Ambient Occlusion Effects for Combined Volumes and Tubular Geometry

Mathias Schott, Tobias Martin, A.V. Pascal Grosset *Student Member IEEE*, Sean T. Smith and Charles D. Hansen, *Fellow IEEE*

Abstract—This paper details a method for interactive direct volume rendering that computes ambient occlusion effects for visualizations that combine both volumetric and geometric primitives, specifically tube shaped geometric objects representing streamlines, magnetic field lines or DTI fiber tracts. The algorithm extends the recently presented Directional Occlusion Shading model to allow the rendering of those geometric shapes in combination with a context providing 3D volume, considering mutual occlusion between structures represented by a volume or geometry. Stream tube geometries are computed using an effective spline based interpolation and approximation scheme that avoids self intersection and maintains coherent orientation of the stream tube segments to avoid surface deforming twists. Furthermore, strategies to reduce the geometric and specular aliasing of the stream tubes are discussed.

Index Terms—Volume rendering, ambient occlusion, stream tubes

1 Introduction

Data sets arising in many scientific research areas are often combinations of commonly used scalar fields and other types of data. Vector fields typically arise while simulating the mechanics of fluids or gases. A variety of methods exist to render those directly, but often, streamlines are traced through those vector fields in order to create a high level abstracted visualization of those data sets. Similarly, DTI fiber tractography is used to gain insight about the configuration and orientation of neural pathways in the field of neurosciences in order to increase the understanding of the functioning of the brain.

It is desirable to combine the visualization of a scalar field with that of streamlines or DTI fibers since they are typically registered with respect to each other. This poses a problem however since the visual interplay between those rather different data modalities introduces additional context that is key in comprehending the combined data sets. It is especially important to be able to reason about the spatial arrangement of the geometry with respect to the volume. Existing visualization techniques are less suited to helping users gain insight into those data sets, since commonly available techniques employ only local information in the form of Phong shading.

- *Mathias Schott is with NVIDIA corporation, Santa Clara, CA 95050. E-mail: mschott@nvidia.com*
- *Tobias Martin is with the Department of Computer Science at ETH, Zürich,Switzerland. E-mail: martint@inf.ethz.ch*
- *Sean T. Smith is with the Department of Chemical Engineering at The University of Utah E-mail:sean.t.smith@utah.edu*
- *A.V. Pascal Grosset and Charles D. Hansen are with the School of Computing and SCI Institute, University of Utah, Salt Lake City, UT 84112. E-mail: {zwetval, hansen}@cs.utah.edu*

In this paper, we present a method for interactive direct volume rendering that allows the computation of occlusion effects for volumetric data sets with both solid and transparent features, combined with solid tube-like geometry from DTI fiber tractography or streamline tracing. The proposed method is an extension of the directional occlusion shading model presented by Schott *et al.* [24] and as such is based on incremental filtering, which has been successfully used in the past to approximate integration for computing advanced volumetric scattering and shading effects [11], [18], [23], [24], [31].

The occlusion effects of the geometric structures are incorporated straightforwardly by modifying the occlusion buffer update of the volumetric occlusion shading method. Additionally, a vicinity occlusion term is computed using the depth buffer of the geometric structures in order to capture more detailed occlusion effects introduced by the geometry. The combined occlusion effects of both the geometry and the volume provide additional context by increasing the spatial comprehensibility due to the consideration of neighboring structures of both volumetric and geometric origin. The method in this paper is based on work presented previously [22] and encompasses the following additions:

- Inclusion of more algorithmic details in Sections 3.2.4, 3.2.5 and 3.2.6.
- Derivation of an effective stream tube interpolation and approximation scheme that avoids self-intersection and maintains consistent orientation of the individual stream tube segments in Section 4.
- Discussion of various anti-aliasing approaches in order to reduce the geometric and specular aliasing of the stream tubes in Section 5.

This paper is structured as follows: Section 2 discusses methods related to screen space ambient occlusion and rendering of streamlines and fibers. Section 3.1 motivates

our approach, followed by a discussion of an integration into a slice-based direct volume rendering system in Section 3.2. Section 4 details the method for computing the geometry of the stream tubes. Section 5 discusses various anti-aliasing methods and results are presented and discussed in Section 6, followed by conclusions and future work in Section 7.

2 Related Work

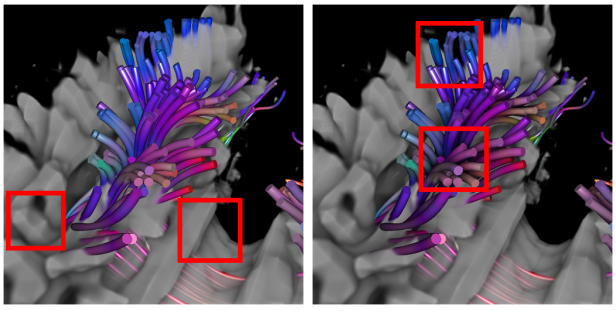
The directional occlusion shading method presented by Schott *et al.* [24] builds the foundation for the extension considering the mutual occlusion of volumetric and geometric structures. It computes volumetric occlusion effects by maintaining and blurring an additional occlusion buffer while traversing the slices.

Screen Space Ambient Occlusion techniques have recently received considerable interest after details about commercial implementations were presented publicly [17]. They use the depth buffer of a scene in order to approximate the geometry in the neighborhood of a pixel and use this representation to estimate the occlusion, thus being independent of the geometric complexity of the scene. Shanmugam *et al.* [27] compute high and low frequency occlusion by defining spheres at each pixel within which they then sample the depth and normal buffers. Our proposed method also computes high and low frequency occlusion effects separately, but differs by capturing low frequency occlusion effects in the volumetric occlusion buffer, instead of approximating them with spheres. Horizon based ambient occlusion methods [2] use the tangent plane of the surface in order to avoid evaluating the occlusion in those parts of the hemisphere that are known to be occluded. Loos *et al.* [12] reformulate ambient occlusion as a 3D volumetric integral which they then solve using line and area samples from the depth buffer in order to reduce the under-sampling artifacts inherent to the point sampling of other screen space methods. The Alchemy screen space ambient obscurance method [15] combines aspects of previous screen space ambient occlusion methods with a different derivation of obscurance in order to increase robustness and performance of their algorithm, which captures obscurances from details of varying scale and provides artist control over key parameters of their model. Reinbothe *et al.* [20] combine image space and object space techniques to compute ambient occlusion effects that also consider geometry not rasterized using other screen space ambient occlusion methods. Bavoil *et al.* [1] propose to use depth peeling in order to consider the occlusion from geometry not captured in the depth buffer. Ambient occlusion effects are computed at lower resolution and then up sampled in order increase the overall performance. Huang *et al.* [9] separate the screen space ambient occlusion computation along two directions in order to increase the performance in conjunction with geometry aware blurring and interleaved sampling. Vicinity Occlusion Maps [5], a direct volume rendering method, use a similar approach to shade a depth buffer derived from accumulated opacities. Summed area tables are used to determine also the low frequency, global occlusion effects of surface like structures contained in the volumetric data set. In contrast, our proposed method uses directional occlusion shading to compute the occlusion effects for the volume and global geometric structures, and a vicinity occlusion term based on the depths of the geometry to determine the occlusion of the high frequency, local geometric structures.

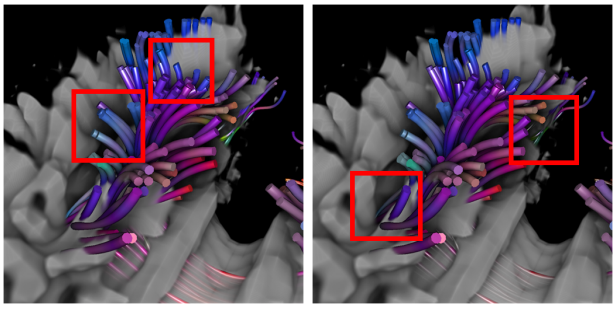
Wenger *et al.* [32] employ a two level rendering method to combine scalar volume rendering with rendering of vector fields, represented by thin threads which are created by filtering the lines into volumes using a cubic B-spline filter spanning multiple voxels. The resolution of those derived thread and halo volumes limits the maximal representable thread radius. Attributes of threads and halos are stored in those volumes and transfer functions are used to classify those, allowing selective culling by setting opacity accordingly. Melek *et al.* [16] use self orienting surfaces [25] to render hundreds of thousands of threads interactively on the GPU. Depth cuing and local lighting are used to shade those surfaces. A non-interactive global illumination rendering method based on hair rendering is proposed to yield high quality images. There, opacity shadow maps are used to provide self shadowing of the threads. An ambient occlusion term for rectangular shaped volumes is derived, based on the distance to the boundary of the volume. Schussman *et al.* [26] render large amounts of dense line data non-interactively by subdividing thin lines into voxels, which store radiance and opacity anisotropically, compressed using spherical harmonics, which are then rendered using direct volume rendering. Stoll *et al.* [28] use a combined CPU / GPU approach to splat lines as generalized cylinders with local lighting and halos and texture and shadow mapping. Tessellated geometry is rendered in areas where the splatted impostors are parallel to the view direction. Zöckler *et al.* [34] exploit fixed-function texture mapping hardware to compute Phong lighting on transparent streamline segments. Stompel *et al.* [29] render streamlines non-photo realistically as strokes of varying lengths. Color gradients and semi-transparency are used to indicate directional information. Interrante *et al.* [10] perform 3D Line Integral Convolution (LIC) to create a flow representing color texture. Variations in hue are used to visually separate dense line bundles, volumetric halos to increase depth perception. Li *et al.* [30] scan convert streamlines into a 3D texture as a preprocessing step, while extracting additional attributes. This allows an interactive exploration by manipulation of a dependent lookup into an *appearance texture*, which can be used to give the effect of illuminated streamtubes, tone shading, or silhouette enhancements. Everts *et al.* [7] use an illustrative approach to render dense line data by creating halos around line bundles, while also increasing the depth perception by attenuating their widths. Their method could be combined with scalar volume rendering, however it is unclear how to maintain context between the volume and the geometry.

3 Combining Occlusion Effects

The recently presented Directional Occlusion Shading (DOS) method [24] extends a slice-based volume renderer to



(a) *Volume* → *Volume* (b) *Geometry* → *Geometry*



(c) *Geometry* → *Volume* (d) *Volume* → *Geometry*

Fig. 1. Occlusion effects between the volume and geometry, where each image introduces additional interactions, beginning with the occlusion of a) the volume , b) the geometry, c) between the geometry and the volume and d) between the volume and the geometry.

compute and store approximate volumetric occlusion effects. There, view-aligned slices are traversed and composited in front-to-back order, incrementally filtering and attenuating an additional 2D image, the so called *occlusion buffer*, to propagate through the volume the occlusion effects, which are conceptually similar to soft shadows cast by structures illuminated by a small circular area light source at the viewer's position. Using this approach, solid and semi-transparent voxels act both as shadow casters and receivers.

3.1 Motivation of the Presented Approach

An algorithm aiming to provide combined rendering of volumetric and geometric structures has to address their mutual interactions, and how to incorporate them into the overall rendering method. Volumetric shading methods that consider only basic local shading models have a minimal set of those interactions, such as determining the visibility of voxels behind the geometry in order to avoid computing their contributions, and properly attenuating the background during compositing the volume on top of the geometry. However, semi-global methods, such as the volumetric directional occlusion shading method [24], introduce additional interactions that need to be addressed in order to provide for a visualization technique that renders depth cues that plausibly combine the occlusion effects of both volumetric and geometric structures.

Volumetric occlusion effects from the *volume* into the *volume* can be computed using the directional occlusion shading approach [24], and can easily be combined with the Phong surface shading of the geometry, as Figure 1(a) shows. Notable here is the lack of depth discrimination between the geometric structures themselves. Occlusion effects between the geometric structures can be computed independently using a screen space ambient occlusion approach, as Figure 1(b) demonstrates, which enhances the depth perception of the geometry, due to the approximated occlusion effects. This naive method however considers each type of occlusion effects *independently*, thus missing important interactions by ignoring occlusion shadows cast from the *geometry* into the *volume* (Figure 1(c)) and those from the *volume* onto the *geometry* (Figure 1(d)). Only the combination of all the occlusion effects between the volume and the geometry create a consistently shaded rendering which enhances the depth perception across the whole image, which is important in order to create a comprehensive visualization. The proposed combined method for rendering occlusion effects considers all the occlusion effects highlighted in Figure 1 and encompasses:

- an adapted volumetric occlusion shading method that is modified to consider the shadows cast by the *geometry* into the *volume* in addition to those cast by the *volume* onto the *volume*,
- an additional data structure, the so called *partial occlusion buffer*, which stores the shadows cast by the *volume* onto the *geometry*,
- an additional screen space ambient occlusion term, the so called *vicinity occlusion term*, which computes the occlusion effects between *geometry* and surrounding *geometry*, based on inspiration by recently proposed techniques [2], [5] and
- the combination of the volumetric occlusion term with the partial occlusion term yielding a combined occlusion term used to shade the geometric surfaces.

3.2 Integration into a Slice-Based Volume Renderer

Algorithm 1 The main combined occlusion shading algorithm

```
ComputeGBuffer()
eye_buffer ⇐ background_color
volume_occl_buf_next ⇐ volume_occl_buf_prev ⇐ 1
partial_occl_buffer ⇐ 1
for all slice s in all slices do
    UpdateEyeBuffer(s) {Algorithm 2}
    UpdatePartialOcclusionBuffer(s) {Algorithm 4}
    UpdateVolumetricOcclusionBuffer(s) {Algorithm 3}
end for
CompositeBuffers() {Algorithm 6}
```

The extended algorithm, as outlined in Algorithm 1 and illustrated in Figure 2, is integrated into a slice-based volume renderer which computes volumetric occlusion effects [24].

The geometric structures, such as DTI fiber tracts or streamlines are rendered before the slice traversal to capture

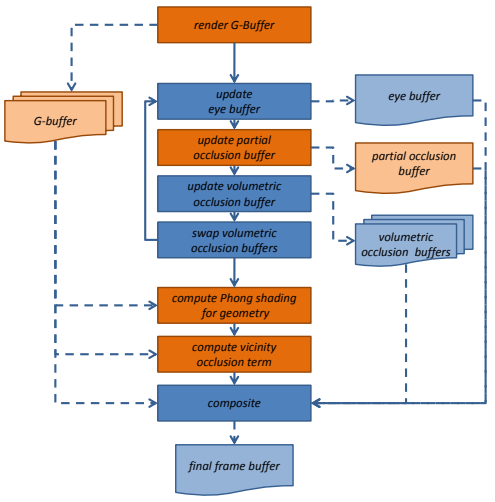


Fig. 2. The extensions to the original DOS method are denoted in orange: additional data structures (G-buffer, partial occlusion buffer) and additional processing steps (computing the G-buffer, updating the partial occlusion buffer, computing surface shading and determining the vicinity occlusion term)

their colors, positions, normals and depth values into images. Those images, commonly referred to as a G-Buffer [21], are used to defer computation of surface lighting and the vicinity occlusion term until after the slice traversal. Deferring those computations is necessary in order to incorporate the volumetric occlusion term and the shaded volume into the final compositing step.

As in the volumetric occlusion shading method, the volumetric occlusion buffer is used to store and compute the volumetric occlusion factor for each slice, additionally incorporating the occlusion from the geometry. An additional render pass then uses the volumetric occlusion buffer to update the partial occlusion buffer, storing the occlusion of the volume in front of the geometric structures. The eye buffer and the volumetric occlusion buffers get updated alternately during the slice-by-slice traversal of the volume.

The view-aligned slices are computed on the CPU and then rendered from the eye's point of view, projecting them into the eye buffer, during which the volume is shaded based on the volumetric occlusion buffer and the result of the transfer function lookup.

The volumetric occlusion information is updated by rendering the slice again, but now into the next volumetric occlusion buffer. During this pass, the occlusion factors of the previous volumetric occlusion buffer are integrated by reading and averaging multiple samples.

The slice is rendered another time during which the volumetric occlusion buffer is read and used to update the partial occlusion buffer with the combined volumetric and low frequency geometric occlusion. The current and next occlusion buffers are swapped and the algorithm proceeds to the next slice, until all slices have been processed.

Finally, the vicinity occlusion term for the geometric structures is computed and combined with the partial occlusion term and the G-Buffer during the final compositing.

3.2.1 Rendering the G-Buffer

Before traversing the slices, the geometry is rendered to store their colors, view space positions, view space normals and view space distance in the individual 2D images comprising the G-Buffer. The stencil buffer is used to mask all the fragments covered by the streamlines geometry in order to optimize updating of the partial occlusion buffer. The data sets discussed in Section 6 are of static nature due to their origin; this however is not a limitation of the presented method, since one could easily stream time varying geometry or even perform streamline tracing on the graphics card.

3.2.2 Updating the Eye Buffer

Algorithm 2 UpdateEyeBuffer(s)

```
BindTexture(volume_occl_buf_prev, geom_linear_depth)
SetRenderTarget(eye_buffer, geom_depth_stencil)
EnableDepthTest(PASSIFLESS,DONOTWRITEDEPTH)
EnableBlending(ONEMINUSDESTINATIONALPHA,ONE)
for all fragments f of s do
  (x, |∇x|) ⇐ Texture3D(volume,f )
  (σ_s, σ_t) ⇐ Texture2D(transfer_function, (x, |∇x|))
  occlusion ⇐ Texture2D(volume_occl_buf_prev, f_clip)
  delta = |f_viewz − Texture2D(geom_linear_depth,f_clip)|
  sampling_distance = min(slice_distance, delta)
  α ⇐ 1 − e^(−σ_t · sampling_distance)
  frame_buffer ⇐ (L_a · σ_s · occlusion · α, α)
end for
```

The eye buffer update, as detailed in Algorithm 2, is performed similar to the volumetric occlusion shading method. The volume and transfer function are evaluated and the resulting color value is modulated by the current volumetric occlusion buffer, additionally testing the front-to-back composited slices against the depth-buffer.

3.2.3 Computing the Volumetric Occlusion Term

Algorithm 3 UpdateVolumetricOcclusionBuffer(s)

```
BindTexture(volume_occl_buf_prev, geom_depth)
SetRenderTarget(volume_occl_buf_next)
DisableBlending()
for all fragments f of s do
  (x, |∇x|) ⇐ Texture3D(volume,f)
  σ_t ⇐ Texture2D(transfer_function, (x, |∇x|))
  if f_depth < Texture2D(geom_depth,f_clip) then
    σ_t ⇐ geometry_opacity
  end if
  occlusion ⇐ ComputeDOS(volume_occl_buf_prev, σ_t)
  frame_buffer ⇐ occlusion
end for
Swap(volume_occl_buf_prev, volume_occl_buf_next)
```

The occlusion factors of the previous volumetric occlusion buffer are integrated by reading and averaging multiple

samples, as described in [24]. An additional look up into the geometric depth image is used to determine whether the current slice sample is behind the geometric structures, and if so, a user specified, (high) opacity value is used to *replace* the opacity resulting from the volume/transfer function lookup in order to incorporate the occlusion of the geometric structure into the volume. This pass, as detailed in Algorithm 3, is computing the occlusion effects from the *volume* into the *volume* and from the *geometry* into the *volume*.

3.2.4 Computing the Partial Occlusion Term

Algorithm 4 UpdatePartialOcclusionBuffer(s)

`BindTexture`($volume\_occl\_buf_{prev}$,$volume\_occl\_buf_{next}$, $geom\_linear\_depth$)
`SetRenderTarget`($partial\_occl\_buffer$, $geom\_depth\_stencil$)
`DisableBlending`()
`EnableDepthTest`(PASSIFLESS,DONOTWRITEDEPTH)
`EnableStencilTest`() {update pixels that contain geometry}
`StencilFunc`(EQUAL, 1, ~ 0)
`StencilOp`(KEEP, DECREASE, KEEP)
for all fragments f of $s-1$ **do**
  $z_{surface} = \texttt{Texture2D}(geom\_linear\_depth, f_{clip})$
  $z_{slice} = f_{view_z}$, $z_\Delta = |z_{surface} - z_{slice}|$
  $z_{ratio} = \frac{\min(slice\_distance, z_\Delta)}{slice\_distance}$
  $occl_{prev} \Leftarrow \texttt{Texture2D}(volume\_occl\_buf_{prev}, f_{clip})$
  $occl_{next} \Leftarrow \texttt{Texture2D}(volume\_occl\_buf_{next}, f_{clip})$
  $partial\_occl = occl_{prev} \cdot (1 - z_{ratio}) + occl_{next} \cdot z_{ratio}$
  $frame\_buffer \Leftarrow partial\_occl$
end for

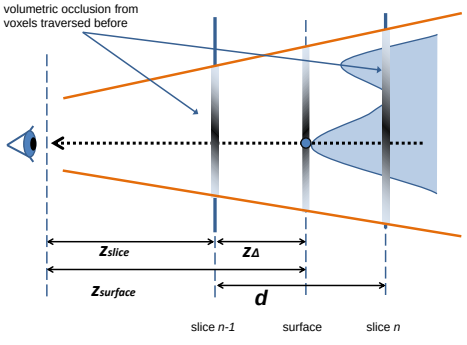


Fig. 3. Illustration of the basic geometric setup for updating the partial occlusion buffer of a geometric surface at a view space distance $z_{surface}$ which is situated between slices $slice_{n-1}$ (at view space distance z_{slice}) and $slice_n$, separated by the slice distance d. The partial occlusion of the surfaces is approximated by interpolating between $volumetric\_occl_{n-1}$ and $volumetric\_occl_n$ based on z_Δ.

The partial occlusion buffer is used to record the volumetric occlusion from the beginning of the volume up to the surface, as Figure 3 and Algorithm 4 illustrate. It is computed by interpolating between the volumetric occlusion $volumetric\_occl_n$ of the current slice and the volumetric occlusion $volumetric\_occl_{n-1}$ of the previous slice, in order to approximate soft shadows cast onto the surface at distance $z_{surface}$ in view space.

The *previous* slice $slice_{n-1}$ is rendered instead of the current slice $slice_n$ because the latter will be discarded by the depth test due to it being occluded by the surface. The interpolation factor z_{ratio} is determined by the difference z_Δ between the surface and the slice, as shown in Equations 1 and 2. The difference z_Δ is clamped by the slice distance d in order to correctly handle the case where there is no surface between two subsequent slices. The $partial\_occl_n$ of the current $slice_n$, is then computed as outlined in Equation 3:

$$z_\Delta = |z_{slice} - z_{surface}| \quad (1)$$

$$z_{ratio} = \frac{\min(z_\Delta, d)}{d} \quad (2)$$

$$partial\_occl_n = volumetric\_occl_{n-1} \cdot (1 - z_{ratio}) + volumetric\_occl_n \cdot z_{ratio} \quad (3)$$

The stencil buffer mask, which was created while computing the G-Buffer, as discussed in Section 3.2.1, is used to increase performance by restricting updating of the partial occlusion buffer to pixels containing actual surfaces. The partial occlusion buffer is repeatedly overwritten until fragments of a slice behind the surface are discarded by the depth test, at which point the stencil buffer will be cleared for that fragment, preventing further updates. The computation of the partial occlusion buffer completes after traversing last slice and as such then contains the occlusion shadows cast from the *volume* onto the *geometry*.

3.2.5 Computing the Vicinity Occlusion Term

Algorithm 5 VicinityOcclusion($geom\_linear\_depth$)

$occlusion \Leftarrow 0$
$z_{surface} \Leftarrow \texttt{Texture2D}(geom\_linear\_depth, f_t)$
$biased\_depth \Leftarrow z_{surface} + depth\_bias$
for all samples $\vec{p}_i$ within the vicinity occlusion extent R_t **do**
  project $\vec{p}_i$ into texture space $\vec{t}_i$ using $z_{surface}$
  $depth\_sample \Leftarrow \texttt{Texture2D}(geom\_linear\_depth, \vec{t}_i)$
  if $depth\_sample > 0 \wedge depth\_sample < biased\_depth$ **then**
    $occlusion \Leftarrow occlusion + 1$
  end if
end for
return $1 - \frac{occlusion}{|t_i|}$

The vicinity occlusion term is determined by rendering a full screen quad while comparing the depth of each pixel with the depths of neighboring pixels in the geometric depth buffer, counting neighboring pixels with greater depth as occluded, as outlined in Algorithm 5. The ratio of occluded to non-occluded pixels is an approximate geometric

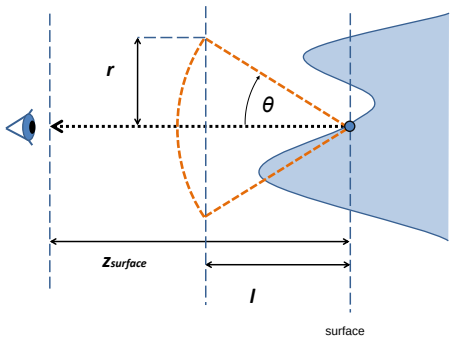


Fig. 4. Illustration of the basic geometric setup for computing the vicinity occlusion term of a geometric surface at the view space distance $z_{surface}$.

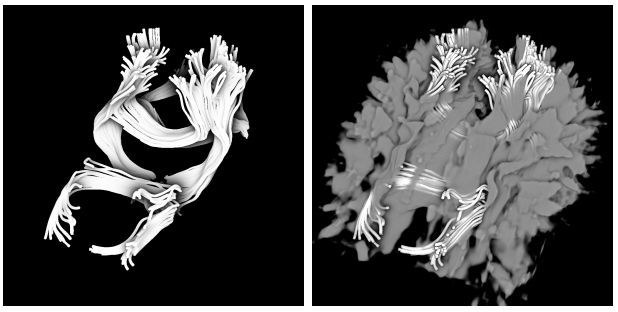
occlusion term for the high frequency geometric details. The depth value of the current pixel is offset to avoid self occlusion, similar in spirit to the depth bias used to prevent equivalent artifacts using shadow maps to compute shadows of solid geometry [33].

The directional occlusion shading model only considers the occlusion from structures within a cone with opening angle θ, as illustrated in Figure 4. An occluder with view space distance l from the surface must be within the base of the cone of radius $r = l \cdot \tan(\theta)$ in order to impact the surface at view space distance $z_{surface}$. The view space radius $r_{volumetric\_occl} = d \cdot \tan(\theta)$ used for blurring the previous volumetric occlusion buffer, is thus determined by the cone opening angle θ and the inter-slice distance d.

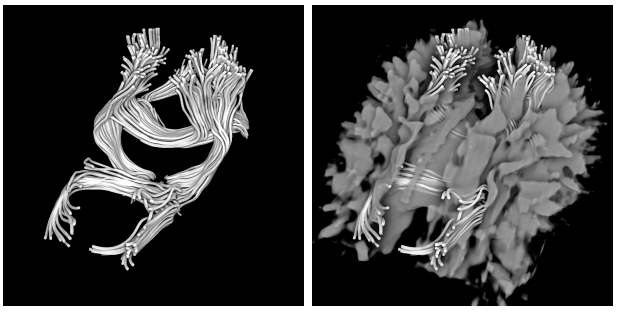
Computing the vicinity occlusion term, in contrast to computing the volumetric occlusion term, however does not intrinsically lead to values for l (and indirectly to the radius of influence r). Other screen space ambient occlusion techniques typically let the user specify r directly in order to change the size of the occlusion effects. This however would decouple the qualitative appearance of the vicinity occlusion effects from the volumetric occlusion effects. Instead, a "virtual" slice distance $l_{vicinity}$ is specified by the user and utilized to compute the vicinity occlusion extent $r_{vicinity\_occl} = l_{vicinity} \cdot \tan(\theta)$. A "virtual"slice distance $l_{vicinity} = 0.005$ was used in Figures 5(b), 5(d) and 5(f) to show occlusion effects both on the surface and the volume which change depending on the values of the blur angle $\theta \in \{25°, 45°, 65°\}$.

The view space vicinity occlusion extent $r_{vicinity\_occl}$ for a surface at view space distance $z_{surface}$ is then projected into image space by the (perspective) projection matrix P, potentially non-uniformly scaling objects, especially when the projection matrix aspect ratio is not corresponding to the aspect ratio of the viewport. This transformation of a circular radius of influence from view space into an ellipse in image space is automatically handled by the following computations.

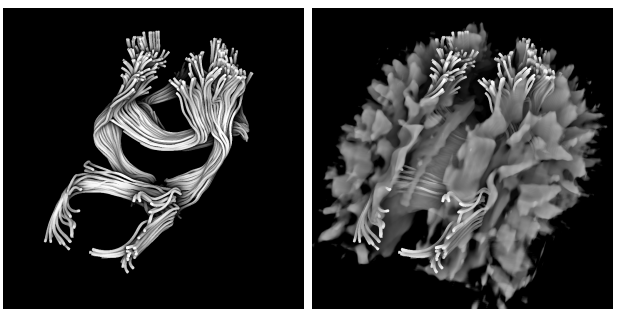
The view space radius of influence $r_{vicinity\_occl}$ is expressed as the homogeneous point R_{view} which then is projected from view space into clip space by the projection matrix P



(a) volumetric occlusion term (b) $\theta = 25°$



(c) vicinity occlusion term (d) $\theta = 45°$



(e) combined occlusion term (f) $\theta = 65°$

Fig. 5. Left column: DTI fiber tracts were rendered where the transfer function was set to render all voxels as transparent in order to emphasize the occlusion effects on the geometric structures. a) shows the volumetric occlusion term, c) shows the vicinity occlusion term and e) shows the combined occlusion term. The right column shows the combined occlusion term for various values of θ.

and then subsequently transformed into the $[0, 1]^2$ range of texture coordinates by the scale and bias matrix T, yielding the texture coordinate space radius of influence $\vec{R}_c$, which is shown in Equation 4. Equation 5 then shows texture space radius of influence $\vec{R}_t$ as the result of a perspective division:

$$\vec{R}_c = \begin{pmatrix} x_t \\ y_t \\ z_t \\ w_t \end{pmatrix} = T \cdot P \cdot \begin{pmatrix} r_{vicinity\_occl}(z_{surface}) \\ r_{vicinity\_occl}(z_{surface}) \\ z_{surface} \\ 1 \end{pmatrix} \quad (4)$$

$$\vec{R}_t = \begin{pmatrix} \frac{x_t}{w_t} \\ \frac{y_t}{w_t} \end{pmatrix} \quad (5)$$

The texture space circle of confusion $\vec{R}_t$, which is computed in the fragment shader since $z_{surface}$ varies per pixel, is used to compute texture coordinate offsets for sampling the buffer storing the view spaced depths of the geometric structures.

A number of N sample offsets $\vec{p}_i$ within the vicinity occlusion extent $\vec{R}_t$ are then generated and added to the projected texture coordinate $\vec{f}_t$ of the currently processed fragment f, yielding the set of texture space sample positions $\vec{t}_i$, as Equation 6 shows:

$$\vec{t}_i = \vec{f}_t + \vec{p}_i \tag{6}$$

Incrementally filtering the volumetric occlusion buffers allows minimization of the number of samples since each individual filter step only covers a rather small number of texels of the previous volumetric occlusion buffer. In contrast, computing the vicinity occlusion requires the consideration of a much larger area in the buffer containing the view space distances of the geometry. Using a Poisson distribution [6] with up to 118 samples proved to sufficiently capture the high frequency surface-to-surface occlusion effects and were thus used generating the results presented in Section 6. The vicinity occlusion term is capturing the occlusion effects received by the *geometry* from the surrounding *geometry*.

Other, more general screen space ambient occlusion techniques geared towards high performance shading of screen covering geometric scenes often employ complex approaches in order to achieve high visual quality at high frame rates. The vicinity occlusion term however is only computed for pixels that contain geometric structures; pixels not covered by the geometric structures are shaded by the direct volume rendering term, which is the major performance limiting part of the presented algorithm. This, combined with the sparsity of the DTI fiber tracts and streamlines reduces the contribution of the vicinity occlusion term to the overall performance and allows an easy to implement vicinity occlusion term that also emphasizes the high frequency details of the geometric structures.

This approach works especially well for the high frequency tube models, since those are typically bundled close together, where the vicinity occlusion term is capturing their mutual occlusion effects with sufficient fidelity. This is not always the case when considering general geometry, which often cover large spans of the domain. The vicinity occlusion term is not adequately considering those, due to its local nature; more general screen space ambient occlusion approaches should be used to for those types of geometry. Changing the blur angle of the volumetric occlusion term can also capture the occlusion effects of remote structures to a certain extent, namely those that are within the blur cone.

3.2.6 Final Compositing and Shading

The last step of our proposed shading method is to combine the partial occlusion term with the vicinity occlusion term in order to yield a final combined occlusion term for shading the geometric surfaces. Figure 5(a) shows the volumetric occlusion term that captures only the shadows of *low frequency* geometric details plausibly, which is in contrast to the vicinity occlusion term (Figure 5(c)) that approximates occlusion effects for *high frequency* local geometric structures. The combination of the volumetric occlusion term containing occlusion effects from the volume and the low frequency geometric structures with the vicinity occlusion term then provided for both local, *high frequency* and global *low frequency* occlusion effects, as demonstrated in Figure 5(e).

We experimented with two approaches, specifically using the minimum or the product of both occlusion terms, as Equations 7 and 8 show:

$$\begin{aligned} combined\_occl &= \min(partial\_occl, vicinity\_occl) && (7)\\ combined\_occl &= partial\_occl \cdot vicinity\_occl && (8) \end{aligned}$$

Multiplying the volumetric and the vicinity occlusion terms has the advantage over taking their minimum, since it allows the vicinity occlusion terms of partially occluding voxels to continue providing depth cues on the surface, as Figures 6(a) and 6(b) demonstrate, and as such was used to create the results presented in Section 6. This is especially important for geometry embedded inside homogeneous regions of the volume, since there, the volumetric occlusion term is rather low, thus effectively discarding any variation in contrast when "minimum" combining it with the vicinity occlusion term. However, the product between the volumetric and vicinity occlusion terms will maintain some subtle variation in image contrast in those homogeneous regions.

The G-Buffer is then used to compute the *amb*, *diff* and *spec* terms of the surface lighting, which are then blended together with the *combined_occl* term and the *surface_color*, as summarized by Equation 9 and detailed in Algorithm 6:

$$\begin{aligned} shaded\_surface &= combined\_occl \cdot surface\_color \cdot \\ &\quad (amb + diff) + spec \end{aligned} \tag{9}$$

The *shaded_surface* is composited on top of the shaded volume ($volume\_color_{rgb}$ and $volume\_color_a$), which is shown in Equation 10:

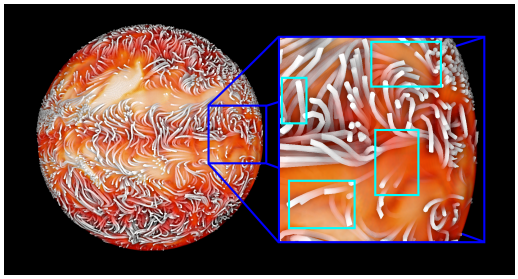
$$\begin{aligned} final\_color &= shaded\_surface \cdot (1 - volume\_color_a) \\ &+ volume\_color_{rgb} \end{aligned} \tag{10}$$

The final color *final_color* is then copied into the frame buffer of the window.

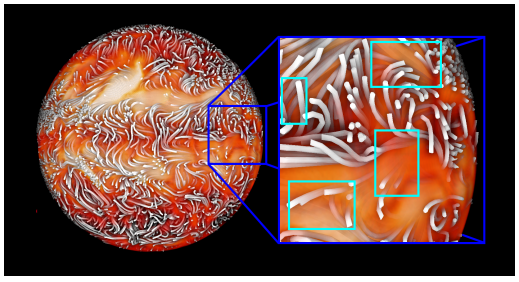
4 Computing the Geometry for the Streamtubes

This section discusses the generation of the streamtubes which are used to represent and render the streamlines.

Given a volumetric dataset $\Omega \subset \mathbb{R}^3$ and an associated vector field $\mathbf{V} : \Omega \to \mathbb{R}^3$, a streamline is defined as the path a particle $\mathbf{p} \in \Omega$ takes along $\mathbf{V}$. It either ends at a sink point or when the extent of Ω is reached. To compute the streamline emanating at $\mathbf{p}$, one has to solve the ODE $\partial \mathbf{x}(t)/\partial t = \mathbf{V}(\mathbf{x}(t))$, where $\mathbf{x}(0) = \mathbf{p}$. In the context of visualization, as we have it here, this ODE is generally discretized in



(a) minimum, Equation 7



(b) product, Equation 8

Fig. 6. Illustration of different ways of combining the volumetric and vicinity occlusion terms. a) The minimum of the volumetric and the vicinity occlusion term is used as the combined occlusion term, reducing the impact of the vicinity occlusion terms in the regions where the surface is occluded by partially transparent voxels. b) The volumetric and the vicinity occlusion terms are multiplied together in order to yield the combined occlusion term, maintaining the depth cues of the surface through in the partially occluded voxels.

Algorithm 6 CompositeBuffers

```
SetRenderTarget(window_frame_buffer)
for all fragments f of full screen quad do
  volume_color ⇐ Texture2D(eye_bufferf_clip)
  position ⇐ Texture2D(geom_position, f_clip)
  normal ⇐ Texture2D(geom_normal, f_clip)
  surface_color ⇐ Texture2D(geom_color, f_clip)
  partial_occl ⇐ Texture2D(partial_occl_buffer, f_clip)
  if f covers geometry then
    vicinity_occl ⇐ VicinityOcclusion(geom_linear_depth)
    {Algorithm 5}
    occlusion ⇐ vicinity_occl · partial_occl
    (amb, diff, spec) ⇐ Phong(position , normal )
    shaded_surface ⇐ occlusion · surface_color · (amb +
    diff) + spec
    frame_buffer ⇐ (shaded_surface · (1 −
    volume_color_a) + volume_color_rgb, 1)
  else
    frame_buffer ⇐ volume_color
  end if
end for
```

respect to time t using Euler's method, i.e., one evaluates $\mathbf{x}_{i+1} = \mathbf{x}_i + \varepsilon\,\mathbf{V}(\mathbf{x}_i)$, where $i \geq 1$ and $\mathbf{x}_1 = \mathbf{p}$. To reduce the integration error, ε is usually set to a small value, appropriate to the given dataset Ω. This results in a piecewise linear curve with coefficients $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n\}$ representing the streamline. To simplify the following discussion, let us assume that each line segment has equal length, i.e., $||\mathbf{x}_2 - \mathbf{x}_1||_2 = \ldots = ||\mathbf{x}_{i+1} - \mathbf{x}_i||_2 = \ldots = ||\mathbf{x}_n - \mathbf{x}_{n-1}||_2$.

From $\mathbf{X}$, a streamtube is constructed in three steps:

1) A parametric curve $\gamma : [0,1] \rightarrow \mathbb{R}^3$ is computed approximating the coefficients in $\mathbf{X}$.
2) A parametric surface $\sigma : [0,1] \times [0,1] \rightarrow \mathbb{R}^3$ is generated by sweeping a circle along $\gamma(u)$.
3) The final streamtube is generated by uniformly triangulating $\sigma(u,v)$.

To reduce the number of patches which need to be tessellated, $\gamma(u)$ and hence $\sigma(u,v)$ should only approximate the data $\mathbf{X}$. $\gamma(u)$ is chosen to be a $C^{(2)}$ B-spline curve, because higher order curve properties are needed to construct $\sigma(u,v)$. $\sigma(u,v)$ is chosen to be a $C^{(2)}$ B-spline surface because it enables one to evaluate visually pleasing smooth surface normals and geometry. The books [4], [19] give a detailed overview of B-spline curves and surfaces, as they are used in this paper. The final streamtube mostly depends on the construction of $\gamma(u)$, therefore, step 1 is discussed in more detail in Section 4.1.

Step 2 constructs $\sigma(u,v)$ by sweeping a circle along $\gamma(u)$, i.e., $\sigma(u,v)$ is a swept surface (see [4]). The swept circle is oriented along the Frenet frame of $\gamma(u)$. Note that the normal of the Frenet frame flips when an inflection point of $\gamma(u)$ is reached, i.e., a consistent orientation of the Frenet frame has to be maintained in order to avoid severe twists in the resulting $\sigma(u,v)$. To avoid self-intersections in the streamtube, the radius of the circle should be chosen to be smaller than $1/\kappa_{max}$, where κ_{max} is the maximum curvature of $\gamma(u)$. Also note that both, $\gamma(u)$ and $\sigma(u,v)$ have the same parameterization in u, implying that both have the same number of patches in the parameter direction u.

Step 3 constructs the streamtube by uniformly triangulating a uniform grid of samples for each surface patch of $\sigma(u,v)$ (see [4]). The main advantage of this tessellation strategy is that the resolution of the grid only depends on the degree of $\sigma(u,v)$, i.e., it is independent of the length of $\gamma(u)$ and also makes sure that all features of the approximated streamline are captured.

4.1 Streamline approximation

The goal of this step is to compute a B-spline curve $\gamma(u)$ approximating the coefficients in $\mathbf{X}$ from which $\sigma(u,v)$ and the final streamtube is constructed.

Various interpolation or approximation methods exist which fit a curve to a set of samples such as $\mathbf{X}$. In the context of visualization, an approximation method is often desired, since interpolation methods generally introduce features (e.g., oscillations) into the resulting interpolant which were not present in the true curve, $\mathbf{x}(t)$, from which the samples

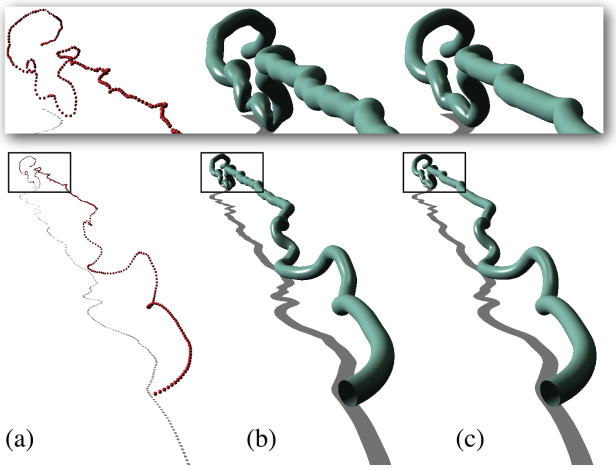


Fig. 7. Streamline approximation: (a) Sampled streamline; (b) Least-Square Approximation; (c) Our method

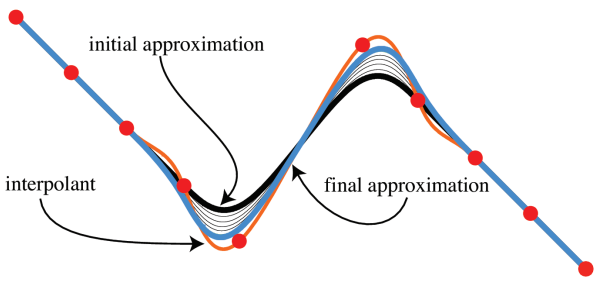


Fig. 8. Interpolation (orange) may introduce oscillations. Our scheme moves an initial curve (black) iteratively closer to the input points (blue). Intermediate curves are shown as well (thin, black curves).

were generated. However, even with an approximation method such as least-squares approximation (see [19]), perturbations could be introduced in the resulting curve approximation producing wriggles in the final streamtube. For an example, the reader is referred to Figure 7b. In many applications, these artifacts do not present a significant issue. However, in a visualization scenario, firstly, these wriggles challenge the generation of the tubular geometry, and secondly, streamlines are visualized which contain features which are not present in the original data.

Based on these observations we propose the following approximation method. Following the method introduced in [14], an initial cubic ($C^{(2)}$) B-spline curve $\gamma_1(u)$ is constructed with the samples in $\mathbf{X}$ as its coefficients. Given the variation diminishing properties of B-splines, $\gamma_1(u)$ approximates $\mathbf{X}$ without introducing new curve perturbations (Schönberg's approximation, see [4]). Then, as illustrated in Figure 8, the coefficients of $\gamma_1(u)$ are iteratively updated so that the curve moves closer to the samples in $\mathbf{X}$ till a user-specified accuracy is achieved. The reader is referred to [14] for the details of this iterative curve fitting algorithm and its convergence properties.

For the shown datasets in this paper, $\mathbf{X}$ generally contains more samples than necessary resulting in a large number of coefficients in $\gamma_1(u)$. Therefore, starting from $\gamma_1(u)$, we apply the following data reduction scheme to compute the final $\gamma(u)$.

The intermediate curve $\gamma_i(u)$ with $i > 1$ is computed by applying the approximation algorithm as discussed above to the set of samples $\mathbf{X}_i$. $\mathbf{X}_i$ is constructed by regularly sampling the parameter space of $\gamma_{i-1}(u)$ with $m_i = \lfloor n/2^{i-1} \rfloor$ samples, i.e., at each data reduction step the number of samples is halved, where $\gamma_{i-1}((j-1)/(m_i-1))$ is the jth sample in $\mathbf{X}_i$. Note that at each step a $C^{(2)}$ curve is sampled, smoothing out high-frequency features, but due to the applied approximation algorithm, as discussed above, the curve approximates the data reasonably well. The data reduction scheme terminates when $\varepsilon_{\max} > \varepsilon$, where $\varepsilon_{\max} := \max_{1 \le j \le n} ||\mathbf{x}'_j - \mathbf{x}_j||_2$. $\mathbf{x}'_j$ is the projection of $\mathbf{x}_j \in \mathbf{X}$ onto $\gamma_i(u)$ and $||\cdot||_2$ is the L_2-norm.

The curve which is then used for the subsequent streamtube generation steps is $\gamma_k(u)$, i.e., $\gamma(u) := \gamma_k(u)$, where k is the number of iterations. For the datasets shown in this paper, $k < 4$ with $\varepsilon = 10^{-4}$. Figure 7 shows the streamtube which was constructed based on this curve fitting method applied to a sampled streamline. The figure also shows the equivalent streamtube generated using least-square approximation. The reduction of wriggles in comparison to least-squares is visually apparent. Figure 9 shows a streamline dataset, where the tubular geometry was generated from curves which were generated using least-squares versus our proposed method. As seen in the figure, the wriggles in the least-square curve approximation causes unwanted visual artifacts which are potentially amplified whithin an ambient occlusion context.

It has to be noted that since the number of coefficients of the final curve is significantly smaller than the number of initial points, a compromise between approximation accuracy and introduction of additional features has to be made. Our method iteratively computes an approximating curve by maintaining features which are present in the original data as much as possible till a user-specified accuracy has been achieved. Incorporating uncertainty information in the approximated curve representation will be addressed in future work.

5 Anti-Aliasing

The generated stream tubes have a high amount of geometric detail, and as such suffer from aliasing of both the geometry, but also of the specular highlights, as demonstrated in Figure 10. Graphics hardware provides multi-sampling to reduce the aliasing at polygon edges, which however is not very well suited to the aliased specular highlights, since those are introduced by the fragment shader itself. However, recent GPU generations allow to render the geometry into multi-sampled textures where attributes, such as normals or positions, can be optionally evaluated at each sub-sample of a fragment. We employ such a multi-sampled G-buffer with per sample normals and view positions in order to compute the specular lighting during compositing at higher effective resolution, thus reducing the specular aliasing, as

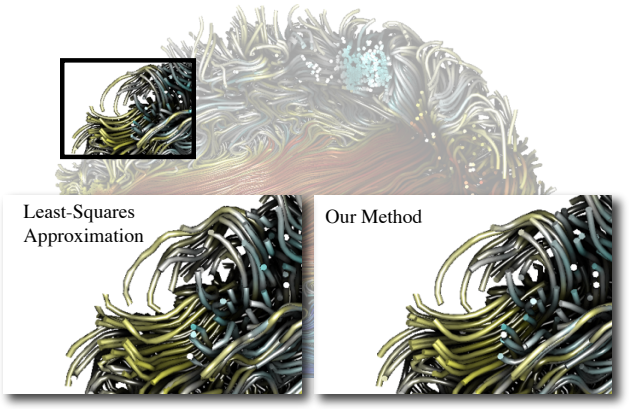


Fig. 9. Detail of our streamline approximation, compared to a Least-Square Approximation

Figures 10(a) and 10(b) illustrate. Another anti-aliasing method, *Fast Approximate Anti-Aliasing* (FXAA) [13], instead operates on a color buffer and performs edge-detection with perceptually correct edge filtering. This approach has the benefit of also providing anti-aliasing of the volume, since it can be applied to the final composited image, as Figure 10(c) shows. The anti-aliasing quality can be improved even further, by combining both anti-aliasing methods, as Figure 10(d) demonstrates.

6 Results and Discussion

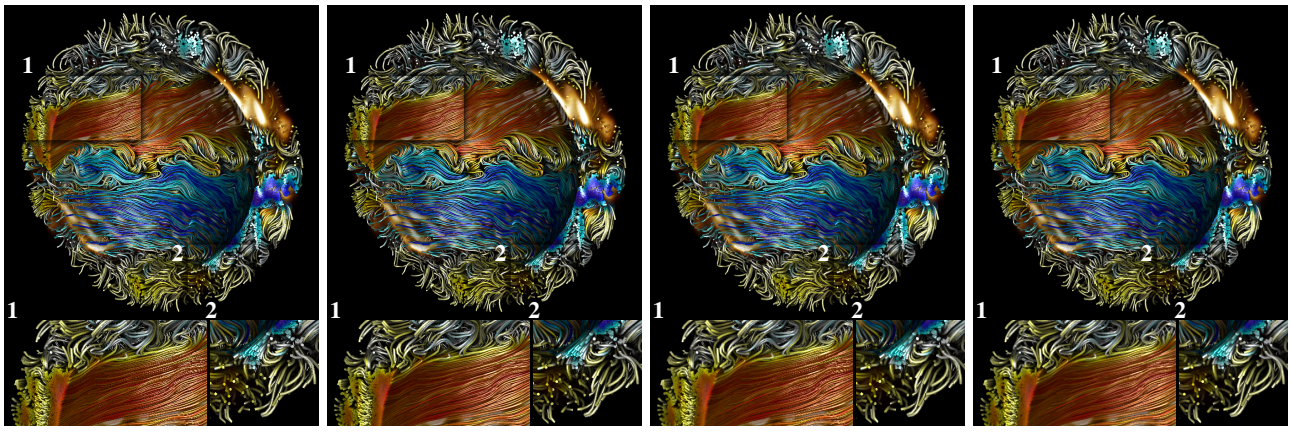
The method was implemented using OpenGL and Cg running on an NVIDIA GeForce GTX 580 GPU with 1.5 GB of video memory. The images were rendered into 16-bit precision floating-point buffers. The tube geometry was computed by sweeping a circle along a B-spline curve approximating the individual line segments. The surface positions, colors and normals were evaluated on the CPU along the curve and then stored in video memory.

Figure 11 shows a chemical simulation data set where two chemical reagents are mixed in order to accelerate their reaction. The streamlines were color mapped with the mixture fraction denoting the relative ratio of two chemical agents shown in red and blue. A 2D transfer function was used to highlight the boundaries of the mixing pipe colored white, while at the same time showing the concentration of the product as the result of the chemical reaction. Figure 11(a) shows the combined data set using volumetric Phong shading with on-the-fly gradient estimation together with streamlines rendered with Phong surface shading which make it difficult to gain insight into the complex turbulent behavior of the streamlines. Figure 11(b) combines the vicinity occlusion term with the volumetric occlusion of the scalar field to obtain mutual occlusion effects which provide additional context between the mixture fraction, visualized on the streamlines, and the concentration of the chemical product, visualized by the volume rendered scalar field.

Figure 12 shows an MRI data set with registered DTI fiber tracts rendered without occlusion effects in Figure 12(a), which makes it hard to gain insight about the relative spatial arrangement between the geometric structures and the volumetric features. Figure 12(b) uses the combined occlusion shading to render physically plausible occlusion effects which impact both the volume and the geometry, thus enhancing the depth perception and visual comprehension of the data set. Being able to precisely reason about the arrangement of structures (neural fibers, sensitive tissue) within the brain is of key importance to surgical planning, where surgical instruments must be placed with uttermost precision, since minimal deviations could damage sensitive tissues, especially when considering brain surgery.

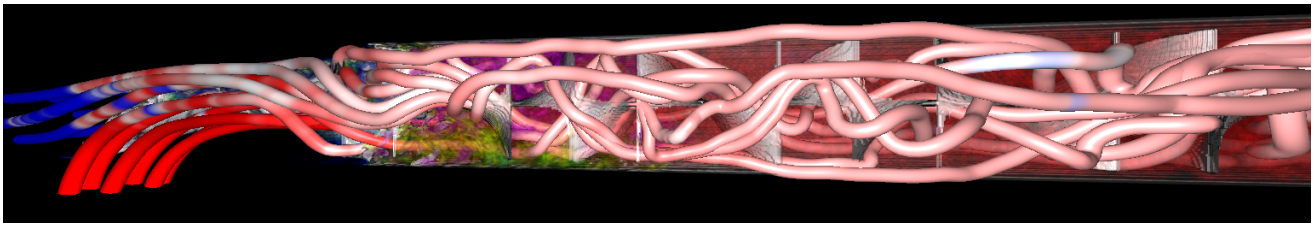
Figure 13 demonstrates a single time step of an astrophysical data set showing the magnetic field of a Sun-like star undergoing an inversion of its magnetic poles [3]. The magnetic field lines were color mapped with the polarity of the longitudinal magnetic field with red and blue showing positive and negative polarity respectively. The scalar volume shows the magnitude of the magnetic field mapped to red, orange and white. Figure 13(a) shows the combined data set rendered with volumetric Phong shading, which makes it difficult to gain insight into the complex configuration of the magnetic field lines with respect to the magnetic field strength. Volumetric Phong shading is also dependent on the volume gradient for its shading, which is problematic for homogeneous or noisy regions since there, the gradients are either not defined, or point into incoherent directions, as observable in the top right part of Figure 13(a). The combined rendering with occlusion effects, as shown in Figure 13(b), visualizes the volume showing the magnetic field strength together with the magnetic field lines illustrating the polarity of the magnetic field on the field lines themselves. The occlusion effects allow better comprehension of the structure of the field lines, while at the same time benefiting from the context providing scalar volume. Its independence from the volumetric gradient avoids the artifacts of using volumetric Phong shading.

Table 1 shows the performance behavior of the combined geometric and volumetric occlusion shading method compared to volume shading with an emission-absorption or Phong shading model with on-the fly gradient estimation. The performance of the combined shading method is reduced compared to the traditional volume shading methods, which is similar to the performance behavior of the original purely volumetric occlusion shading method. Introducing the geometric structures into the occlusion shading computation reduces the performance by about 25% compared to computing volumetric occlusion shading alone, by about 86% compared to computing emission/absorption and by 71% compared to computing combined Phong shading. Despite this, sufficiently high performance is maintained and as such allows interactive exploration of the data sets. The computation of the vicinity occlusion term alone performs at interactive frame rates, despite the high number of samples taken and does not constitute the performance limiting aspect of the presented combined shading method, which instead is the incremental filtering inherent to the directional occlusion shading method. The image space approach for computing the vicinity occlusion term is especially beneficial for the

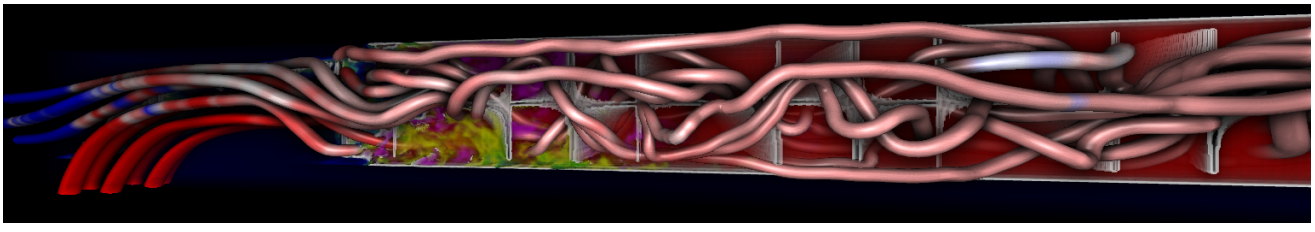


(a) no AA (b) MSAA (c) FXAA (d) MSAA + FXAA

Fig. 10. Comparison of a) no anti-aliasing, b) multi-sample anti-aliasing with per-sample normal evaluation, c) FXAA and d) multi-sample anti-aliasing with per-sample normal evaluation combined with FXAA.



(a) Phong volume shading with Phong surface shading



(b) combined occlusion shading with Phong surface shading

(c) color map for the mixture fraction of the two reagents

(d) color map for the concentration of the chemical product

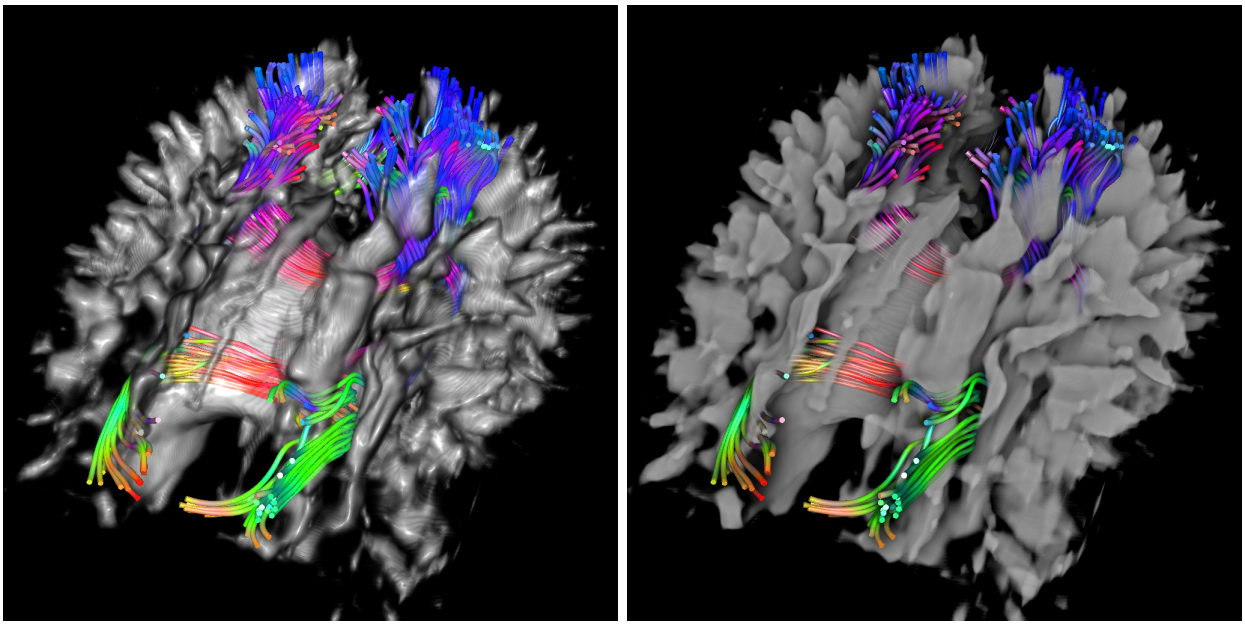
Fig. 11. Stream lines (127k triangles) of a mixing pipe and a scalar volume ($1800 \times 121 \times 121$ resolution), of which the front half was removed by a clipping plane. The images were rendered with a) Phong volume shading with on-the-fly gradient estimation and Phong surface shading and b) the presented combined surface and volumetric occlusion shading method. Figure c) shows the color map of the steam lines illustrating the mixture fraction and Figure d) shows the color map of volume which displays the concentration of the resulting chemical product.

astrophysical data set with its 7.9 million triangles which renders only at half of the speed, compared to the DTI data set which has a fifth of the number of triangles. Noticeable however is the lack of scaling for the mixing pipe data set for the GeForce GTX 580 GPU, indicating that the bottle neck is in other parts of the system, possibly caused by the internal memory layout of the anisotropic 3D texture used to store the volume.

7 Conclusion and Future Work

An extension to the volumetric directional occlusion shading method has been presented which renders occlusion effects of geometric structures combined with those of volumetric origin. Both geometric and volumetric structures act as shadow casters and shadow receivers and their shadows are computed by modifying the occlusion buffer update of the original directional occlusion shading method [24].

An additional surface based vicinity occlusion term is computed to capture high frequency shadowing effects between fine detailed geometric structures, such as streamlines



(a) Phong volume shading with Phong surface shading (b) combined occlusion shading with Phong surface shading

Fig. 12. DTI fiber tracts (720k triangles) and a registered MRI volume ($96 \times 96 \times 81$ resolution) rendered with a) Phong volume shading with on-the-fly gradient estimation and Phong surface shading and b) combined directional occlusion shading and Phong surface shading.

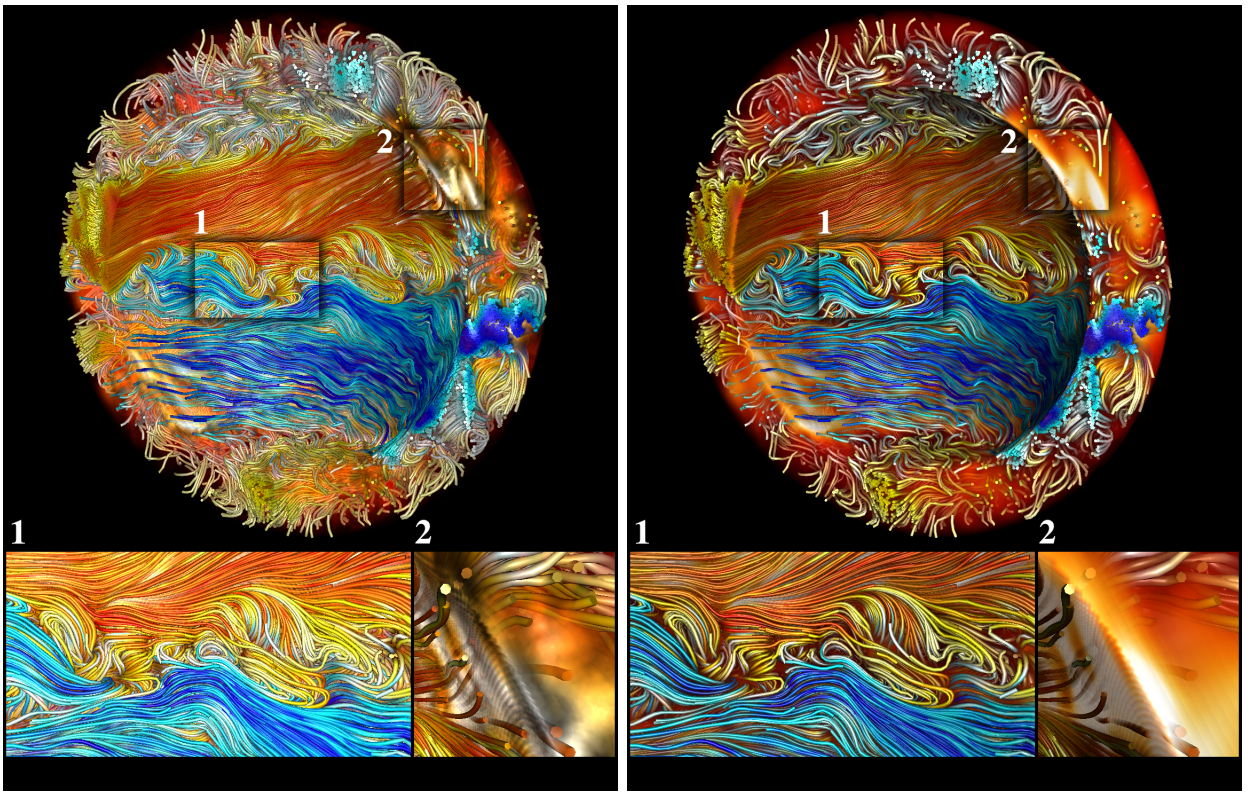
TABLE 1
Performance for various rendering methods and data sets, in FPS on an NVIDIA GeForce GTX 580 GPU.

data set	mixing pipe (Figure 11)		DTI fibers (Figure 12)		astrophysical (Figure 13)	
slices, resolution, triangles	574, $1800 \times 121 \times 121$, 137k		512, $96 \times 96 \times 81$, 720k		961, $256 \times 512 \times 512$, 7.9M	
θ, volumetric samples, vicinity samples	45°, 2×2, 118		45°, 4×4, 81		65°, 2×2, 81	
image resolution	768×128	1153×256	512×512	1024×1024	512×512	1024×1024
vicinity occlusion shading only	102.87	102.35	106.66	103.78	67.05	54.41
volumetric occlusion shading only	16.75	16.67	11.24	3.49	10.61	5.78
combined occlusion shading	11.73	11.79	9.73	3.09	7.31	4.54
emission/absorption & Phong	101.26	101.1	82.61	26.9	36.31	22.75
volumetric Phong & Phong	101.93	51.76	32.22	8.82	18.41	9.43

or DTI fibers, thus supplementing the volumetric occlusion term which captures the low frequency occlusion effects of the geometry and volume contained in the scene.

The extended method, similar to the volumetric directional occlusion shading algorithm, does not rely on precomputation and thus enables interactive manipulation of camera position, transfer function and the geometry as well, which are all taken into account during the occlusion computation. The required extensions reduce the rendering performance by about 25% compared to only computing the occlusion shading of the volumetric structures and as such allow interactive exploration of geometric and volumetric data sets while increasing the depth perception and scene comprehension. Stream tube geometries are computed from stream lines by an effective approach that reduces wiggles and orientation flips inherent to other approaches. Various hardware anti-aliasing approaches were discussed and used to reduce the aliasing inherent to the geometric detail and per-pixel lighting of the stream tubes.

In the future, we would like to allow the user to change the light position instead of itbeing fixed at the viewer. The multi-directional occlusion shading method by Šoltészová *et al.* [31] could be incorporated into the combined rendering method by changing the filtering of the volumetric occlusion buffer appropriately. Similarly, weighting of the samples could be done for the vicinity occlusion term as well in order to qualitatively match the shadowing of the volume and the geometry. Alternatively, a more generic soft shadow method could be used in order to compute soft shadows for the high detail geometry [8]. It would also be interesting to apply the presented method to combined data sets with arbitrary geometry and evaluate the effects of large scale geometric structures to the vicinity occlusion term. Another venue for future research would be a GPU-based implementation of our stream tube approximation to facilitate the visualization of time-dependent data sets.



(a) Phong volume shading with Phong surface shading

(b) combined occlusion shading with Phong surface shading

(c) color map for polarity of the magnetic field

(d) color map for the magnetic field magnitude

Fig. 13. An astrophysical data set was used to compute magnetic field lines (7.9M triangles) with Phong surface shading and a scalar volume ($256 \times 512 \times 512$ resolution) showing the magnitude of the magnetic field, rendered with a) Phong volume shading with on-the-fly gradient estimation and Phong surface shading, and b) combined directional occlusion shading and Phong surface shading. Figure c) shows the color map for polarity of the magnetic field and Figure d) shows the color map for the magnetic field magnitude.

ACKNOWLEDGMENTS

This research was sponsored by the National Nuclear Security Administration under the Advanced Simulation and Computing program through DOE Cooperative Agreement #DE-NA0000740, and by Award No.KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST), and DOE SciDAC-2:SDAV, NSF OCI-0906379.

REFERENCES

[1] L. Bavoil and M. Sainz. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH 2009: Talks*, SIGGRAPH '09, pages 45:1–45:1, New York, NY, USA, 2009. ACM.

[2] L. Bavoil, M. Sainz, and R. Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*, SIGGRAPH '08, pages 22:1–22:1, New York, NY, USA, 2008. ACM.

[3] B. P. Brown, M. S. Miesch, A. S. Browning, M. K. andBrun, and J. Toomre. Magnetic Cycles in a Convective Dynamo Simulation of a Young Solar-type Star. *The Astrophysical Journal*, 731:69–+, Apr. 2011.

[4] E. Cohen, R. F. Riesenfeld, and G. Elber. *Geometric modeling with splines: an introduction*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[5] J. Díaz, H. Yela, and P. Vázquez. Vicinity occlusion maps: Enhanced depth perception of volumetric models. In *Computer Graphics International*, 2008.

[6] D. Dunbar and G. Humphreys. A spatial data structure for fast poisson-disk sample generation. *ACM Trans. Graph.*, 25(3):503–508, 2006.

[7] M. H. Everts, H. Bekker, J. B. T. M. Roerdink, and T. Isenberg. Depth-dependent halos: Illustrative rendering of dense line data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1299–1306, 2009.

[8] J. M. Hasenfratz, M. Lapierre, N. Holzschuch, F. Sillion, and A. GRAVIR/IMAG-INRIA. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, 2003.

[9] J. Huang, T. Boubekeur, T. Ritschel, and M. H. E. Eisemann. Separable approximation of ambient occlusion. In *Short paper at Eurographics*, 2011.

[10] V. Interrante and C. Grosch. Strategies for effectively visualizing 3d flow with volume lic. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 421–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

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

[12] B. J. Loos and P.-P. Sloan. Volumetric obscurance. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 151–156, New York, NY, USA, 2010. ACM.

[13] T. Lottes. Fxaa. Technical report, 2011.
[14] T. Martin, E. Cohen, and R. Kirby. Volumetric parameterization and trivariate b-spline fitting using harmonic functions. *Computer Aided Geometric Design*, 26(6):648 – 664, 2009. Solid and Physical Modeling 2008, ACM Symposium on Solid and Physical Modeling and Applications.
[15] M. McGuire, B. Osman, M. Bukowski, and P. Hennessy. The alchemy screen-space ambient obscurance algorithm. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 25–32, New York, NY, USA, 2011. ACM.
[16] Z. Melek, D. Mayerich, C. Yuksel, and J. Keyser. Visualization of fibrous and thread-like data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1165–1172, 2006.
[17] M. Mittring. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.
[18] D. Patel, S. Bruckner, I. Viola, and E. Gröller. Seismic volume visualization for horizon extraction. In *PacificVis*, pages 73–80, 2010.
[19] L. Piegl and W. Tiller. *The NURBS book (2nd ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.
[20] C. Reinbothe, T. Boubekeur, and M. Alexa. Hybrid ambient occlusion. *EUROGRAPHICS 2009 Areas Papers*, pages ??–??, 2009.
[21] T. Saito and T. Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, pages 197–206, New York, NY, USA, 1990. ACM.
[22] M. Schott, T. Martin, A. Grosset, C. Brownlee, T. Hollt, B. Brown, S. Smith, and C. Hansen. Combined surface and volumetric occlusion shading. In *Pacific Visualization Symposium (PacificVis), 2012 IEEE*, pages 169 –176, 03 2012.
[23] M. Schott, A. Pascal Grosset, T. Martin, V. Pegoraro, S. T. Smith, and C. D. Hansen. Depth of field effects for interactive direct volume rendering. In *Computer Graphics Forum (Proceedings of Eurographics/IEEE VGTC Symposium on Visualization 2011)*, volume 30, pages 941–950, 2011.
[24] M. Schott, V. Pegoraro, C. Hansen, K. Boulanger, and K. Bouatouch. A directional occlusion shading model for interactive direct volume rendering. In *Computer Graphics Forum (Proceedings of Eurographics/IEEE VGTC Symposium on Visualization 2009)*, volume 28, pages 855–862, 2009.
[25] G. Schussman and K.-L. Ma. Scalable self-orienting surfaces: A compact, texture-enhanced representation for interactive visualization of 3d vector fields. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 356, Washington, DC, USA, 2002. IEEE Computer Society.
[26] G. Schussman and K.-L. Ma. Anisotropic volume rendering for extremely dense, thin line data. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 107–114, Washington, DC, USA, 2004. IEEE Computer Society.
[27] P. Shanmugam and O. Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *In I3D Š07: Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM*. Press, 2007.
[28] C. Stoll, S. Gumhold, and H.-P. Seidel. Visualization with stylized line primitives. *Visualization Conference, IEEE*, 0:88, 2005.
[29] A. Stompel, E. B. Lum, and K.-L. Ma. Visualization of multidimensional, multivariate volume data using hardware-accelerated non-photorealistic rendering techniques. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 394, Washington, DC, USA, 2002. IEEE Computer Society.
[30] F. V. Three-Dimensional, G. shi Li, U. D. Bordoloi, and H. wei Shen. Chameleon: An interactive texture-based rendering framework. In *in Proceedings IEEE Visualization 2003. IEEE, 2003*, pages 241–248, 2003.
[31] V. Šoltészová, D. Patel, S. Bruckner, and I. Viola. A multidirectional occlusion shading model for direct volume rendering. *Computer Graphics Forum*, 29(3):883–891, J 2010.
[32] A. Wenger, D. F. Keefe, S. Zhang, and D. H. Laidlaw. Interactive volume rendering of thin thread structures within multivalued scientific datasets. *IEEE Transactions on Visualization and Computer Graphics*, 10:2004, 2003.
[33] L. Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, 1978.
[34] M. Zöckler, D. Stalling, and H.-C. Hege. Interactive visualization of 3d-vector fields using illuminated stream lines. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 107–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

Mathias Schott received his undergraduate degree in Computer Science (Diplom-Informatiker FH) in 2005 from the University of Applied Sciences Schmalkalden in Germany. In 2011 he earned a PhD degree in Computing: Graphics and Visualization from the University of Utah in Salt Lake City. Currently he is a Developer Technology Engineer at NVIDIA corporation. His research interests include volume rendering and hardware accelerated computer graphics and visualization.

Tobias Martin received the undergraduate degree in computer science (Diplom-Informatiker FH) from the University of Applied Sciences Furtwangen, Germany, in 2004. He received his PhD degree in Computer Science from the University of Utah, Salt Lake City, in 2012. Currently he is a postdoctoral fellow at the Computer Graphics Laboratory at ETH Zürich. His research interests are computer graphics, geometric modeling, rendering, and visualization.

A.V. Pascal Grosset received his B.Sc. in Computer Science at the University of Mauritius and his M.Sc. in Computer Graphics at the University of Teesside, England. Currentlym he is a PhD student in Computing: Graphics and Visualization at the University of Utah. His research interests include computer graphics, visualization and GPGPU. He is a student member of IEEE.

Sean T. Smith achieved his BS in Chemical Engineering at the University of Utah in 2003. In 2008, he received his PhD in Chemical Engineering from Iowa State University. He is currently with the Department of Chemical Engineering of the University of Utah.

Charles D. Hansen received a PhD in computer science from the University of Utah in 1987. He is a professor of computer science at the University of Utah and an associate director of the SCI Institute. From 1989 to 1997, he was a Technical Staff Member in the Advanced Computing Laboratory (ACL) located at Los Alamos National Laboratory, where he formed and directed the visualization efforts in the ACL. He was a Bourse de Chateaubriand PostDoc Fellow at INRIA, Rocquencourt France, in 1987 and 1988. His research interests include large-scale scientific visualization and computer graphics.