Image Plane Sweep Volume Illumination

Erik Sundén, Anders Ynnerman, *Member, IEEE*, and Timo Ropinski, *Member, IEEE*

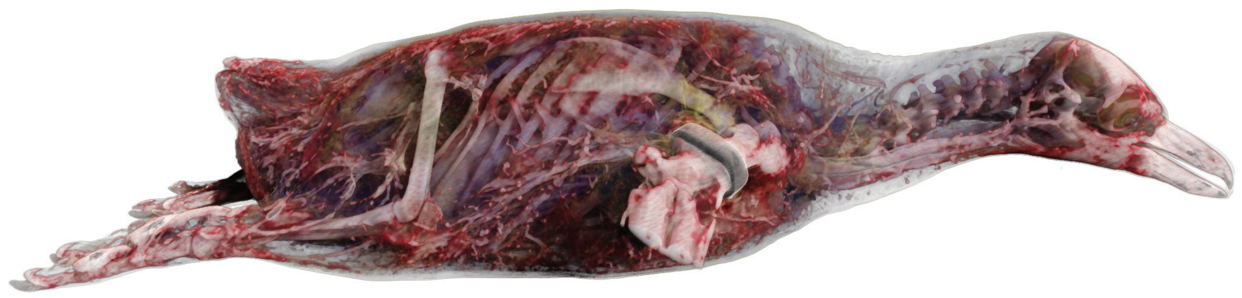


Fig. 1. A CT scan of a penguin ($512 \times 512 \times 1146$ voxels) rendered using image plane sweep volume illumination. Local and global illumination effects are integrated, yielding in high quality visualizations of opaque as well as transparent materials.

Abstract—In recent years, many volumetric illumination models have been proposed, which have the potential to simulate advanced lighting effects and thus support improved image comprehension. Although volume ray-casting is widely accepted as the volume rendering technique which achieves the highest image quality, so far no volumetric illumination algorithm has been designed to be directly incorporated into the ray-casting process. In this paper we propose *image plane sweep volume illumination* (IPSVI), which allows the integration of advanced illumination effects into a GPU-based volume ray-caster by exploiting the plane sweep paradigm. Thus, we are able to reduce the problem complexity and achieve interactive frame rates, while supporting scattering as well as shadowing. Since all illumination computations are performed directly within a single rendering pass, IPSVI does not require any preprocessing nor does it need to store intermediate results within an illumination volume. It therefore has a significantly lower memory footprint than other techniques. This makes IPSVI directly applicable to large data sets. Furthermore, the integration into a GPU-based ray-caster allows for high image quality as well as improved rendering performance by exploiting early ray termination. This paper discusses the theory behind IPSVI, describes its implementation, demonstrates its visual results and provides performance measurements.

Index Terms—Interactive volume rendering, GPU-based ray-casting, Advanced illumination.

1 Introduction

The benefits of advanced illumination techniques for interactive volume rendering are numerous and have been demonstrated in recent user studies [30, 41, 17]. Accordingly, many researchers have addressed the area of advanced volume illumination while focusing on interactive techniques [28]. The approaches tackling this area can be roughly divided into two groups. First, approaches exploiting volume preprocessing, which can be done either on the CPU [32] or, for increased performance, on the GPU [14]. Techniques belonging to this group precompute the illumination information and store it in an additional volume. While preprocessing can, in principle, be combined with any volume rendering algorithm, another widely used strategy directly exploits the benefits of the slice-based volume rendering paradigm [4]. In contrast to ray-casting approaches, slice-based rendering inherently synchronizes the *ray front* and thus reduces the complexity of the illumination computation when focusing on directional effects [12, 13, 34, 40]. While all methods belonging to the two mentioned groups support advanced illumination at interactive frame rates, they also incorporate the drawbacks of the respective group. For instance, the preprocessing techniques are not applicable to large volumetric data sets since the precomputed illumination volume would consume too much extra graphics memory. While this is not a drawback of the slice-based approaches, they have the downside that they are tightly bound to the slice-based rendering paradigm, which is known to result in inferior image quality as compared to ray-casting based techniques [37]. To achieve high quality results with slice-based rendering, a rather large number of slices is necessary which directly impacts rendering performance. Furthermore, the slices need a sufficient bit depth to allow high precision compositing during the integration. To incorporate advanced illumination effects, the memory footprint is further enlarged since an additional slice serving as light buffer is crucial. In addition to their superior image quality, ray-casting based techniques also grant more control over the ray marching, allowing adaptive sampling, early ray termination and empty space skipping. Additionally, some of the slice-based volume illumination techniques have restrictions on the light source positioning, for example, either head-lights only [34] or lights in the viewer's hemisphere [40] are supported. Until now no interactive illumination technique has been proposed which directly integrates global illumination effects into an interactive ray-casting based volume renderer without requiring precomputation of an illumination volume.

In this paper we introduce *image plane sweep volume illumination* (IPSVI), which enables shadowing and scattering effects within GPU-based volume ray-casters. Expressing the illumination computation as an iterative process allows us to exploit the plane sweep paradigm [25] in order to achieve advanced illumination effects. The plane sweep paradigm has often been applied in computational geometry to reduce problem complexity by transforming a problem into a sequence of problems of lower dimensionality, which are then solved sequentially. In our case we use a sweep line, which sweeps over the image plane to introduce the desired interactive illumination effects. During

• *Erik Sundén, Anders Ynnerman and Timo Ropinski are with the Scientific Visualization Group, Linköping University, Sweden. E-mail: {erik.sunden|anders.ynnerman|timo.ropinski}@liu.se.*

Manuscript received 31 March 2011; accepted 1 August 2011; posted online 23 October 2011; mailed on 14 October 2011.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org.

the sweep we can keep the memory footprint low since we only need to store the state of the sweep line as opposed to the entire volume illumination information. Thus, we only need one additional 2D texture which is interactively updated during rendering, by exploiting the features of state-of-the-art graphics hardware. This makes the additional memory requirements widely independent of the data set size, and IPSVI the first method of its kind, which is applicable to large volumetric data sets (see Figure 1), and the multimodal data sets which frequently arise in medical imaging.

In the remainder of this paper, we will first discuss relevant research in Section 2, before introducing the concepts underlying IPSVI in Section 3. All relevant details needed for the implementation of the algorithm are explained in Section 4. In Section 6 we will discuss the visual results as well as the performance achievable with the presented method and discuss our findings regarding certain parameter choices. Finally, the paper will conclude in Section 7 by summarizing our contribution and discussing future work.

2 Related Work

Volume illumination. In the past, different illumination techniques have been proposed in the area of interactive volume rendering. Synchronization is one of the main issues when dealing with on-the-fly volume illumination. Therefore, many of the proposed techniques are inherently bound to a specific rendering paradigm. The half angle slicing technique, presented by Kniss et al. [12, 13], was the first one allowing interactive volumetric illumination. It is based on the slice-based rendering paradigm [5] and synchronization is achieved by incorporating the light source position when selecting the main slicing direction. Thus an intermediate light buffer can be synchronized with the eye buffer to ensure that the relevant illumination information has been already computed. More recently occlusion-based shading models have been proposed, based on the slice-based rendering paradigm [34, 40]. Schott et al. [34] assume the existence of a headlight and can thus incorporate occlusion information when slicing from front to back. This light source constrain has been loosened by Šoltészová et al. [40], who introduce the use of warped filter kernels in order to support light sources located anywhere within the hemisphere pointing towards the viewer. Moloney et al. [23] have also recently proposed how to exploit sort first distributions in parallel volume rendering, while allowing advanced illumination using half angle slicing. Besides the slice-based techniques, Zhang et al. [42, 43] have demonstrate how to add shadows into sheet-based splatting techniques.

Given the fact, that ray-casting can be considered as the volume rendering technique producing the highest image quality [37], it is remarkable that only a few techniques for interactive advanced volume illumination have focused on this paradigm. Rezk-Salama [27] has proposed a fast Monte-Carlo-based algorithm which incorporates shadowing and scattering effects into a GPU-based volume ray-caster. However, the performance impact can still be considered as high and the technique is optimized by restricting scattering events to a limited set of isosurfaces. Ropinski et al. [31] present an overview of how to add shadows to GPU-based ray-casting. In fact many of the discussed techniques can also be combined with other rendering paradigms. As the most promising approach, they have identified the deep shadow mapping technique [19], which also allows the incorporation of semi-transparent shadows. Hadwiger et al. [8] have demonstrated how to exploit GPU capabilities in order to apply deep shadow maps efficiently. However, the deep shadow mapping paradigm is inherently based on an approximation of the shadowing function, which is stored in an illumination volume, and furthermore user-specified bias values are required. Alternatively, also ray-based techniques have been developed in order to support advanced volume illumination. Hernell et al. [9] and Ljung et al. [18] proposed a ray-based technique, which interactively simulates ambient occlusion by incorporating neighboring structures. A similar idea is the foundation of the piecewise linear integration technique proposed by Hernell et al. [10]. By assuming that distant structures have only little illumination impact, they propose an algorithm which is based on a low-resolution grid and thus allows faster ray marching. All of these ray-based techniques have constraints, which the technique proposed in this paper is not subject to. Thus, we do not restrict the illumination effects to local features or particular surfaces, allow semi-transparent shadows and require no user-specified bias or an illumination volume.

Besides the approaches bound to a specific rendering paradigm, more general approaches are also available. These often exploit a pre-computation stage and store the resulting illumination-related information in an additional volume. Behrens and Ratering [2] were the first to use illumination volumes where they store a precomputed illumination value for each voxel. Ropinski et al. [30] have also adapted this approach, but instead recompute the volume on the GPU and loosen the constraints regarding the light source. Qiu et al. [26] have proposed a volumetric illumination method which supports multiple scattering by representing a diffuse photon volume and a density volume as FCC lattices. Recently spherical harmonic lighting has also been considered in the area of interactive volume rendering. Ritschel [29] was the first to apply this concept to volume rendering. Similar in spirit to the piecewise-linear integration technique [10] he exploits a level-of-detail sampling scheme when applying ray marching, to recompute the spherical harmonic coefficients. Lindemann and Ropinski [16] applied a similar strategy and were able to simulate advanced material properties in volume rendering. More recently Kronander et al. [14] presented an improved approach which recomputes the spherical harmonic coefficients even faster. While all these techniques have demonstrated their potential to generate impressive results, they all need to store the precomputed information, and are thus not applicable when rendering large volumetric data sets. Furthermore, in some cases the pre-processing time hinders interactive transfer function specification.

Being a less expensive approximation of global illumination, more general ambient occlusion techniques [44] have also been applied in the area of volume rendering than the techniques presented by Hernell et al. [9]. The first occurrences of this concept had the goal to improve the spatial comprehension of isosurfaces [1, 24, 38]. Ropinski et al. [32] exploit an expensive pre-processing which involves computation and clustering of local histograms, which could have been accelerated later on [22]. Diaz et al. [11] have adopted screen-space ambient occlusion techniques within their vicinity occlusion map approach. A more general adaption has been proposed by Ruiz et al. [33]. Their obscurance-based method is targeted towards illustrative purposes.

Sweep-based rendering. Sweeping is an algorithmic paradigm, that has been introduced in the field of computational geometry [25]. Depending on the problem space, plane-sweep and space-sweep techniques are distinguished. In the 2D plane-sweep case, the problem space is a plane which is swept with a line. In the 3D case, the 3D problem space is swept by a plane. Sweeping sequentializes computations by only considering events encountered by the sweep structure. Thus, an n-dimensional static problem is transformed into an $(n-1)$-dimensional dynamic problem. While the classical slice rendering [5] can be considered as a space-sweep algorithm, other volume rendering techniques make more explicit use of the sweep paradigm. In the following, we only focus on those techniques which have been directly linked to the sweeping paradigm. Giertsen applied the sweeping paradigm to allow a volumetric visualization of sparse irregular grids [7]. Thus, he was able to transfer the static 3D ray-casting problem into a dynamic 2D cell sorting problem, which allows efficient rendering. Bitter and Kaufman [3] present ray-slice-sweeping, a slice-parallel volume ray-casting technique. The volume is swept in front-to-back order, whereby volumetric slices are processed before compositing is performed. Silva and Mitchell [36] have exploited the sweeping paradigm to render disconnected and non-convex irregular grids. Farias et al. [6] also propose a sweeping algorithm for the rendering of unstructured volume data. They sweep the data front-to-back and project the faces of each occurring cell. More recently the sweeping paradigm has been used to enhance rendering of dynamic height fields [39]. To compute the occlusion of dynamic height fields the height field is swept in parallel along azimuthal directions for which the occlusion has to be determined. Due to the sweep-based dimensionality reduction the occlusion information can thus be computed very efficiently. While all these approaches convincingly demonstrate the benefits of the sweeping paradigm in rendering, to the authors knowledge it has not been previously exploited in order to achieve advanced volume illumination through ray synchronization.

3 Image Plane Sweep Volume Illumination

Max [20] emphasizes the importance of advanced optical models in the area of volume rendering. Although the theoretical concepts behind these advanced illumination models are well understood, a practical realization which allows interactive visualization is still demanding. Thus, accurate solutions of the physical equations of light scattering are still too expensive to be computed in real-time. Nevertheless, the techniques proposed in past years already allow convincing illumination effects by approximating accurate solutions. As already stated above, all these techniques, independent of whether they are ray-casting- or slice-based, have to deal with the synchronization problem inherent to advanced volume illumination, i. e. samples scattering light onto the current sample need to be computed beforehand. Supporting the desired synchronization directly within a volume ray-caster is more demanding, since we do not have a unique *ray-front* as is the case with slice-based volume rendering. However, when considering the theoretical formulation of advanced illumination within the context of volume rendering, it becomes clear that synchronization can also be achieved within volume ray-casters.

To clarify this, we briefly review the concepts how advanced light effects can be incorporated into the volume rendering integral. The following can only be considered as a brief introduction, a more detailed explanation of the concepts behind the volume rendering integral and the incorporation of advanced illumination effect is given by Max [20] as well as Max and Chen [21]. The standard volume rendering integral which only considers emission and absorption can be used to compute the intensity I at the eye point s_e for the direction ω_o as follows:

$$I(s_e,\omega_o) = T(s_b,s_e)\cdot I(s_b) + \int_{s_b}^{s_e} T(s',s_e)\cdot\tau(s')\cdot c_e(s')ds', \qquad (1)$$

where s_b is the background position and $I(s_b)$ the intensity at s_b. The integral itself considers all samples s' between s_b and s_e to compute their emissive contribution $c_e(s')$, which is absorbed based on the extinction coefficient $\tau(s')$. $T(s_1,s_2) = exp(-\int_{s_1}^{s_2}\tau(s)ds)$ is the probability that a ray does not hit a particle between s_1 and s_2, i. e., the transparency. While Equation 1 only supports local illumination effects, it can be rewritten to incorporate also scattering. Max and Chen present reformulations that either consider only single scattering with a single light direction, or multiple scattering resulting from a rather high albedo [21]. In IPSVI, we combine both models with the goal to have shadowing effects of higher frequency due to the single scattering approximation, as well as to simulate more diffuse scattering effects for homogeneous regions. To do so, we substitute $c_e(s)$ with the sum of the emissive omnidirectional contribution and the light scattered at s in direction ω_o:

$$\begin{aligned} g(s,\omega_o) = c_e(s) + (1-a(s))\cdot p(\omega_l,\omega_o,s)\cdot I_{ss}(s,\omega_l) + \\ a(s)\cdot\int_\Omega p(\omega_i,\omega_o,s)\cdot I_{ms}(s,\omega_i)d\omega_i. \end{aligned} \qquad (2)$$

Here, ω_o is again the outgoing light direction for the light leaving the current sample and usually set based on the viewing ray. ω_l is the principal light direction and ω_i are all directions from which light may hit a sample. $a(s)$ is the albedo at sample s, which is the probability that light is scattered rather than absorbed at s. p is the phase function and $I_{ss}(s,\omega_l)$ describes the single scattering contribution of the light intensity reaching s from direction ω_l. In contrast, $I_{ms}(s,\omega_i)$ describes the multiple scattering contribution of the light intensity reaching s from direction ω_i. We use the albedo $a(s)$ in order to interpolate between the single and the multiple scattering contribution. Hence, when the albedo $a(s)$ is low, we consider mainly single scattering events. We have further chosen to make the albedo as well as the phase function

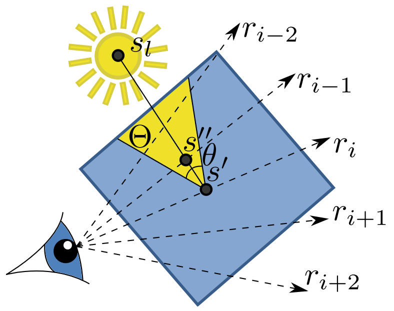


Fig. 2. When considering single scattering only, the scattering contribution influencing the illumination at sample s' only depends on those samples lying on the ray between s' and the light source position s_l. When multiple scattering is considered, the contributing samples lie in the cone Θ.

dependent on the actual sample s to allow a richer set of effects. Similar to the formulation presented by Max and Chen [21], the single scattering contribution can be written as:

$$I_{ss}(s,\omega_l) = T(s_l,s)\cdot L + \int_{s_l}^{s} T(s',s)\cdot\tau(s')\cdot c_e(s')ds', \qquad (3)$$

where s_l is the sample position coinciding with the light source and L is the initial light intensity. In contrast, multiple scattering has to be considered when $a(s)$ is higher in order to compute $I_{ms}(s,\omega_i)$. To simplify the computation of multiple scattering, it is often assumed that a very high albedo is present, such that multiple scattering events are the predominant factor in the illumination computation, which results in a diffusion-like process [21]. We also make this assumption and model multiple scattering as:

$$I_{ms}(s,\omega_i) = \nabla_{diff}\int_{s_b}^{s}\frac{1}{|s'-s|}\cdot c_e(s')ds'. \qquad (4)$$

Here, ∇_{diff} represents the diffusion approximation and $\frac{1}{|s'-s|}$ results in a weighting, such that more distant samples have less influence. In practice, multiple scattering is often simulated by applying low-pass convolution kernels to the neighborhood of the current sample [13]. Nevertheless, computing the convolution over the neighborhood of each processed sample is still computationally too expensive to be performed in real-time. Therefore, the assumption is often made that a forward scattering phase function is present. Several authors [12, 34, 40] show that this is a valid assumption which leads to convincingly realistic results. We also make this simplification, and assume that only forward scattering phase functions are present which supports an efficient computation of Equation 2. When making this assumption, the integral in Equation 2 does not integrate over the whole sphere Ω, but only over a cone-shaped region Θ, which is specified through the cone angle θ (see Figure 2). This simplification reduces computing complexity drastically, since scattering becomes directed and thus only samples between the current sample s' and s_l need to be considered during illumination computation. For the single scattering case, the dependent samples for the current sample s' are also illustrated in Figure 2. It can be seen that all relevant samples lie on the light ray between s' and s_l. A similar observation holds for the samples contributing to the multiple scattering, they all lie in the region of the volume oriented towards s_l from s'. The same relation as between s' and all contributing samples towards the light source holds also for all other samples s'' along the light ray. Thus, we are able to modify Equation 3, such that we can substitute the existing integral with two integrals, one for the illumination calculation between s_l and s'' and one for the illumination calculation between s'' and s':

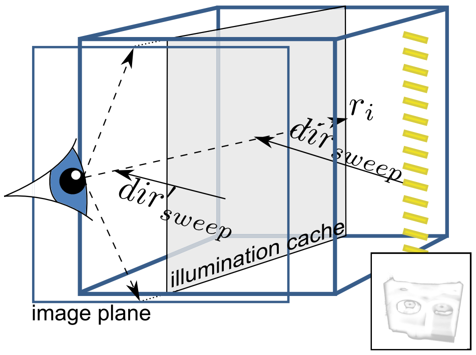


Fig. 3. The sweeping paradigm is exploited to iteratively compute illumination information. By performing a line-sweep in image space, a plane-shaped illumination cache is *fanned* through the volume. The illumination cache captures the relevant illumination information between the currently rendered line and the light source (see inset).

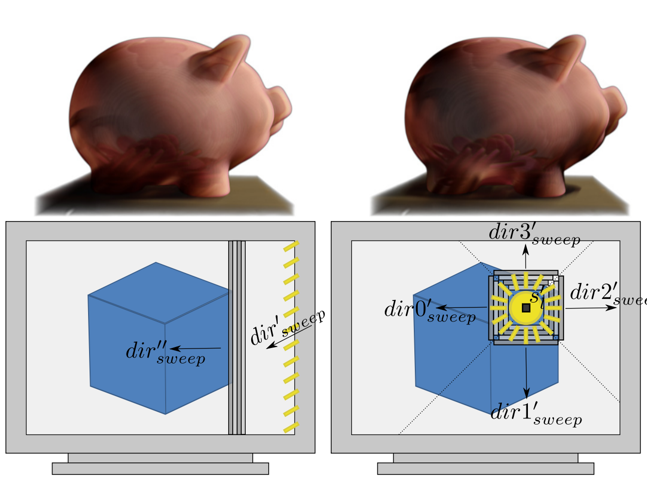


Fig. 4. Sweeping is performed based on the light source to be used. When simulating directional light sources, a single sweeping direction (left) is used, while four sweeping directions are used to simulate point light sources (right).

$$I'_{ss}(s',\omega_i) = T(s_l,s')\cdot L + \int_{s''}^{s'} T(s_i,s')\cdot\tau(s_i)\cdot c_e(s_i)ds_i + \int_{s_l}^{s''} T(s_i,s')\cdot\tau(s_i)\cdot c_e(s_i)ds_i. \tag{5}$$

Splitting the integration into these two parts allows us to compute the illumination iteratively, as it is similarly done in recent slice-based volume rendering techniques [12, 34, 40]. The only term avoiding the iterative computation based on Equation 5 is $T(s_i,s')$ in the second integral, since it would require us to have knowledge about the structures between s'' and s' when solving the second integral. However, when using back-to-front compositing this is not a practical issue, since the compositing is done iteratively anyway. A similar splitting as in Equation 5 can be found for the multiple scattering contribution:

$$I'_{ms}(s',\omega_i) = \nabla_{diff}\int_{s''}^{s'} \frac{1}{|s_i-s|}\cdot c_e(s_i)ds_i + \nabla_{diff}\int_{s_b}^{s''} \frac{1}{|s_i-s|}\cdot c_e(s_i)ds_i. \tag{6}$$

While this splitting also allows to compute the multiple scattering iteratively, as done in the slice-based approaches, we can in contrast directly exploit the ray-casting paradigm. This is also illustrated in Figure 2, where the different rays cast through the volume are annotated as $r_{i-2}...r_{i+2}$. As can be seen in the figure, the illumination at all samples of r_i only depends on samples lying on rays closer to the light source. Thus, when we are able to synchronize ray marching, such that r_{i-1} is always cast before r_i is cast, we can reduce the computation at each sample to the first integrals in Equation 5 and Equation 6 instead of the whole integral as in Equation 3 and Equation 4. As depicted by the distance between s' and s'' as compared to the distance between s'' and s_b this results in a vast speedup.

To achieve the synchronized ray computation, such that r_{i-1} is computed before r_i, we exploit a modified sweeping paradigm [25]. The idea behind this approach is, for a directional light source, illustrated in Figure 3. We denote the sweep direction in image space by dir'_{sweep} and in volume space as dir_{sweep}. As can be seen in Figure 3, dir'_{sweep} is the projection of dir_{sweep} onto the image plane. Since we want to iteratively compute the influence of structures when moving further away from the light source, the sweep direction dir_{sweep} corresponds to the main light direction in volume space. By projecting dir_{sweep} into image space, we obtain the direction to which the sweep-lines should be perpendicular. When rendering, we process the sweep-lines starting in the direction where the light comes from, such that all rays cast through one sweep-line form the according sweep-plane.

We are able to support illumination effects of both directional as well as point light sources, by performing one or four passes of line rendering in order to realize the sweeping, as seen in Figure 4. As can be seen, all rays cast through the same line in screen space are lying in one plane in world space. In the 2D projection shown in Figure 2 this plane would be associated with r_i. Thus, depending on the coordinate frame under consideration, IPSVI can be considered either as a line-sweep algorithm operating in image space, or a plane-sweep algorithm operating in world space. In the latter case it can be considered as modified, since the sweep planes are not parallel, but form a fan-like structure. To store the illumination information accumulated along all light rays between the samples of a specific sweep plane, we use an illumination cache. This illumination cache needs to cover the cross section of the volume, which is formed by all rays cast through one line in the image plane. To be able to capture the relevant illumination information, the illumination cache is updated continuously during the sweep. An example of an illumination cache is shown in the inset in Figure 3. It is generated by iteratively solving the second integral in Equation 5 and Equation 6. We will discuss how to compute the coordinates for reading from and writing into the illumination cache in Section 4. Nevertheless, we would already like to point out here, that in any case we do not require any preprocessing and the illumination cache is represented by a 2D image, rather than a 3D volume as it is the case in other volumetric illumination techniques. Thus, IPSVI has a very small memory footprint and can even be applied to large volumetric data sets.

4 Technical Realization

One benefit of the proposed illumination model is its easy implementation. It can be integrated with only little effort into existing volume ray-casting systems and does not require architectural changes. Our technical realization is based on two conceptual steps, the line sweeping performed on the CPU and the actual ray-casting performed along with the update of the illumination cache on the GPU. While in standard GPU-based ray-casters the shader doing the ray marching is triggered by rendering a screen-aligned quad, we trigger the ray-caster line by line to allow the sweeping, such that when a line is drawn it is used to trigger the ray-casting on pixels contained within the line. Thus, when rendering one line, we know that all lines closer to the light source have already been processed. During the processing of each line we iteratively update the illumination cache, to accumulate and store the visibility as seen from the light source, which is further elaborated in Subsection 4.3. In the following subsections we discuss how to determine the line sweeping direction, perform line synchronization and ray marching, before discussing special cases.

4.1 Line Sweeping

The line sweeping is determined based on the type of light source. In cases where we want to simulate directional lighting we sweep only in one direction (see Figure 4 (left)), and in cases where we want to simulate a point light source, we sweep into four directions starting at the point light source position in image space (see Figure 4 (right)).

Directional lights. To determine the sweeping direction for directional light sources, we project the light direction into image space coordinates. Based on this direction, we are able to derive the sweeping start point, which lies at the screen border close to the light source, as well as the sweeping direction (see Figure 4 (left)). To facilitate an easy implementation in our case, we sweep either horizontally or vertically over the image space. When using arbitrarily oriented lines, the realization would be more complex, since overlap between adjacent lines needs to be avoided. Therefore, it would be essential to exactly know how arbitrarily oriented lines are rasterized. Using horizontal and vertical lines does not require this knowledge and is therefore independent of the underlying system. Therefore, we do not directly use the sweeping direction dir'_{sweep} (see Figure 3), but derive a vertical or horizontal sweeping direction dir''_{sweep}, such that the angle between dir'_{sweep} and dir''_{sweep} is minimized.

Point lights. The sweeping for point light sources is slightly more complex, since its propagation is approximated by four sweep passes as illustrated in Figure 4 (right). To allow light propagation into all directions from the projection of the light source s'_l, we process lines in the four directions $dir0'_{sweep}...dir3'_{sweep}$. Each sweeping originates from s'_l and is propagated either along the positive/negative x or y direction of the screen. Thus, the image plane is divided into four zones, where the borders intersecting s'_l are defined by two lines which form an angle of 45° with the current sweeping direction, as shown in Figure 4 (right). Having a static 45° angle is an advantage over connecting the borders to the edge of the screen, since the latter might result in grazing angles between the image space sweeping directions and the actual projection of the light direction. When handling point lights, lines are also drawn horizontally or vertically to ensure that the lines do not overlap. To avoid lines covering a zone they do not belong to, we use a 2D masking texture, generated by measuring the angle of the vector between the current pixel and s'_l. We also exploit this texture to perform edge blending between the zones, in order to improve image quality by allowing smoother transitions between the borders.

4.2 Synchronization

Using either of the two sweeping approaches, we have to ensure that the ray-casting is synchronized such that the parts of the volume being closer to the light source are processed earlier, than those being further away from it. To do this, we use OpenGL's atomic image access functions and the `glMemoryBarrierEXT()` function when building up the illumination cache. `glMemoryBarrierEXT()` in combination with the bitfield indicator `SHADER_IMAGE_ACCESS_BARRIER_BIT_EXT` provides a synchronization point, which should be called after each processed line to ensure that all memory access operations have been performed and the illumination cache has been built up before processing the next line. Thus, we can assume that when processing the fragments of a certain line l_i, all lines between l_i and the light source have already been processed. Thus all illumination effects resulting from the structures between l_i and the light source are already computed when processing l_i. By making this illumination information available during processing of l_i advanced illumination effects can be achieved.

4.3 GPU Ray-Casting

We have implemented the technique using OpenGL 3.0 by exploiting GLSL 1.30 functionality. This is necessary, since it is essential to have fragment scattering functionality in the fragment shader stage in order to write and read from random positions in the illumination cache. By binding textures to image units, we obtain the desired behaviour.

The pseudo code in Listing 4.1 shows how we perform the ray-casting while accessing the illumination cache at the same time. The

Listing 4.1 Pseudo code for the GPU part of IPSVI. The illumination cache data structures `illumCacheIn` and `illumCacheOut` serve as ping-pong buffers. Compositing occurs in light direction in order to obtain the illumination value, and along the view direction as in standard DVR.

```
col4 res = col4(0.0);
ivec2 illumPrevPos = calcIllumPos(s);
illumIn = getIllum(illumCacheIn, illumPrevPos);
for all samples s along the ray {
  ivec2 illumPos = calcIllumPos(s);
  illumIn = composite(illumIn, castLightRaySeg(s));

  float intensity = getIntensity(volume,s);
  col4 col = classify(intensity);
  col = localIllum(col);

  // back to front compositing (light direction)
  illumOut.rgb=(1.0-col.a)*illumIn.rgb+col.a*col.rgb;
  illumOut.a  =(1.0-col.a)*illumIn.a  +col.a;

  col = applyIllum(illumIn);

  // front to back compositing (view direction)
  res.rgb = res.rgb + (1.0-res.a)*col.a*col.rgb;
  res.a   = res.a   + (1.0-res.a)*col.a;

  if (illumPrevPos != illumPos) {
    if (toBeSaved(s)) {
      storeIllum(illumOut,illumCacheOut,illumPrevPos);
    }
    illumPrevPos = illumPos;
    illumIn = getIllum(illumCacheIn, illumPos);
  }
}
return res;
```

result of the ray-casting loop is stored in the 4-tuple `res` and represents the integration along one viewing ray as specified in Equation 1. In order to take into account the indirect illumination, we exploit the illumination cache bound to an image unit. Ping-pong buffering is used to avoid access conflicts, whereby the illumination cache to be read from is bound to `illumCacheIn` and the one to write to is bound to `illumCacheOut`. Before reading the indirect illumination contribution, we need to compute the appropriate `illumPosIn` coordinate for each sample s. This is done with the function `calcIllumPosIn`. To perform this computation efficiently, we use similar concepts as exploited when realizing projective texture mapping [35], but have to consider that adjacent illumination caches are in general not parallel in world space. Once we have obtained the correct position according to the the current sample s in the illumination cache we can fetch the indirect lighting contribution.

After we have computed the illumination contribution, we can fetch the current intensity from the volume, and optionally apply classification as well as standard local illumination. Now, that we know the contribution `col` of the current sample `s`, we can use it to modulate the previously computed incoming illumination which is stored in `illumIn`. Since we sweep away from the light source, we have to apply a back-to-front compositing as used in standard DVR. Now, that we have computed the outgoing illumination, we use the incoming illumination to modulate the current color `col` before it is used to perform compositing along the viewing ray. Finally, we have to write out the outgoing illumination to the illumination cache `illumCacheOut`. For efficiency reasons, this operation is only performed when necessary, i. e., when the writing position on the illumination cache is different as compared to the previous sample. This is not the case for every sample, since usually the sampling rate is higher than the actual number of texels in the illumination cache.

4.4 Special cases

Simply reading the illumination information at the appropriate position in the illumination cache would not be sufficient in some cases. When considering distant camera positions where the volume will cover only a few pixels in image space, it would result in a low number of processed sweep lines and lead to undersampling of the features potentially having an influence on the illumination. Therefore, we perform a piecewise integration within `castLightRaySeg()`, where we march along short ray segments pointing towards s_l' until we hit the previous illumination cache plane given in volume space. When performing a perspective projection, the length of these piecewise rays also depends on the distance to the camera, since viewing rays diverge. To account for this variable lengths, we exploit an additional 3-tuple `res` image storage, in which we store the according sample position in the previous illumination cache. The ray direction and its length for the piecewise integration are then obtained by subtracting the current sample position from the one stored in the `res` image storage. To combine the result of the previous lookup and the piecewise ray marching, we perform a compositing as used in standard DVR. As we demonstrate in Subsection 6.2, the piecewise integration can also be used when increasing the sweep line width in order to improve performance. However, regardless of the chosen line width, the first processed line always has to have a line width of one, in order to initialize the voxel positions used for the piecewise integration.

Furthermore, it should be pointed out that we cannot compute an appropriate sweep direction for the case when the light direction is parallel to the view direction. In this case IPSVI could be disabled or the previous sweeping direction could be used to allow coherency. In practice this proceeding did not lead to visually noticeable behavior. Also, to allow a smoother transition when the main sweeping direction dir''_{sweep} changes, the arbitrarily oriented sweep lines can be used.

While the presented concepts work for most cases, the described ray-casting paradigm leads to problems when a viewing ray travels through s_l. In these cases, the ray traversal needs to be also adapted based on s_l. When s_l lies in front of the volume, a front-to-back traversal is needed and when s_l is located behind the volume a back-to-front traversal is required, such that light is correctly propagated along these rays. In cases where s_l lies inside the volume, light propagation along these rays should even be chosen with respect to s_l, i. e., front-to-back behind s_l and back-to-front in front of s_l.

5 Performance Optimizations

In this section we discuss how to extend IPSVI with early ray termination and how to increase performance based on parameter choices.

5.1 Early Ray Termination

Today, early ray termination is widely used in volume ray-casting systems in order to improve rendering performance [15]. By stopping the ray marching when a certain opacity threshold is reached, computation time can be decreased without having a visual impact. However, early ray termination as implemented in most of todays ray-casters is in conflict with the volume illumination sweeping proposed in this paper. The problem is illustrated in Figure 5, where a volume containing only opaque structures is shown. As can be seen, the feature B is not visible from the current view position, since it is occluded by the fully opaque feature A. A typical case, where early ray termination would come into play, and the ray marching would be aborted before reaching B. While this does not pose a problem for standard DVR, since B is not visible, it poses a problem for IPSVI, since B affects the illumination of the feature depicted with C in Figure 5. When applying early ray termination, B would not appear in the illumination cache and therefore its influence on other features could not be considered.

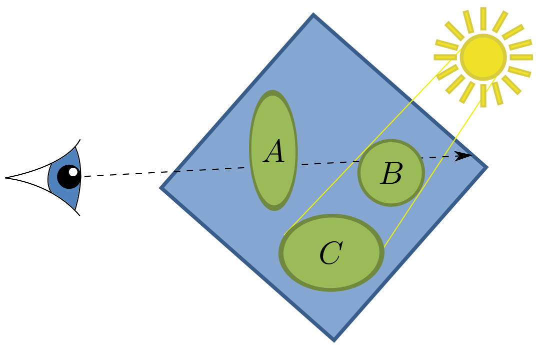


Fig. 5. Early ray termination is not inherently supported by IPSVI. Since feature *B* is occluded by feature *A*, early ray termination would not allow it to appear in the illumination cache. Thus, the influence of feature *B* on feature *C* could not be simulated.

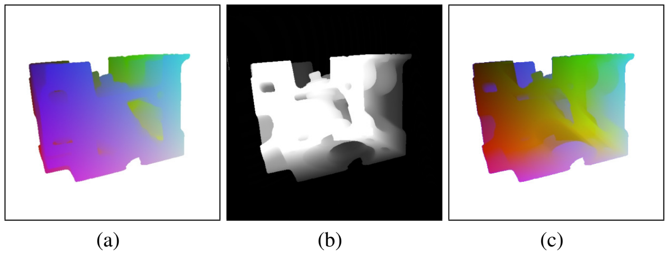


Fig. 6. To enable early ray termination, we modify the exit point texture, by color coding volume coordinates (a) where the early ray termination threshold is reached. Based on the light source position, the lengths of the view rays, shown luminance coded in (b), are propagated to obtain the modified exit point texture (c). In this figure, the light source is located top left.

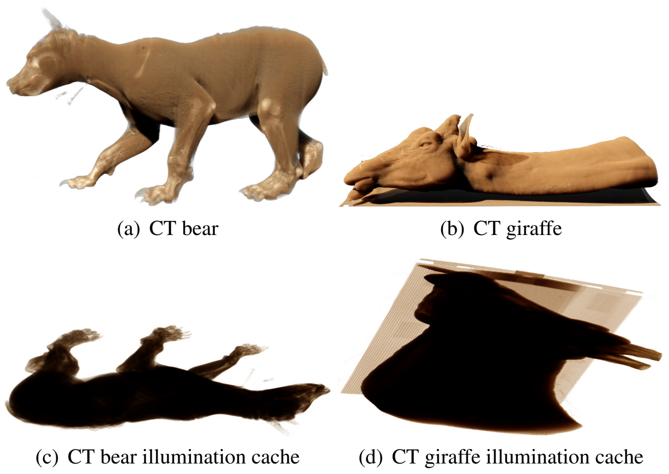
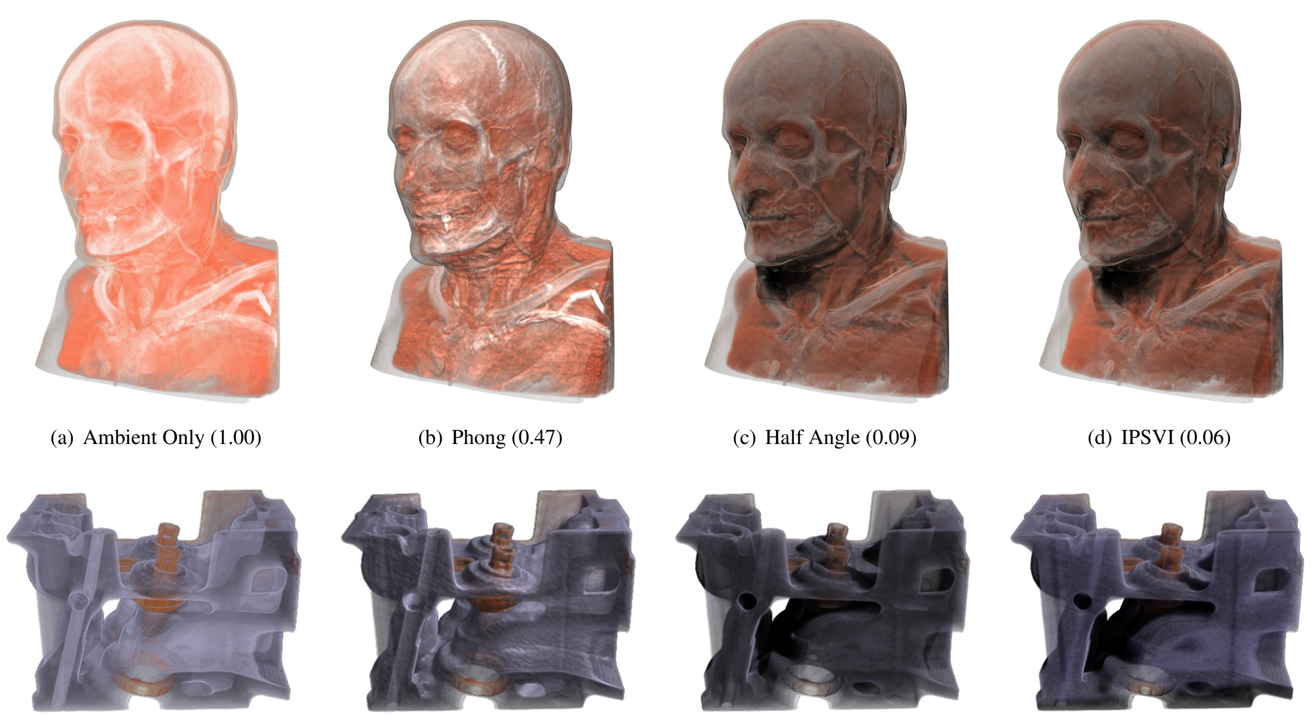


Fig. 7. A CT scan of a bear ($512 \times 512 \times 412$ voxels) and a giraffe ($512 \times 512 \times 553$ voxels) rendered in in real-time using IPSVI. Hard shadows are cast onto the back legs and the chest. The bottom images shows the state of illumination cache at rendering completion.

To be able to benefit from early ray termination, we propose an extension to IPSVI which enables to further traverse only those rays reaching the opacity threshold, which are required for a sufficient illumination cache. We do so, by providing modified exit points for the used ray-caster. To be able to actually benefit from this extension, it is important that it can be implemented efficiently. The proposed extension is based on a simplified ray-casting pass, which is performed before the actual rendering. Within this ray-casting pass, we exploit a lower sampling rate and disable shading, and write out for each ray the volume position $s_f(r)$, at which the opacity threshold triggering early ray-termination has been reached, as well as $s_b(r)$, where the furthest away structures have been encountered along r. Thus, we obtain two images as shown in Figure 6 (a) and Figure 6 (b), where



(a) Ambient Only (1.00) (b) Phong (0.47) (c) Half Angle (0.09) (d) IPSVI (0.06)

(e) Ambient Only (1.00) (f) Phong (0.47) (g) Half Angle (0.10) (h) IPSVI (0.07)

Fig. 8. Comparison of IPSVI with three other commonly used approaches. From left to right: unshaded DVR, gradient-based Phong shading, half angle slicing and image plane sweep volume illumination. The number inside the subcaption brackets represents a normalized speed factor between the different methods, where 1.00 indicates highest measured performance rate.

the volume positions are color-coded in (a) and the ray length is depicted by a gray value in (b). To use these results of the pre-rendering pass, we need to further modify them. The goal is to find the maximal ray length for each pixel position, which is needed to build up a correct illumination cache. To do so, we sweep over the images in parallel in a similar fashion as during the CPU line sweeping discussed in Subsection 4.1. However, this time we sweep towards the light source instead of away from it, and only consider 2D information. For each p_i encountered on the sweep line, we analyze the predecessor p_{i-1} on the previous sweep line, which is in the point light case intersected by a line through p_i and s_l'. In order to determine the maximal ray length for p_i, we assume that p_{i-1} lies further away from s_l' than p_i. Thus, the structures encountered by the ray cast through p_i might have an influence on the illumination of those structures encountered by p_{i-1}. Therefore, the maximal ray length $l_{max}(r(p_i))$ for a ray r through pixel p_i has to be computed as $l_{max}(r(p_i)) = min(s_b(r(p_i)), max(s_f(r(p_{i-1})), l_{max}(r(p_{i-1}))))$. During this processing, we write out the $l_{max}(r(p_i))$. Thus, we can transform the initial textures shown in Figure 6 (a) and Figure 6 (b) into the one shown in Figure 6 (c), which can be directly used as exit point texture defining the maximal ray lengths for IPSVI.

To reduce the overhead of our early ray termination extension, the described processing is performed at a lower image resolution than the resolution of the final image. Since this only affects the resolution of the exit point texture and the final image is ray-cast using the full resolution, our extension does not result in any visible effects. We will discuss the performance impact of this extension in Subsection 6.2.

5.2 Line Width & Illumination Cache Size

In both described cases, when using directional or point light sources, we can compute the illumination as introduced in Section 3, since we limit ourselves to forward scattering effects. To limit the amount of draw calls a line width greater than one can be chosen. As shown in Subsection 6.2, this improves performance since it better exploits the block-wise GPU processor layout and therefore results in a higher degree of parallelization. The amount of lines to be processed is further reduced by restricting the line processing to the area inside the axis aligned bounding box of the volume.

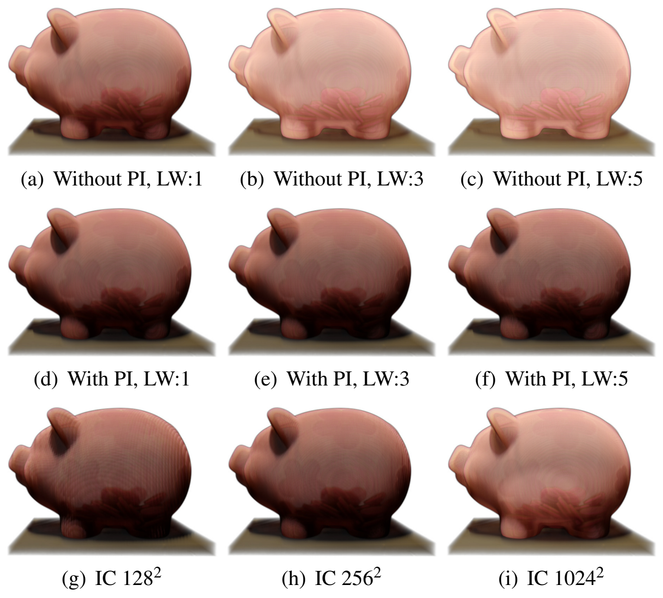
For optimal use of the illumination cache we have the option to employ a light frustum adaptation. Therefore, the volume bounding box is projected into light space and used to define a rectangular area which contains the entire proxy geometry as seen from the light source. While this is similar to how we define the area to be covered by sweep lines, we use this information here to redefine the near plane seen from the light source for an optimal field of view, when projecting the current sample on the illumination cache. Though we change the size accordingly, we keep the aspect ratio in order to avoid shadow popping artifacts. The cache optimization can further benefit from a camera frustum clipping of the proxy geometry against the volume bounding box, such that the illumination cache can be adapted to cover only the visible parts of the volume.

6 Results & Discussion

To assess the practical use of IPSVI, we will first provide some visual results before discussing performance measurements.

6.1 Visual Results

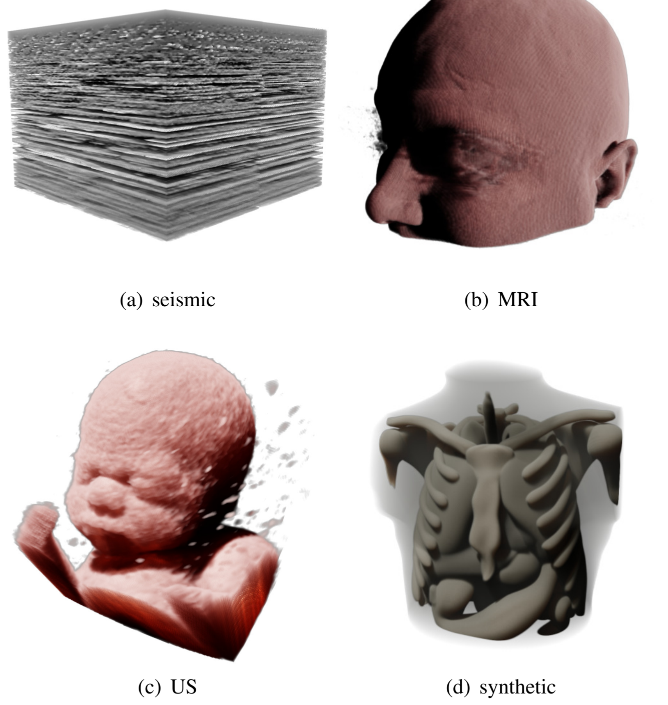
To investigate the visual quality achievable with IPSVI, we have applied it to various data sets of different modalities. Figure 1 shows a rendering of a CT scan of a penguin having a resolution of $512 \times 512 \times 1146$ voxels. As can be seen, shadowing and scattering effects are visible, which increase the degree of realism. Due to the small memory footprint of IPSVI, the effects can be applied despite the relatively large size of the data set, as also demonstrated by the rendering of the giraffe's head and the bear in Figure 7. Because the effects achieved with IPSVI are similar to those which can be obtained with half angle slicing [12], we have decided to also include a comparison with this technique. Figure 8 shows this comparison, as well as the


(a) Without PI, LW:1 (b) Without PI, LW:3 (c) Without PI, LW:5

(d) With PI, LW:1 (e) With PI, LW:3 (f) With PI, LW:5

(g) IC 128^2 (h) IC 256^2 (i) IC 1024^2

Fig. 9. Comparison of renderings of the piggybank data set, with and without piecewise integration (PI), different line widths (LW) and with different illumination cache size (IC). As shown, the choice of parameters can have a significant impact on the visual result. The bottom images are rendered with PI as well as a line width of one.


(a) seismic (b) MRI

(c) US (d) synthetic

Fig. 10. Application of IPSVI to frequently used modalities, which do not allow to facilitate gradient-based shading due to a relatively low signal-to-noise ratio or the discrete nature of the data.

differences to unshaded DVR and Phong lighting. As can be seen in the first row of Figure 8, both half angle slicing as well as IPSVI improve the degree of realism as compared to unshaded DVR and Phong shading. Furthermore, the difference between half angle slicing and IPSVI is hardly noticeable. While the first statement is also true for the engine data set shown in the second row, the differences between half angle slicing and IPSVI are more prominent. As can be seen, the color saturation of the half angle result is slightly different, while our cases better matches the color of the unshaded DVR as well as the Phong result. Furthermore, there are slight difference regarding the shadow borders when comparing ours with the half angle results. This differences at the shadow borders are more noticeable, since the engine data set has been rendered with a higher degree of opacity, which results in harder shadow borders.

Figure 9 shows a visual comparison of IPSVI when changing parameters such as line width or illumination cache size, as well as renderings with and without piecewise integration. As shown in Figure 9 (a), (b) and (c) the quality without piecewise integration is far less satisfying than in (d), (e) and (f) where this method is used. Line artifacts are, unfortunately, noticeable in some cases when a large line width is used (see Figure 9 (c) and (f)). To emphasize the importance of having an optimal illumination cache size, we vary its size in (g), (h) and (i) to such extent that false results and artifacts are introduced. A small illumination cache size might introduce severe line artifacts as less visibility information can be stored.

While all the results discussed so far have shown the application to CT data sets, IPSVI is also beneficial when applying it to other modalities. Especially when dealing with modalities where the gradient is not sufficient to be used for shading, IPSVI can improve the image quality. Figure 10 shows the application to such modalities. In Figure 10 (a) the application to a seismic data set is shown. Seismic data is known to suffer from a low signal-to-noise ratio, which makes the visual differentiation of horizon layers difficult. As can be seen in Figure 10 (a), the application of IPSVI adds depth to the image and allows to grasp the relationship between adjacent layers. In contrast to CT data, MRI suffers from noisy gradients. Since IPSVI does not rely on gradient computation, the surfaces appear smoother, as can be seen in Figure 10 (b). Furthermore, the incorporation of scattering effects results in a more realistic skin appearance. Regarding noise, ultra sound (US) is probably the most difficult modality to be visualized. While gradient-based shading effects would still produce reasonable images when applied to MRI data, this is not the case for US data. In contrast we can generate high fidelity images even based on US data (see Figure 10 (c)). So far the discussed modalities all suffered from a too low signal-to-noise ratio. A different case is the synthetic data set shown in Figure 10 (d). When dealing with synthetic data sets or segmentation masks, no partial volume effect is present and thus boundaries are not smoothed out. This is often the reason for staircase artifacts to become visible, when applying gradient-based shading techniques to these data sets. In contrast when applying IPSVI and choosing a rather high albedo, the scattering contribution results in an inherent border smoothing and thus no staircase artifacts are visible.

6.2 Performance Analysis

To analyze the performance of IPSVI, we have measured the average frames per second (fps) over 100 frames while performing a rotation around the camera up-vector. Measurements have been performed for different cases by varying the screen resolution, line width and data set size, as these parameters have a major impact on the overall performance. Table 1 shows the results, we could achieve on a standard desktop computer, equipped with an Intel Xeon W3550 processor running at 3.07 GHz, 6 GB of RAM and an nVidia GeForce GTX 580 graphics processing unit. Our results show, that the factor having the highest impact on performance, is an increased image resolution, which was also expected since IPSVI is an image-based algorithm. Furthermore, it is visible and obvious that a higher line width increases performance in most cases. The larger data set are actually in some cases rendered with better performance as compared to smaller data sets. The reason for this is that we only draw lines which cover the volume bounding geometry projected into screen space. As an example, the giraffe's projected bounding volume does not cover as much of the screen space area, therefore it is rendered faster at certain screen resolutions. For additional reference, the penguin dataset in Figure 1, which has a resolution of $512 \times 512 \times 1146$ voxels, is rendered at 0.8 fps (at 1.9 fps

Table 1. Frame rates of IPSVI with piecewise integration enabled for different data sets, screen resolutions and line widths.

image space	data set:	US Baby			CT Engine			CT Giraffe		
resolution	size:	$199 \times 159 \times 161$			$256 \times 256 \times 256$			$512 \times 512 \times 553$		
(pixels)	spacing:	$1 \times 1 \times 1$			$1 \times 1 \times 1$			$0.977 \times 0.977 \times 3$		
	line width:	LW:1	LW:2	LW:4	LW:1	LW:2	LW:4	LW:1	LW:2	LW:4
64×64		50.28 fps	91.58 fps	116.28 fps	33.7 fps	65.40 fps	82.17 fps	58.28 fps	110.50 fps	152.67 fps
128×128		25.94 fps	48.59 fps	62.11 fps	16.23 fps	31.74 fps	40.83 fps	27.75 fps	52.55 fps	70.42 fps
256×256		12.70 fps	25.03 fps	33.98 fps	7.79 fps	15.37 fps	21.16 fps	13.16 fps	25.85 fps	32.54 fps
512×512		5.68 fps	11.86 fps	17.60 fps	3.55 fps	7.11 fps	10.74 fps	6.42 fps	12.69 fps	15.30 fps
1024×1024		2.63 fps	5.44 fps	8.33 fps	1.61 fps	3.24 fps	5.02 fps	3.05 fps	6.20 fps	7.75 fps

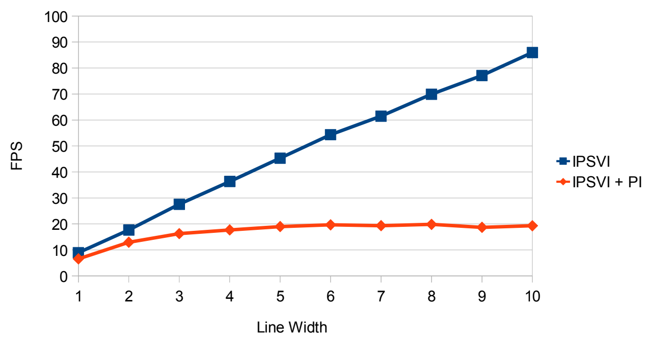


Fig. 11. Performance measurements of IPSVI with/without piecewise integration (PI) while adapting the line width. The results have been achieved for a screen resolution of 256^2 and with a data set of size 256^3.

with unshaded DVR) with a screen size of 1650×800 and with the use of bilinear interpolation when sampling as well as updating the illumination cache. We have also supplied a relative performance factor (normalized to the measured speed of Ambient Only) to provide a performance comparison, apart from the visual comparison in Figure 8.

We have also measured the performance of the algorithm when utilizing the piecewise integration. As can be seen in Figure 11, the piecewise integration reduces performance when increasing the line width. The reason for this is that, as the line width increases the ray length for the piecewise integration also increases. Due to this effect, the performance may actually decrease when increasing the line width, such that we need to consider a balance between the number of draw calls and the size of the integration area to acquire the best possible performance. The incorporation of our modified early ray termination algorithm does in fact increase performance. However, as we have noticed in our tests, it is crucial to apply this extension on a low resolution representation of the exit point texture to achieve a performance increase, due to the overhead of the additional line sweeping. It is also feasible to assume that the performance gain depends on the spatial relation of the data, as well as the choice of transfer function. Furthermore, in comparison to regular ray-casting, IPSVI is slower, mainly due to the amount of CPU processing and synchronization performed during each render pass. Work load might not be optimal as compared to unshaded DVR, but when considering computing the influence of features being closer to the light source, these generally need to be computed and stored in a preprocessing stage for many other global illumination techniques.

7 Conclusions & Future Work

In this paper, we have presented *image plane sweep volume illumination*, a novel sweep-based volumetric illumination technique. By sweeping the image plane, we are able to synchronize illumination computation, and thus integrate advanced volume illumination techniques into a standard ray-casting based volume renderer without either performing pre-processing or building up an illumination volume. Thus, we are able to generate high-quality volume renderings with advanced illumination effects at interactive frame rates. To our knowledge, this is the first approach which allows integration of these effects directly into a ray-casting based renderer. We were able to show how to easily implement the concepts and have discussed the competitive performance as well as quality results, which have been obtained when comparing IPSVI to previous approaches. Besides its simple realization, IPSVI has several benefits. Since we do not need to store an intermediate illumination volume, we have a considerably smaller memory footprint as compared to previous techniques and can therefore apply IPSVI even to large volumetric data sets without worrying about extra GPU memory consumption. Furthermore, clipping is inherently supported, since the illumination is computed based on what is actually visible. IPSVI can also be applied when rendering multimodal data sets, because the illumination computation is directly performed within the ray-caster, which makes it independent of the data set resolution and orientation. While the illumination cache can be seen as similar to the light buffer used in slice-based techniques, its resolution and precision can be lower since it is only used for illumination calculation and not for the actual compositing along the view ray.

IPSVI also has limitations. In those cases, where the volume is partially outside the screen, we do not compute the illumination for the invisible parts. Thus, structures outside the screen do not affect the illumination, which is not the case in other approaches [12, 34, 40]. While this might probably be a downside in some cases, it can be also beneficial in certain scenarios. For instance, when zooming inside a volume, global shadows may result in a rather dark appearance and a low contrast. On the other hand, when just considering the visible structures during the illumination processing, the relation between the visible structures is enhanced while having a brighter image with a potentially higher contrast. For cases where this behavior is not desired, we plan to solve it by applying a multi-resolution rendering approach, where we render the relevant parts of the volume outside the screen at a lower resolution into an off-screen illumination cache first.

In the future we also see interesting ways to extend IPSVI. Rim lighting is often used to further emphasize object shapes, by placing an additional rim light behind the object's silhouette. IPSVI is extensible to support two light sources, and thus enabling rim lights. By choosing the sweep direction, such that it is perpendicular to the line traveling in image space through two light positions, we are able to propagate illumination for both light sources simultaneously. Furthermore, volumetric light sources could be integrated by modifying the piecewise ray-casting. To investigate the perceptual benefits of IPSVI, we would also like to conduct a user study.

Acknowledgments

The presented concepts have been integrated into the Voreen volume rendering engine (*www.voreen.org*). The CT animal data is courtesy of Center for Medical Image Science and Visualization (CMIV).

References

[1] K. M. Beason, J. Grant, D. C. Banks, B. Futch, and M. Y. Hussaini. Pre-computed illumination for isosurfaces. In *Conference on Visualization and Data Analysis*, pages 1–11, 2006.

[2] U. Behrens and R. Ratering. Adding shadows to a texture-based volume renderer. In *IEEE Int. Symp. on Volume Visualization*, pages 39–46, 1998.

[3] I. Bitter and A. Kaufman. A ray-slice-sweep volume rendering engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 121–130, 1997.

[4] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *1994 Symp. on Volume visualization*, pages 91–98, 1994.

[5] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 symposium on Volume visualization*, pages 91–98, 1994.

[6] R. Farias, J. S. B. Mitchell, and C. T. Silva. Zsweep: an efficient and exact projection algorithm for unstructured volume rendering. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 91–99, 2000.

[7] C. Giertsen. Volume visualization of sparse irregular meshes. *IEEE Computer Graphics and Applications*, 12:40–48, 1992.

[8] M. Hadwiger, A. Kratz, C. Sigg, and K. Bühler. GPU-accelerated deep shadow maps for direct volume rendering. In *ACM SIGGRAPH/EG Conference on Graphics Hardware*, pages 27–28, 2006.

[9] F. Hernell, P. Ljung, and A. Ynnerman. Efficient ambient and emissive tissue illumination using local occlusion in multiresolution volume rendering. In *IEEE/EG Int. Symp. on Volume Graphics*, pages 1–8, 2007.

[10] F. Hernell, P. Ljung, and A. Ynnerman. Interactive global light propagation in direct volume rendering using local piecewise integration. In *IEEE/EG Int. Symp. on Volume and Point-Based Graphics*, 2008.

[11] P. V. J. Daz, H. Yela. Vicinity occlusion maps - enhanced depth perception of volumetric models. In *Computer Graphics Int.*, pages 56–63, 2008.

[12] J. Kniss, S. Premoze, C. Hansen, and D. Ebert. Interactive translucent volume rendering and procedural modeling. In *IEEE Visualization 2002*, pages 109–116, 2002.

[13] J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson. A model for volume lighting and modeling. *IEEE Trans. on Visualization and Computer Graphics*, 9(2):150–162, 2003.

[14] J. Kronander, D. Jonsson, J. Low, P. Ljung, A. Ynnerman, and J. Unger. Efficient visibility encoding for dynamic illumination in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2011. to appear.

[15] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization '03*, 2003.

[16] F. Lindemann and T. Ropinski. Advanced light material interaction for direct volume rendering. In *IEEE/EG Int. Symp. on Volume Graphics*, pages 101–108, 2010.

[17] F. Lindemann and T. Ropinski. About the influence of illumination models on image comprehension in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics (Vis Proceedings)*, 2011.

[18] P. Ljung, F. Hernell, and A. Ynnerman. Local ambient occlusion in direct volume rendering. *IEEE Trans. on Visualization and Computer Graphics*, 15(2), 2009.

[19] T. Lokovic and E. Veach. Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 385–392, 2000.

[20] N. Max. Optical models for direct volume rendering. *IEEE Trans. on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[21] N. Max and M. Chen. Local and global illumination in the volume rendering integral. In *Scientific Visualization: Advanced Concepts*, pages 259–274, 2010.

[22] C. Me and T. Ropinski. Efficient acquisition and clustering of local histograms for representing voxel neighborhoods. In *IEEE/EG Volume Graphics*, pages 117–124, 2010.

[23] B. Moloney, M. Ament, D. Weiskopf, and T. Möller. Sort first parallel volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2011. to appear.

[24] E. Penner and R. Mitchell. Isosurface ambient occlusion and soft shadows with filterable occlusion maps. In *IEEE/EG Int. Symp. on Volume and Point-Based Graphics*, pages 57–64, 2008.

[25] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[26] F. Qiu, F. Xu, Z. Fan, N. Neophytos, A. Kaufman, and K. Mueller. Lattice-based volumetric global illumination. *IEEE Transactions on Visualization and Computer Graphics*, 13:1576–1583, 2007.

[27] C. Rezk-Salama. GPU-based monte-carlo volume raycasting. In *Pacific Graphics*, 2007.

[28] C. Rezk-Salama, M. Hadwiger, T. Ropinski, and P. Ljung. Advanced illumination techniques for gpu volume raycasting. In *ACM SIGGRAPH Courses Program*, 2009.

[29] T. Ritschel. Fast GPU-based Visibility Computation for Natural Illumination of Volume Data Sets. In *Short Paper Eurographics 2007*, pages 17–20, 2007.

[30] T. Ropinski, C. Döring, and C. Rezk-Salama. Interactive volumetric lighting simulating scattering and shadowing. In *IEEE Pacific Visualization*, pages 169–176, 2010.

[31] T. Ropinski, J. Kasten, and K. H. Hinrichs. Efficient shadows for GPU-based volume raycasting. In *Int. Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 17–24, 2008.

[32] T. Ropinski, J. Meyer-Spradow, S. Diepenbrock, J. Mensmann, and K. H. Hinrichs. Interactive volume rendering with dynamic ambient occlusion and color bleeding. *Computer Graphics Forum (Eurographics 2008)*, 27(2):567–576, 2008.

[33] M. Ruiz, I. Boada, I. Viola, S. Bruckner, M. Feixas, and M. Sbert. Obscurance-based volume rendering framework. In *IEEE/EG Int. Symp. on Volume and Point-Based Graphics*, pages 113–120, 2008.

[34] M. Schott, V. Pegoraro, C. Hansen, K. Boulanger, and K. Bouatouch. A directional occlusion shading model for interactive direct volume rendering. *Computer Graphics Forum (Proceedings of Eurographics/IEEE VGTC Symposium on Visualization 2009)*, 28(3):855–862, 2009.

[35] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast shadows and lighting effects using texture mapping. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, pages 249–252, 1992.

[36] C. T. Silva and J. S. B. Mitchell. The lazy sweep ray casting algorithm for rendering irregular grids. *IEEE Transactions on Visualization and Computer Graphics*, 3:142–157, 1997.

[37] M. Smelyanskiy, D. Holmes, J. Chhugani, A. Larson, D. M. Carmean, D. Hanson, P. Dubey, K. Augustine, D. Kim, A. Kyker, V. W. Lee, A. D. Nguyen, L. Seiler, and R. Robb. Mapping high-fidelity volume rendering for medical imaging to cpu, gpu and many-core architectures. *IEEE Trans. on Visualization and Computer Graphics*, 15:1563–1570, 2009.

[38] A. J. Stewart. Vicinity shading for enhanced perception of volumetric data. In *IEEE Visualization*, page 47, 2003.

[39] V. Timonen and J. Westerholm. Scalable height field self-shadowing. *Computer Graphics Forum*, 29(2):723–731, 2010.

[40] V. Šoltészová, D. Patel, S. Bruckner, and I. Viola. A multidirectional occlusion shading model for direct volume rendering. *Computer Graphics Forum (Eurographics/IEEE VGTC Symp. on Visualization 2010)*, 29(3):883–891, 2010.

[41] V. Šoltészová, D. Patel, and I. Viola. Chromatic shadows for improved perception. In *Proceedings of Non-Photorealistic Animation and Rendering (NPAR)*, 2011.

[42] C. Zhang and R. Crawfis. Volumetric shadows using splatting. In *IEEE Visualization*, pages 85–92, 2002.

[43] C. Zhang and R. Crawfis. Shadows and soft shadows with participating media using splatting. *IEEE Trans. on Visualization and Computer Graphics*, 9(2):139–149, 2003.

[44] S. Zhukov, A. Iones, and G. Kronin. An ambient light illumination model. In *EGRW '98: Proceedings of the Eurographics workshop on Rendering*, pages 45–55, 1998.