of indirect light). Lastly, the multiple resolutions are combined to a final image.

Tessellation of the scene into a surfel cloud The first step comprises turning the input triangle mesh of the scene into a surfel cloud. A surfel is an oriented disk defined by its position, normal and radius [PZVBG00]. The surfel cloud serves as an approximation of the original scene geometry using a uniform primitive type which significantly simplifies shading computations.

To generate surfels from triangles efficiently, hardware tessellation in "point mode" is used [BBH13]. This is done in such a way that all surfels have approximately equal size in screen space, explicitly including triangles seen under grazing angles as well as those which are back-facing.

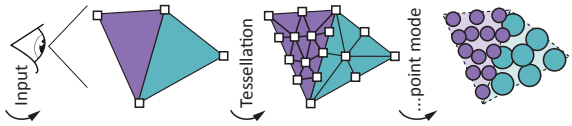

Figure 2: *Triangles are "surfelized" depending on their distance to the camera, leveraging hardware tessellation in point mode. Vertices are depicted as rectangles, surfels as disks.*

Splatting the point cloud onto a framebuffer To compute the actual effect, splatting is used to scatter the shading contribution from each surfel to multiple pixels. For this, a special framebuffer layout based on interleaved sampling [SIMP06] is employed. The framebuffer is organized as an array of l_{max} textures where each texture corresponds to a different image resolution level. On level $l \geq 0$, the pixels of the original image are partitioned into $2^l \times 2^l$ small "sub-buffers": Neighborhoods consisting of $2^l \times 2^l$ pixels are taken and each sub-buffer is assigned one of the pixels at random. The pixel takes the same relative position in the sub-buffer that the neighborhood had in the original image (Fig. 3).

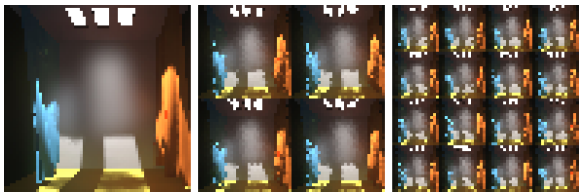

Figure 3: *The first three levels of a possible framebuffer layout* (left to right). *The total number of pixels on each level is identical, however the original pixels are distributed among more and more sub-images as the level number increases. Consequently, same-sized screen areas correspond to increasing world space areas, but with fewer samples.*

The surfels' shading contribution to disjoint, increasingly large (world space) *shells* around their centers is computed on increasing levels. To this end, for each surfel, a shell is splatted into one random sub-buffer on each level. Point-primitives are used to tightly bound the shells in the framebuffer.

Drawing a quad of size $n \times n$ into a random sub-buffer of image level l will invoke the fragment shader for a random subset of (at least) n^2 pixels in a $2^l n \times 2^l n$ neighborhood of the original framebuffer. Thus, with increasing level, the method achieves a gradual decrease of precision in relating a surfel with surfaces of increasing distance from it. Every detail of the scene still can receive shading from the surfel, only the probability decreases. To compensate for the fact that each pixel can only be affected by $1/(2^l \cdot 2^l)$ of all surfels on level l, the surfels' areas are scaled accordingly. Effectively, each enlarged surfel serves as approximation for $2^l \cdot 2^l$ original surfels.

The size of the shells (and consequently of the splats) depends on the concrete effect being computed. It is determined by a function `getMaxDist`, returning the world space distance in which the effect of a particular surfel becomes smaller than a user-defined threshold ε.

Drawing a splat will invoke the fragment shader which is passed the relevant attributes of the surfel as well as the inner and outer radius of the shell associated with the splat. The latter two are necessary to check if world space positions associated with a fragment are actually inside the shell for which the effect is to be computed. The fragments are blended suitably to sum up the effect of all splats.

The formula to compute the splat size and the actual effect computations in the fragment shader depend on the specific effect. We discuss them for the case of indirect lighting of volumes in Sec. 4.2.

Reconstructing the final image After splatting, the level textures are still partitioned into sub-buffer grids. They are "unshuffled" and the levels are blurred separately to eliminate noise due to the random sub-sampling. Finally, the contribution from different levels is summed up in an appropriate way.

4. Volume shell splatting

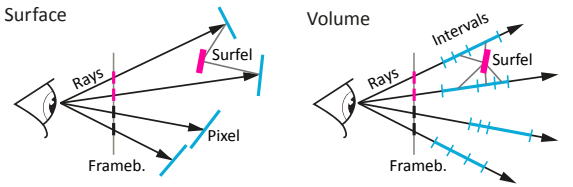

Figure 4: *In contrast to surface-to-surface shading where surfels influence nearby pixels* (left)*, for the surface-to-volume case* (right) *surfels influence ray intervals at nearby pixels.*

A slow but correct approach to compute indirect lighting from a surfel cloud onto a volume would be to iterate for each surfel over all points on all camera rays and compute the contribution to the sensor in a cubic iteration. We will reduce this iteration to a linear traversal of ray segments, allowing for a parallel hierarchical evaluation with guaranteed error bounds at interactive rates as follows:

Transmittance interval map At every pixel we need to iterate over all points on a ray through that pixel to compute the contribution from a surfel to the sensor. In practice, the number of points is in the order of hundreds and a pixel receives splats from many surfels. We significantly reduce this iteration by caching relevant information from the ray traversal in a transmittance interval map. This map decomposes the ray into a low number of intervals which are expected to have similar contribution to the sensor. When splatting, instead of traversing the ray, only the intervals are traversed. The transmittance interval map is explained in Sec. 4.1.

Splatting Second, we exploit that the screen space area of *rays* affected by a surfel can be tightly bound when giving an error bound ε in the same way as the screen space area of surfaces affected by a surfel was bound previously (Fig. 4). This bounding avoids that each surfel has to iterate all rays (pixels) and instead only iterates over some rays (pixels). The splatting is explained in Sec. 4.2.

4.1. Transmittance interval map

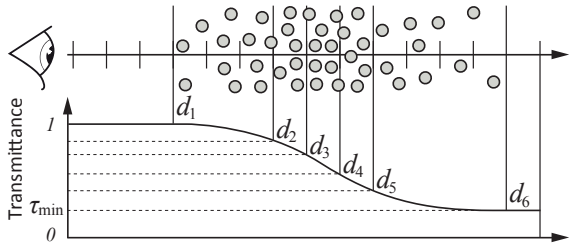

Figure 5: *Transmittance interval map at one pixel. Intervals between the distances d_1 to d_n equally divide the transmittance range between the maximum* 1 *and the minimum* $\tau_{\min}$.

Our strategy is to split the ray under each pixel into *intervals* of equal transmittance to the sensor. The process is independent for each pixel and will be described for a pixel p in the following. We parameterize the n ray intervals under pixel p by a sequence $\{d_1,\dots,d_n \in \mathbb{R}\}$ of distance intervals. The 3D positions $\{\mathbf{x}_1,\dots,\mathbf{x}_n \in \mathbb{R}^3\}$ of the ray interval start and end points can be computed from those distances, the view parameters and the pixel coord p. Let $\tau(d)$ be the transmittance at distance d along the ray as defined in Sec. 3.1, which can be computed iteratively by front-to-back ray-marching [Max95]. First, the largest distance d_1 such that $\tau(d_1) = 1$ and the minimum of τ which we denote as $\tau_{\min}$ are found. We refer to the distance where $\tau_{\min}$ is achieved by d_n. This is done by ray-marching the volume front-to-back once, using a maximum of $n_{\max}$ steps, potentially stopping at an intersection point with a solid surface. Whether and where the latter occurs is derived from a deferred shading position buffer. Finally, a second ray-marching pass starts at distance d_1 and stores the smallest sampling distances d_2 to d_n such that $\tau(d_i) < 1 - i(1-\tau_{\min})/(n-1)$ as shown in Fig. 5. As τ is approximated by accumulation with finite step sizes, the inversion is also approximate i. e., the fraction of transmittance is also only almost equal. (Put differently, the vertical distance of the dotted lines in Fig. 5 is not fully equal.)

Besides the intervals, we also store the minimal transmittance values $\tau_{\min}$ for all pixels. They are later used during splatting (Sec. 4.2) to efficiently determine the extinction of the in-scattering on the way to the camera.

In the following, we will use the name *transmittance interval map* for the map holding all d_i and the minimal transmittance value $\tau_{\min}$ for each pixel. In contrast to deep shadow maps, which store a mapping from depths to transmittance values, the transmittance interval map stores an inverse mapping from transmittance values to depths.

Jittering To avoid banding artifacts, we jitter the length of the first ray-marching step, which is routinely done in classic methods. Similarly, we add the same random offset to all the transmittance thresholds (determining where to place the samples) per pixel, and use $\tau(d_i) < 1-(i-\xi)(1-\tau_{\min})/(n-1)$, where ξ is a uniform random variable between 0 and 1. Fig. 6 demonstrates the effectiveness of this approach.

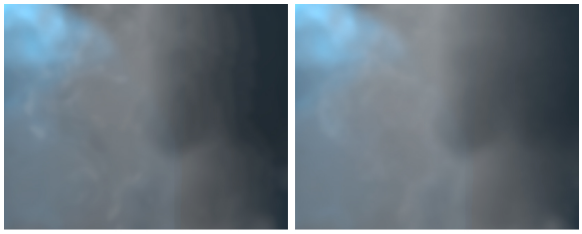

Figure 6: *The relatively large step size we use leads to banding artifacts* (left)*, jittering resolves this problem.*

Encoding During splatting, the transmittance interval map will receive light from surfels several hundred times. For this reason, we encode the transmittance interval map to reduce bandwidth when actually exchanging radiance.

We use $n = 16$ and store the d_i in an unsigned integer texture of $4 \times 32 = 128$ bits per pixel by encoding the number of ray-marching steps between the samples (and from the camera to the first sample) in 8 bit each. Doing this, we assume that the samples are not further than 255 steps apart, which is valid if the medium's density varies only slowly. Given the camera's position and the ray marching step length, we can reconstruct the sampling positions precisely (Fig. 7).

Finally, to compute the in-scattering light at each sampling position, the scattering coefficients of the medium at those points are needed. Since the sampling positions are known, we can pre-fetch the coefficients. For our case, we store the coefficients c_1 to c_n at the respective distances d_1 to d_n by encoding them relative to the maximal coefficient $c_{\max} = \max\{c_1, \ldots, c_n\}$ in 8 bit each. Again, we use a 128 bit texture. The remaining 8 bits are used to encode $c_{\max}$ relative to the maximal scattering coefficient across the whole medium, which we assume to be known.

4.2. Splatting to the sampling positions

To light a volume from a surface, we need to enumerate pairs of deep screen space surfels (representing the surface) and pixels of the transmittance interval map (representing the volume). A simple, but prohibitively slow, solution would be to enumerate all pairs. Instead, we enumerate pairs with a probability proportional to an approximate bound of their light transport. Doing so, strongly-coupled pairs are more likely to be evaluated resulting in more precision, pairs with weak exchange are computed rarely and with less precision.

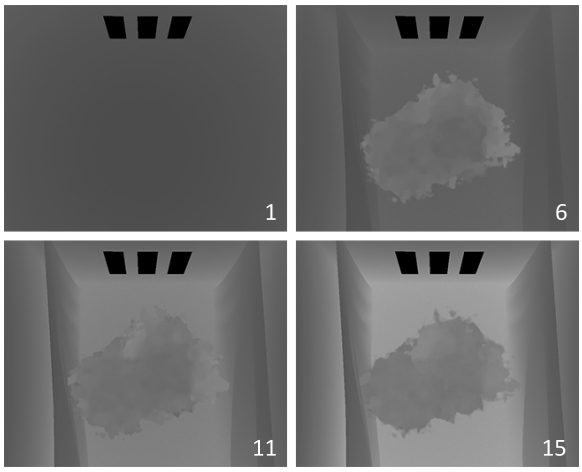

Figure 7: *Four sampling depths from a transmittance interval map. Brighter pixels correspond to larger depth. Note the smaller depth range for the cloud-parts due to the adaptive sampling which concentrates samples in the denser areas.*

Surfel information To compute the light transport from the surfels to the volume, we first need to compute the irradiance at each surfel. We use a cube shadow map for visibility checks with respect to point light sources and simple ray-marching with constant step size to determine the extinction of light between the light source and the surfel.

Bounding As outlined in Sec. 3.2, the size of the splat for a surfel (with possibly increased radius) is determined by a function `getMaxDist` depending on the surfel S and a threshold ε. Given the radiosity B_S and radius r_S of the surfel, we compute the world-space radius of the splat as $r_S \max\left(B_{S,\mathrm{r}}, B_{S,\mathrm{g}}, B_{S,\mathrm{b}}\right)/\varepsilon$, which is based on the assumption of a quadratic fall-off of the radiance received by other points with increasing distance. Note that surfels which are in shadow are discarded automatically as their radiosity amounts to 0. Unfortunately, we cannot easily take the density of the medium around the surfel into account here.

The bound for volumes is naturally less tight, in particular for small splats: When a small splat is drawn to nearby pixels, the potential for overdraw is small. When a small splat is drawn to nearby rays, little overdraw in the image occurs, however, due to the additional depth-dimension, a small splat is likely to have an effect on only a small fraction of the intervals of the ray below the pixel.

Splatting At each pixel p covered by a splat, we are given a surfel S with its position $\mathbf{x}_S$, normal $\mathbf{n}_S$, radius r_S and radiosity B_S as well as the inner and outer radius of the shell associated with the splat. Also, by using three texture lookups, we have access to the transmittance interval map containing d_1 to d_n, the transmittance $\tau_{\min}$, the cached scattering coefficients c_i of the d_i and finally the maximum coefficient $c_{\max}$ with respect to which they have been encoded.

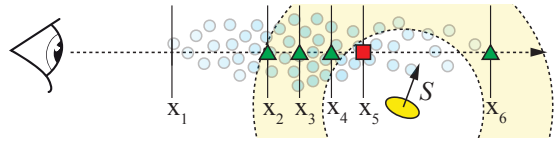

Figure 8: *Only at points inside the shell* (triangles) *associated with the surfel S, the latter's contribution is computed.*

We march over all sampling points $\mathbf{x}_2$ to $\mathbf{x}_n$ corresponding to the d_i, adding up the radiance contributed by S to the pixel p. At each point, we first have to check whether $\mathbf{x}_i$ is inside the shell belonging to the splat (Fig. 8). If this is not the case, we proceed with the next point, otherwise we compute the contribution of the surfel S by

$$\underbrace{L_{\text{in}}(\mathbf{x}_i, \mathbf{x}_S)}_{\text{In-scattering}} \cdot \underbrace{\left(1 - \frac{i\,(1-\tau_{\min})}{n-1}\right)}_{\text{Transmittance}} \cdot \underbrace{(d_i - d_{i-1})}_{\text{Segment length}} .$$

The first term is the in-scattering from the surfel, computed using the looked-up scattering coefficient, the second term corresponds to the transmittance between d_i and the camera (cf. Sec. 4.1) and the third is the length of the ray segment associated with the sample. The in-scattering is computed as

$$L_{\text{in}}(\mathbf{x}_i, \mathbf{x}_S) = \frac{\pi {r_S}^2 \max(\cos\alpha_S, 0)}{\max\left(\|\mathbf{x}_i - \mathbf{x}_S\|_2^2, \delta\right)} B_S \cdot c_i \cdot \rho(\alpha_{\text{scatt}}),$$

where α_S is the angle between the surfel's normal and the vector pointing from the surfel to $\mathbf{x}_i$ and ρ is the phase function depending on the scattering angle α_{scatt} between the vector from $\mathbf{x}_S$ to $\mathbf{x}_i$ and the vector at $\mathbf{x}_i$ pointing towards the camera (Fig. 9). To avoid intensity singularities near the surfels, we clamp the distance $\|\mathbf{x}_i - \mathbf{x}_S\|_2^2$ in the computation of the in-scattering to a minimum of $\delta > 0$.

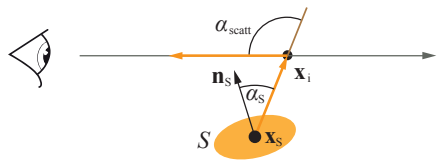

Figure 9: *In-scattering from a surfel S at position* $\mathbf{x}_S$ *with normal* $\mathbf{n}_S$ *to a point* $\mathbf{x}_i$.

Blurring After splatting and unshuffling the array of textures containing the solution for the different shells, we perform a Gaussian blur. We use a Gaussian MIP map over the textures and replace level l with an up-sampled version of level $l+1$ of its MIP map.

4.3. Implementation Details

Direct single scattering To compute the direct single scattering from the main light sources, we perform analogous computations at all pixels, merely replacing the surfel by the direct light source and dropping the restriction to a shell.

Number of ray marching steps As a hint for the necessary number of ray-marching steps when preparing the transmittance interval map, the depth of the wall in our test scene corresponded to roughly 200 steps. The maximal number of steps $n_{\max}$ should cover the whole range to the far plane.

Phase function We used the popular phase function by Henyey and Greenstein [HG41] in our experiments with varying anisotropy settings.

5. Results

Fig. 10 compares results obtained using our method to a reference based on ray-marching and ray-tracing for gathering indirect light. Where not specified otherwise, we used a resolution of 800×600 px and a Nvidia GTX 770 graphics card. The number of levels for the hierarchical framebuffer was constantly set to six with an effect threshold ε of 0.06 and about 14k surfels were used to represent the original scene. The single scattering was linearly scaled to accentuate it.

A notable difference is in brightness. On one hand, our method exhibits increased brightness towards the inside of the medium due to the lack of attenuation between the surfels and the receiving sampling points. On the other hand, the brightness is underestimated over the long range since we clamp contributions less than ε. The sub-sampling in our method leads to noise which makes blurring necessary. This is in particular visible around the boundaries of the medium which are less defined. Still, our result looks plausible while being orders of magnitude faster than the ray-marching approach.

The main parameter influencing quality is the effect threshold ε. Fig. 11 shows how it influences the result and computation times. For lower quality settings (larger values of ε), the cloud becomes darker and its boundary is less defined due to stronger blurring.

Performance Tbl. 1 gives an exemplary performance breakdown for Fig. 10a. As is to be expected, the running time is dominated by the time for splatting. Another costly part is generating the transmittance interval map. We only list the timings for computations related to the indirect lighting of the volume. Besides classic direct lighting of the surfaces,

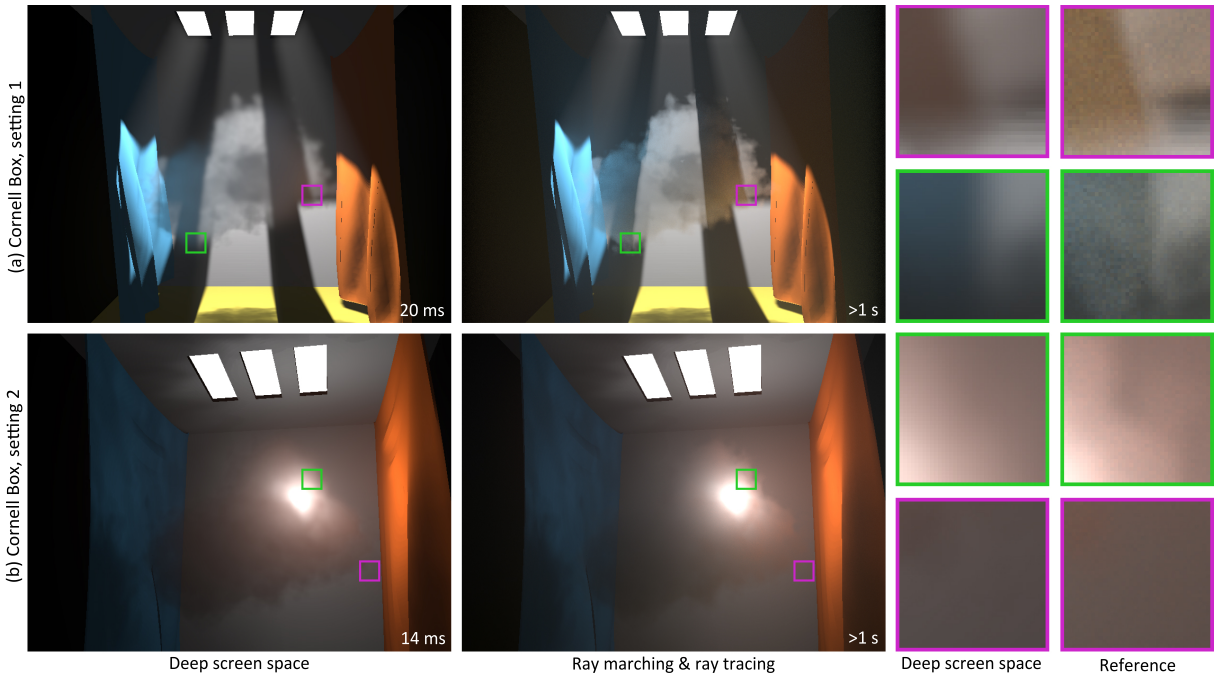

Figure 10: *The first column contains our results; the second shows the reference; the last two columns contain details of our result and the reference, respectively. Please see the video for some animated scenes.*

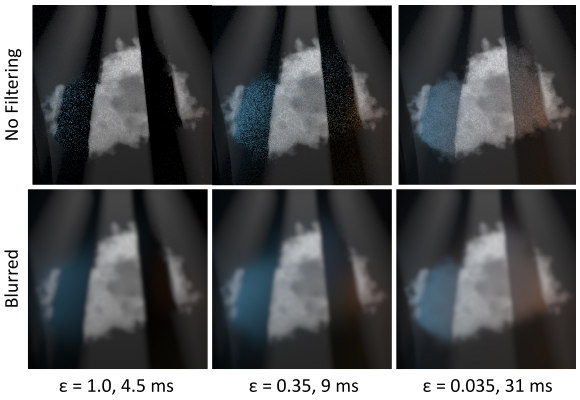

Figure 11: *Crops from the Cornell box scene using different values for epsilon, without* (top) *and with* (bottom) *blurring. Only direct and indirect single scattering are shown. The timings correspond to the splatting for indirect scattering alone as other parts are unaffected by the changed setting.*

Stage		Time
Shadow Map		0.6 ms
Transmittance Interval Map	+	8 ms
Scattering Coefficient Caching	+	1.4 ms
Surfel Cloud Generation	+	3 ms
Splatting	+	20 ms
Unshuffling	+	1 ms
Blurring	+	3 ms
Summing	+	0.4 ms
Indirect Single Scattering	=	37.4 ms

Table 1: *Computation time for different stages.*

our result images also show direct single scattering and absorption of direct light by the medium which typically took us about 1.5 and 3 ms to compute, respectively.

Limitations The main shortcoming of deep screen space is its lack of indirect visibility, i. e., we have no information about the visibility between the surfels and the points receiving shading from them. In particular, the indirect light is not occluded by surfaces or attenuated by the medium.

One theoretical shortcoming of our method is the fact that it does not allow to bound the effect of a surfel onto rays but only onto ray intervals. If a surfel only affects a small number n' of intervals in a ray with $n \gg n'$ intervals, the splatting will still traverse all n ray intervals. In future work, for large n it could be useful to also cull groups of ray intervals.

Our chosen encoding for the volume map using sixteen 8 bit values to fit everything into just one texture is not sufficient if either the density of the medium or the lighting

exhibit high frequencies (e. g., due to rapidly changing visibility). However, nothing prevents from using more samples at the cost of slower computations.

A simplification was to only consider media whose properties do not vary with the wavelength, i. e., monochromatic media. The extension to the more general case only amounts to performing analogous computations for each wavelength, though, obviously at larger computational cost.

6. Conclusions

This paper presented an interactive method to compute light transport from surfaces to volumes in fully dynamic scenes. This was achieved by discretizing the problem into two adapted representations that allow for efficient hierarchical transfer. The first is a deep screen space, represented by a surfel point cloud corresponding to senders of indirect light to the current view. The second is a transmittance interval map, that re-parameterizes the volume and the scene as a collection of ray intervals, also for the current view. Transport from surfaces into the volume can now be achieved by hierarchical splatting. All steps can be performed at interactive rates for dynamic scenes and work without maintaining any pre-computed data structures to be re-used over frames.

Future work will consider more general forms of transport, such as from volumes to surfaces or multiple bounces. We have only demonstrated bounces from diffuse surfaces into the medium, yet specular surfaces using Phong instead of Lambertian surfels are a potential extension.

References

[BBH13] BARÁK T., BITTNER J., HAVRAN V.: Temporally coherent adaptive sampling for imperfect shadow maps. *Comp. Graph. Forum (Proc. EGSR) 32*, 4 (2013), 87–96. 3

[Cha50] CHANDRASEKHAR S.: *Radiative Transfer*. Oxford Univ. Press, 1950. 2

[ENSD12] ENGELHARDT T., NOVÁK J., SCHMIDT T.-W., DACHSBACHER C.: Approximate bias compensation for rendering scenes with heterogeneous participating media. *Comp. Graph. Forum (Proc. Pacific Graphics) 31*, 7 (2012), 2145–54. 2

[ERDS14] ELEK O., RITSCHEL T., DACHSBACHER C., SEIDEL H.-P.: Interactive light scattering with principal-ordinate propagation. In *Proc. Graphics Interface* (2014). 2

[Fat09] FATTAL R.: Participating media illumination using light propagation maps. *ACM Trans. Graph. (TOG) 28*, 1 (2009), 7. 1, 2

[HDI*10] HU W., DONG Z., IHRKE I., GROSCH T., YUAN G., SEIDEL H.-P.: Interactive volume caustics in single-scattering media. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), I3D '10, pp. 109–117. 2

[HG41] HENYEY L. G., GREENSTEIN J. L.: Diffuse radiation in the galaxy. *Astrophysical J 93* (Jan. 1941), 70–83. 6

[JC98] JENSEN H. W., CHRISTENSEN P. H.: Efficient simulation of light transport in scenes with participating media using photon maps. In *Proc. SIGGRAPH* (1998), pp. 311–20. 1

[JSG09] JIMENEZ J., SUNDSTEDT V., GUTIERREZ D.: Screen-space perceptual rendering of human skin. *ACM Trans. App. Perception 6*, 4 (2009), 23. 2

[KD10] KAPLANYAN A., DACHSBACHER C.: Cascaded light propagation volumes for real-time indirect illumination. In *Proc. ACM i3D* (2010), pp. 99–107. 2

[Kel97] KELLER A.: Instant radiosity. In *Proc. SIGGRAPH* (1997), pp. 49–56. 2

[KN01] KIM T.-Y., NEUMANN U.: Opacity shadow maps. In *Proc. EGWR* (2001). 2

[KSE14] KLEHM O., SEIDEL H.-P., EISEMANN E.: Prefiltered single scattering. In *Proc. ACM i3D* (2014), pp. 71–8. 2

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *Proc. SIGGRAPH* (2000), Proc. SIGGRAPH, pp. 385–392. 2

[Max95] MAX N.: Optical models for direct volume rendering. *IEEE Trans. Vis. Comp. Graph. 1*, 2 (1995), 99–108. 3, 4

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses* (2007), pp. 97–121. 1, 2

[MKBVR04] MERTENS T., KAUTZ J., BEKAERT P., VAN REETH F.: A self-shadow algorithm for dynamic hair using density clustering. In *ACM SIGGRAPH 2004 Sketches* (2004), p. 44. 2

[NED11] NOVÁK J., ENGELHARDT T., DACHSBACHER C.: Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *Proc. ACM i3D* (2011), ACM, pp. 119–124. 2

[NNDJ12] NOVÁK J., NOWROUZEZAHRAI D., DACHSBACHER C., JAROSZ W.: Virtual ray lights for rendering scenes with participating media. *ACM Trans. Graph. (Proc. SIGGRAPH) 31*, 4 (2012), 60. 1

[NRS14] NALBACH O., RITSCHEL T., SEIDEL H.-P.: Deep screen space. In *Proc. ACM i3D* (2014), ACM. 1, 2, 3

[PH10] PHARR M., HUMPHREYS G.: *Physically Based Rendering, Second Edition: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., 2010. 2, 3

[PKK00] PAULY M., KOLLIG T., KELLER A.: Metropolis light transport for participating media. 1

[PZVBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proc. SIGGRAPH* (2000), pp. 335–342. 3

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proc. ACM i3D* (2009), ACM, pp. 75–82. 2

[RSK08] RAAB M., SEIBERT D., KELLER A.: Unbiased global illumination with participating media. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, Keller A., Heinrich S., Niederreiter H., (Eds.). 2008, pp. 591–605. 2

[RT87] RUSHMEIER H. E., TORRANCE K. E.: The zonal method for calculating light intensities in the presence of a participating medium. In *ACM SIGGRAPH Computer Graphics* (1987), vol. 21, pp. 293–302. 1

[SIMP06] SEGOVIA B., IEHL J. C., MITANCHEY R., PÉROCHE B.: Non-interleaved deferred shading of interleaved sample patterns. In *Proc. Graphics Hardware* (2006), pp. 53–60. 3

[Sta95] STAM J.: Multiple scattering as a diffusion process. In *Rendering Techniques*. 1995, pp. 41–50. 1

[WKSD13] WEBER C., KAPLANYAN A., STAMMINGER M., DACHSBACHER C.: Interactive direct volume rendering with many-light methods and transmittance caching. In *Proc. VMV* (2013), pp. 195–202. 2

[YK08] YUKSEL C., KEYSER J.: Deep opacity maps. In *Comp. Graph. Forum* (2008), vol. 27, pp. 675–80. 2