Interactive Clipping Techniques for Texture-Based Volume Visualization and Volume Shading

Daniel Weiskopf, *Member*, *IEEE Computer Society*, Klaus Engel, and Thomas Ertl, *Member*, *IEEE Computer Society*

Abstract—We propose clipping methods that are capable of using complex geometries for volume clipping. The clipping tests exploit per-fragment operations on the graphics hardware to achieve high frame rates. In combination with texture-based volume rendering, these techniques enable the user to interactively select and explore regions of the data set. We present depth-based clipping techniques that analyze the depth structure of the boundary representation of the clip geometry to decide which parts of the volume have to be clipped. In another approach, a voxelized clip object is used to identify the clipped regions. Furthermore, the combination of volume clipping and volume shading is considered. An optical model is introduced to merge aspects of surface-based and volume-based illumination in order to achieve a consistent shading of the clipping surface. It is demonstrated how this model can be efficiently incorporated in the aforementioned clipping techniques.

Index Terms—Volume rendering, volume shading, clipping, hardware acceleration.

1 Introduction

Volume clipping plays a decisive role in understanding 3D volumetric data sets because it allows us to cut away selected parts of the volume based on the position of voxels in the data set. Very often, clipping is the only way to uncover important, otherwise hidden details of a data set. We think that this geometric approach can be regarded as complementary to the specification of transfer functions, which are based on the data values and their derivatives only. Therefore, we suggest using a combination of data-driven transfer functions and geometry-guided clipping to achieve very effective volume visualizations.

The design of useful transfer functions has recently attracted some attention [1]. One important issue is how to generate a transfer function automatically [2]. Current work is focused on other aspects and mainly deals with the question how multidimensional transfer functions [3] are better suited to extract information than a classification that is only based on the scalar values of the data set.

In most applications of interactive volume clipping, however, only a straightforward approach is adopted—with one or multiple clip planes serving as clip geometry. Typical applications are medical imaging of radiological data or volume slicing of huge seismic 3D data sets in the oil and gas industry [4], [5]. Although some clip geometries can be approximated by multiple clip planes, many useful and important geometries are not supported. Cutting a cube-shaped opening into a volume (Fig. 1) is a prominent example. Therefore, it is quite surprising that volume clipping has been rather neglected as a research topic in scientific visualization and that clipping approaches beyond standard clip planes have rarely been considered.

In this paper, we extend previous work [6] on texture-based volume clipping. Several techniques to efficiently implement complex clip geometries for volume visualization are presented and discussed. All methods are tailored to texture-based volume rendering on graphics hardware.

All hardware-based volume visualization approaches reduce the full 3D problem to a collection of 2D slices. Since these slices are displayed by the graphics hardware, we can make use of its features, such as advanced fragment operations. Our clipping approaches support both 3D textures with slices perpendicular to the viewing direction [7], [8] and stacks of 2D textures. For each fragment of a slice, fragment operations and tests on the graphics hardware decide whether the fragment should be rendered or not. Therefore, operations on the slower CPU are avoided. The proposed clipping techniques allow the user to change the position, orientation, and size of clip objects in real time. This is an important advantage because interactivity provides the user with immediate visual feedback.

In addition to the core clipping techniques for volume visualization, issues related to volume shading are specifically considered in this paper. Volume shading, in general, extends mere volume rendering by adding illumination terms to the volume rendering integral. Lighting introduces further information on the spatial structure and orientation of features in the volume data set and can thus facilitate a better understanding of the original data. The combination of volume shading and volume clipping, however, introduces issues of how illumination needs to be computed in the vicinity of the clipping object. On the one hand, the

• *The authors are with the Institute for Visualization and Interactive Systems, University of Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany. E-mail: {weiskopf, engel, ertl}@informatik.uni-stuttgart.de.*

Manuscript received 17 Nov. 2002; accepted 9 Dec. 2002.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number SI0014-1102.

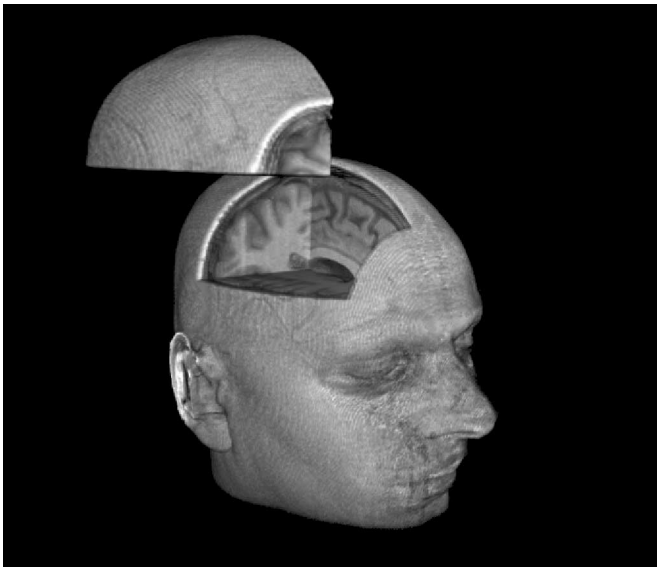


Fig. 1. Volume clipping in an MR data set.

orientation of the clipping surface should be represented. On the other hand, properties of the scalar field should still influence the appearance of the clipping surface. In this paper, we propose an optical model which takes into account consistent shading of the clipping surface.

Two basic approaches to volume clipping are discussed. In the first approach, a clip object is represented by a tessellated boundary surface. The basic idea is to store the depth structure of the clip geometry in 2D textures whose texels have a one-to-one correspondence to pixels on the viewing plane. The depth structure of the clip object can be used by fragment operations to clip away unwanted parts of the volume. This depth-based approach is particularly well-suited for producing high-quality images because clipping is performed with per-pixel accuracy. In the second approach, a clip object is voxelized and represented by an additional volume data set. Clip regions are specified by marking corresponding voxels in this volume. Both approaches can be used with or without volumetric shading.

We think that complex clip geometries could improve visualization in many fields of application. For example, current implementations of 3D LIC (line integral convolution) [9] already benefit from the use of volume clipping. Without clipping, the dense representation of a flow by streamlines would hide important features further inside the volume. Complex clip geometries can be adapted to curved boundaries of the problem domain in flow simulations. For example, the clip object could be aligned to the surface of a car body in automotive applications of CFD (computational fluid dynamics) to explore interesting features of a flow close to such boundaries.

Medical imaging is another field that would benefit from sophisticated clipping approaches. For example, segmentation information, which is often available, could serve as a basis for defining curved clip geometries. In this way, features of the data set could specifically be uncovered. In a sense, volume clipping is able to map outside semantic information into the volume data set.

Finally, volume clipping in combination with volume shading is particularly well-suited for volume illustrations. Illustrations usually contain elements with rather high opacity to facilitate the recognition of spatial structure and orientation. On the other hand, small transparency requires clipping to allow the viewer to watch interior, otherwise occluded parts. Therefore, we think that clipping is very important for shaded volume illustrations.

The paper is organized as follows: In the next section, previous work is briefly described. In Section 3, different clipping techniques based on the evaluation of depth information are presented. Section 4 contains a description of clipping by means of a voxelized clip object. It is followed by a treatment of clipping in the context of volume shading. In Section 6, results are presented and the advantages and disadvantages of the different clipping techniques are discussed. The paper ends with a short conclusion and an outlook on future work.

2 Previous Work

Clip planes are frequently used in texture-based volume rendering. For example, van Gelder and Kim [10] use clip planes to specify the boundaries of the data set in 3D texture-based volume rendering.

Westermann and Ertl [11] propose volume clipping based on stencil tests. In this approach, the geometry of the clip object has to be rendered for each slice to set the stencil buffer correctly. A clip plane that is coplanar with the current slice discards the clip geometry in front of this slice. In this way, stencil buffer entries are set at the positions where the clip plane is covered by an inside part of the clip geometry. Finally, the slice that is textured by the actual data set is rendered into the frame buffer, using the stencil test to correctly clip away fragments.

Our clipping techniques based on a volumetric description of clipping objects are related to so-called volume tagging or attributed volumes. Volume tagging is particularly useful in medical applications which provide segmentation data. For example, Hastreiter et al. [12] modify texture-based volume rendering with preclassification by applying different transfer functions to different regions of the segmented data set. In their approach, volume tags are spatially fixed and cannot be modified interactively. Tiede et al. [13] use a similar approach for the visualization of attributed volumes by ray casting. In a recent publication on texture-based volume visualization with OpenGL Volumizer [14], a clipping mechanism based on a volumetric description of clipping objects is described. The implementation requires two-pass rendering and models the visibility by means of a stencil test on each slice.

In [15], an approach similar to one of the depth-based methods of this paper is used for depth sorting semitransparent surfaces. This technique is related to *depth-peeling* by Everitt [16], facilitating the ideas of virtual pixel maps [17] and dual depth buffers [18]. A related field of research deals with issues of spatial sorting in more generality without considering specific issues of volume rendering. For a survey of this well-established topic, we refer, for example, to Foley et al. [19] or Durand [20]. These

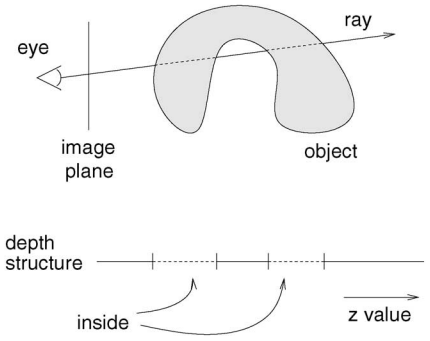


Fig. 2. Depth structure of a clip geometry for a single pixel in the frame buffer.

references also cover object-space algorithms, which are not discussed in this paper.

3 Depth-Based Clipping

In this section, we focus on algorithms for volume clipping that exploit the depth structure of the clip geometry. In all of our depth-based approaches, we assume that clip objects are defined by boundary surface representations. Graphics hardware expects surfaces to be tessellated—usually in the form of triangular meshes.

Fig. 2 illustrates the depth structure of a typical clip object. In what follows, we will no longer consider a complete 3D scenario, but reduce the problem to a 1D geometry along a single light ray that originates from the eye point and reaches into the scene. From this point of view, just the corresponding single pixel on the image plane is taken into account. Therefore, the following descriptions can be mapped to operations on the fragments that correspond to the respective position of the pixel in the frame buffer.

Let us start with a generic algorithm for clip objects of arbitrary topology and geometry. First, the depth structure of the clip geometry is built for the current pixel. This structure stores the depth values for each intersection between eye ray and object. In this way, the 1D depth values are partitioned into intervals whose bounds are given by the boundaries of the objects. In addition, the intervals are classified either as inside or as outside of the clip object. The classification is determined by the clip geometry and could be based on the orientation of the tessellated surface or on the direction of normal vectors associated with the clip geometry. For a closed, non-self-penetrating clip surface, the classification alters repeatedly from outside to inside and vice versa. The user can choose to clip away either the inside or outside of the clip object. In this way, the intervals are uniquely specified as visible or invisible. Second, the rendering of each fragment of a volume slice has to check to which class of interval the fragment belongs. Based on the visibility, the fragment is blended into the frame buffer or clipped away.

As mentioned before, intervals are either marked as visible or invisible. We introduce the term *volume probing* for a scenario where the volume is clipped away outside the clip geometry and the volume inside the clip object remains unchanged. Conversely, a *volume cutting* approach inverts the role of the visibility property. Here, only the volume outside the clip object remains visible.

The following issues have to be considered for an actual implementation of the above algorithm: What kind of data structure is appropriate to store the depth structure? How is the depth structure generated from the clip geometry? How is the visibility property of a fragment evaluated?

To be more specific, the question is how the above algorithm can be efficiently implemented on current graphics hardware. Since the depth structure is stored on a per-pixel basis, a pixel-oriented mechanism has to be considered, such as a buffer or texture on the graphics board. However, the need for high-precision depth values rules out color-based "storage media" on the GPU (graphics processing unit) such as the frame buffer or a texture with low accuracy (for example, a standard RGB texture).

3.1 Depth-Based Clipping against a Single Boundary

A straightforward approach exploits the depth buffer as a storage medium for interval boundaries. A major disadvantage of this approach is the fact that only one depth value per pixel can be stored. Therefore, just a single boundary can be implemented, which greatly reduces the fields of application. The implementation of this approach begins with rendering a simple clip geometry to the depth buffer, i.e., the depth structure is generated from the clip geometry by standard scan conversion by the graphics hardware. Then, writing to the depth buffer is disabled and depth testing is enabled. Finally, the slices of the volume data set are rendered and blended into the frame buffer. Here, the depth test implements the evaluation of the visibility property of a fragment. The user can choose to clip away the volume in front of or behind the geometry, depending on the logical operator for the depth test.

3.2 Two-Pass Rendering for Convex Volume Cutting

In this section, we show how volume cutting with a convex and closed clip object can be implemented by a two-pass rendering based on depth tests. A convex and closed object implies that the number of boundaries along the depth structure is not larger than two because, regardless of the viewing parameter, an eye ray cannot intersect the object more than twice. Concave clip geometries can benefit from this approach as well: Very often, a concave clip geometry can be approximated by two boundaries without introducing inappropriate artifacts, see the results and discussion in Section 6.

The algorithm is illustrated in Fig. 3. First, the depth values z_{back} for the second boundary are determined by rendering the backfaces of the clip geometry into the depth buffer. The depth buffer contains the initial value for the far plane of the viewing frustum in regions where the clip object is not apparent. Then, volume rendering is applied with the depth test set to "greater" and without modifying the depth buffer. In this way, the volume behind the clip object is drawn (cf., the gray portion in Fig. 3).

The second rendering pass is responsible for drawing the volume in front of the clip object and in regions not covered by the clip object (i.e., the hatched portion in Fig. 3). First,

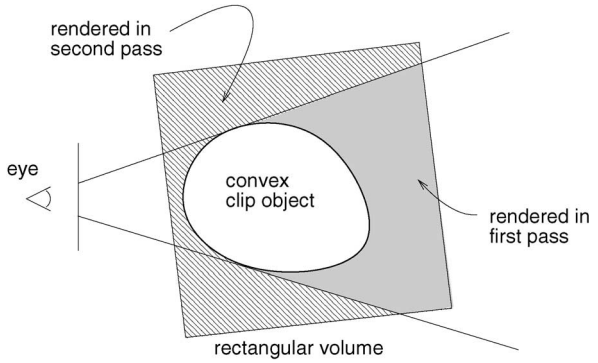


Fig. 3. Illustration of depth-based two-pass rendering for a convex clip geometry.

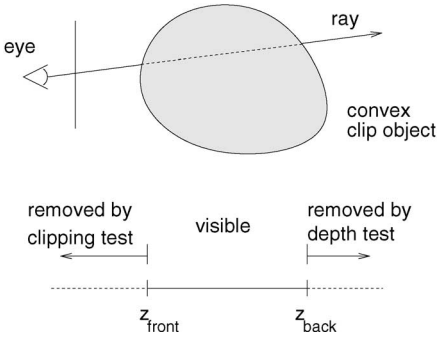


Fig. 4. Illustration of depth-based volume probing for a convex clip geometry.

the depth buffer is filled with the depth values z_{front} for the first boundary by rendering the frontfaces of the clip geometry. Afterward, volume rendering is accomplished with the depth test set to "less" and without modifying the depth buffer.

3.3 Single-Pass Rendering for Convex Volume Probing Based on Depth Clipping

Let us now consider a volume probing approach for a convex clip object. We demonstrate how a maximum number of two boundaries along the depth structure can be implemented as single-pass rendering by exploiting depth tests, depth clipping, and depth computations in the fragment operations unit.

The basic algorithm for depth-based volume probing is as follows: First, the depth values z_{front} for the first boundary are determined by rendering the frontfaces of the clip geometry into the depth buffer. Second, the contents of the depth buffer are stored in a texture and the fragment shader is configured to shift the depth values of all fragments in the following rendering passes by $-z_{\text{front}}$. Third, the depth buffer is cleared and the backside of the clip geometry is rendered into the depth buffer (with depth shift enabled) to build the second boundary. Finally, slices through the volume data set are rendered and blended into the frame buffer, without modifying the depth buffer. However, depth shift and depth testing have to be enabled.

Let us examine in more detail how the depth of a fragment determines its visibility. The visibility depends on both depth clipping and depth testing. The decision whether a fragment passes depth clipping and depth testing can be expressed by a Boolean function $d_f(z_f)$, where z_f is the depth of the fragment. Depth values are assumed to be described with respect to normalized window coordinates, i.e., valid depth values lie in $[0,1]$. In general, the value of $d_f(z_f)$ can formally be written as a logical expression,

$$d_f(z_f) = d_{\text{clip}}(z_f) \wedge (z_f \ \mathbf{op}\ z_b), \tag{1}$$

with

$$d_{\text{clip}}(z_f) = (z_f \geq 0) \wedge (z_f \leq 1).$$

In this notation, $\wedge$ symbolizes a logical "and." The function $d_{\text{clip}}(z_f)$ describes depth clipping against the bounds 0 and 1 of the view frustum. The binary logical operator $\mathbf{op}$ is used for depth testing against z_b, the current entry in the depth buffer.

Fig. 4 illustrates how our clipping algorithm operates on the depth structure. A shift by $-z_{\text{front}}$ is applied to all depth values. By rendering the backfaces of the clip object with the same shift, entries in the depth buffer are initialized to $z_{\text{back}} - z_{\text{front}}$, where z_{back} is the unmodified depth of the backface's fragment. During the final rendering of the volume slices, the logical operator for the depth test is set to "$\leq$". Therefore, (1) can be written as

$$\begin{aligned} d_f(z_f) &= \big((z_f - z_{\text{front}} \geq 0) \wedge (z_f - z_{\text{front}} \leq 1)\big) \\ &\quad \wedge (z_f - z_{\text{front}} \leq z_{\text{back}} - z_{\text{front}}) \\ &= d_{\text{probe}}(z_f) \wedge (z_f \leq 1 + z_{\text{front}}), \end{aligned}$$

with

$$d_{\text{probe}}(z_f) = (z_f \geq z_{\text{front}}) \wedge (z_f \leq z_{\text{back}}). \tag{2}$$

The term $d_{\text{probe}}(z_f)$ exactly represents the required logical operation for displaying the volume only in the interval $[z_{\text{front}}, z_{\text{back}}]$. The last term, $(z_f \leq 1 + z_{\text{front}})$, implies just a clipping against a modified far clipping plane.

A shift of depth values by $-z_{\text{front}}$ can be efficiently accomplished on a per-fragment basis by modern graphics hardware. We refer to previous work [6] for a detailed description of the implementation on an NVidia GeForce 3/4. A high-resolution texture is used to store the respective depth values z_{front} and a fragment program is employed to replace the z value of a fragment by $z - z_{\text{front}}$. The details of the fragment program are presented in [6]. Furthermore, that paper describes how texture coordinates for this multitexturing are issued on a per-vertex basis within the transform and lighting part of the rendering pipeline by means of a vertex program.

3.4 Single-Pass Rendering for Convex Volume Cutting Based on Depth Clipping

In this section, we adapt the previous clipping mechanism to invert the role of the visibility property. Volume cutting replaces the above volume probing approach, i.e., only the volume outside the clip object remains visible. Once again, we restrict ourselves to a maximum number of two boundaries along the depth structure.

By inverting (2) for volume probing, one obtains

$$d_{\text{cutting}}(z_f) = (z_f \leq z_{\text{front}}) \vee (z_f \geq z_{\text{back}}), \tag{3}$$

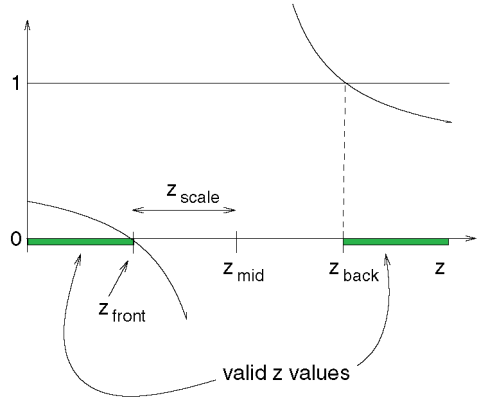


Fig. 5. Schematic illustration of the depth structure for volume cutting via $1/z$ mapping.

for volume cutting, with $\vee$ representing the logical "or" operator. The cutting function $d_{\text{cutting}}(z_f)$ is almost identical to the negation of the probing function, $\neg\, d_{\text{probe}}(z_f)$—except for the fact that z_{front} and z_{back} satisfy the Boolean functions for both volume probing and volume cutting. Unfortunately, logical operations for depth clipping and depth testing have to be of the form of (1), which is not compatible with (3).

Our solution to this problem is based on the following considerations: Let us first have a look at a simplified scenario with the expression $(x \geq -1) \wedge (x \leq 1)$ for values x. This expression can be rewritten as $(f(x) \geq -1) \wedge (f(x) \leq 1)$ by introducing an identity function $f(x) = x$.

The valid values x are determined by the intersection of $f(x)$ with the horizontal lines at ± 1. Now, consider a similar situation in which the function $f(x)$ is replaced by another function $g(x) = 1/x$. In this case, valid values of the logical expression $(g(x) \geq -1) \wedge (g(x) \leq 1)$ satisfy $(x \leq -1) \vee (x \geq 1)$. Therefore, a mapping between the form of (3) and the required form of (1) can be achieved. As a matter of fact, there are other possible choices of mapping functions to achieve a similar inversion of z clipping. The advantage of a hyperbolic mapping is its availability in the fragment operations of modern graphics boards.

The adaption of this idea to the graphics hardware needs to take into account that window coordinates lie in $[0, 1]$ and not in $[-1, 1]$. Moreover, the vertical asymptote of the hyperbola must not meet the origin, $z = 0$, but has to be located at the midpoint of the clip geometry. The midpoint is obtained by $z_{\text{mid}} = (z_{\text{front}} + z_{\text{back}})/2$. In addition, the actual distance between the front and the back boundaries of the clip geometry is taken into account by the scaling factor $z_{\text{scale}} = (z_{\text{back}} - z_{\text{front}})/2$. Fig. 5 illustrates how the quantities of the clip object can be used for volume cutting via $1/z$ mapping. The correct function has to be

$$g(z) = \frac{1}{2}\frac{z_{\text{scale}}}{z - z_{\text{mid}}} + \frac{1}{2}, \tag{4}$$

in order to produce the graph of Fig. 5.

The algorithm for depth-based volume cutting can now be formulated as follows: First, the depth values of the frontfaces of the clip geometry are rendered into the depth buffer. Then, these values z_{front} are transferred to main memory. Analogously, the depth values of the backfaces, z_{back}, are determined and copied to main memory. Based on z_{front} and z_{back}, the CPU computes z_{mid} and z_{scale} and stores them as first and second components of a high-resolution texture.

A fragment program is enabled to implement the mapping according to (4) by employing the above-mentioned texture to modify the depth value of each fragment. Finally, slices through the volume data set are rendered and blended into the frame buffer. In contrast to the volume probing described in the previous section, the depth buffer does not contain any information on the clip geometry and depth testing is not required. Volume cutting is only based on the data from the high-resolution texture and on view frustum clipping.

Once again, we refer to previous work [6] for details of the implementation on a GeForce 3/4, including both the fragment program and the vertex program that issues texture coordinates on a per-vertex basis.

3.5 Convex Volume Clipping Based on Shadow Test

The two previous clipping techniques are based on high-resolution textures, transformation of depth values, and view frustum clipping. An alternative, similarly structured approach exploits shadow mapping and the shadow test to achieve comparable results. A similar approach is taken by Everitt [16] for spatial sorting in surface-based rendering.

Compared to the algorithm from the previous two sections, shadow-based clipping introduces the following modifications: First, the contents of the z buffer are transferred to a depth texture that can store values for a subsequent shadow test with high accuracy. Second, the fragment operations now perform a comparison between the depth of a fragment and the stored depth structure (from the depth texture) by means of the shadow test. In a typical application of shadow mapping, the positions of the virtual light and camera are different. In this paper, the virtual light for shadow testing is identical to the camera. The test result is stored in the α component of the fragment. Finally, the fragment may be removed by the α test, depending on the result of the precedent depth comparison.

An advantage of the shadow-mapping approach is the fact that only one texture stage (in the fragment operations unit) is needed for each depth texture, whereas the approach based on view frustum clipping needs three texture stages to implement the shift of z values on a GeForce 3/4. On the other hand, the techniques from Sections 3.3 and 3.4 have the advantage that the visibility property is checked by a clipping test against the z values of the viewing frustum. Since the fragment's color attribute is not needed to code the result of the clipping test, clipped fragments can be distinguished from fragments that are transparent because of their α value assigned by the transfer function.

3.6 Multipass Rendering for Concave Clip Objects

So far, only convex clip objects can be correctly handled. In order to allow for concave objects, the management of the depth structure has to be extended to consider a larger number of possible intersections between the clipping geometry and an eye ray.

We propose the following multipass rendering technique for concave clipping. First, the depth structure of the clip

object is analyzed and neighboring pairs of z values in this depth structure are combined. These pairs are organized in a way that ensures that each pair represents one visible segment of the viewing ray, i.e., the volume remains visible within the interval bounded by such a pair of z values. Note that, in the case of volume cutting, this organization generates single, unpaired depth entries at the front and back parts of the depth structure. The *depth-peeling* algorithm [16] can be used to determine the depth structure from a triangle mesh. Each pair of z values is stored in separate textures, i.e., n pairs of textures are required if n describes the maximum number of depth pairs in the depth structure. Note that this process is performed only once per frame.

Second, the actual multipass rendering of the volume is accomplished. For each pair of z values from the depth structure, the corresponding region of the volume is rendered. Since each depth pair represents a local volume probing against two boundaries, rendering is based on the techniques from Sections 3.3 or 3.5. To put it another way, the complete volume is reconstructed in a layer-by-layer fashion by processing the depth structure pair-by-pair. The depth structure is processed in the same direction as the direction of slice compositing. For example, a back-to-front compositing implies that the depth pairs are also processed back-to-front. The case of volume cutting—with single depth values at the front and back of the depth structure—can be handled by clipping only at a single boundary, as described in Section 3.1.

4 Clipping Based on Volumetric Textures

In contrast to the depth-based clipping algorithms from the previous sections, volumetric clipping approaches model the visibility information by means of a second volume texture whose voxels provide the information for clipping voxels of the "real" volume. Here, the clip geometry is voxelized and stored as a binary volume. This clipping texture is either a 3D texture or a stack of 2D textures. It is initialized with one for all voxels inside the clip geometry and with zero for all voxels outside. Thus, arbitrary voxelized clip objects are possible.

Conceptually, the clipping texture and the volume data set are combined by multiplying their texture entries. In this way, all the voxels to be clipped are multiplied by zero. During actual rendering, a texture slice of the data set and a slice of the clip texture are simultaneously mapped onto the same slice polygon and combined using a per-component multiplication. Fragments lying outside the clip object have an α value of zero and can be discarded by a subsequent α test.

The clip object can be scaled and moved by adapting the texture coordinates of the clipping texture. Additionally, when using 3D textures, a rotation of the clip object is possible. To be more precise, any affine transformation of the 3D texture-based clip geometry can be represented by a corresponding transformation of texture coordinates. Switching between volume probing and volume cutting is easily achieved by applying a per-fragment invert mapping to the clipping texture. In this way, the original texture value is replaced by $(1 - \text{value})$. Since all these operations only require the change of texture coordinates or a reconfiguration of per-fragment operations, all of the above transformations of the clip geometry can be performed very efficiently. Only a complete change of the shape of the clip geometry requires a revoxelization and a reload of the clipping texture.

The above technique is based on a binary representation and therefore requires a nearest neighbor sampling of the clipping texture. If a bilinear (for a stack of 2D textures) or a trilinear (for a 3D texture) interpolation was applied, intermediate values between zero and one would result from a texture fetch in the clipping texture. Therefore, a clearly defined surface of the clip geometry would be replaced by a gradual and rather diffuse transition between visible and clipped parts of the volume, which is inappropriate for most applications. However, a missing interpolation within the clipping texture introduces quite visible, jaggy artifacts similar to those in aliased line drawings.

To overcome this problem we use the following modification of the original approach: The voxelized clipping texture is no longer used as a binary data set, but interpreted as a distance volume: Each voxel stores the (signed) Euclidean distance to the closest point on the clip object. Since all common 3D texture formats are limited to values from the interval $[0, 1]$, the distance is mapped to this range. In this way, the isosurface for the isovalue 0.5 represents the surface of the clip object. During rendering, a trilinear interpolation in the 3D clipping texture is applied (a stack of 2D textures is not considered here because of the missing interpolation between slices). Based on the comparison of the value from the clipping texture with 0.5, either the $\mathrm{RGB}\alpha$ value from the data set (for a visible fragment) or zero (for an invisible fragment) is used as the final fragment color. In this way, a clearly defined surface without jaggy artifacts is introduced at the isovalue 0.5.

Due to the availability of multiple texturing units, usually no or only little performance penalty is introduced for the actual rasterization. However, performance might be affected by additional memory access for the clipping texture. An important advantage of this method is the fact that any kind of clip geometry can be specified—without any limitation with respect to topology or geometry. The main disadvantages of the approach are the additional texture memory requirements for the clipping volume, the need for a voxelization of the clip geometry, and possible artifacts due to the limited resolution of the clipping texture. Moreover, the clipping surface can only be represented as an isosurface in a piecewise trilinearly interpolated scalar field. Therefore, it is difficult to reproduce clip objects with sharp edges and creases.

5 Clipping for Volume Shading

Volume shading extends mere volume rendering by adding illumination terms. It is based on gradients of the scalar field and works very well if clipping is not applied. The combination of volume shading and volume clipping, however, introduces issues of how illumination has to be computed in the vicinity of the clip object. Here, illumination should not only be based on the properties of the scalar field, but should rather represent the orientation of the clipping surface itself. On the other hand, the scalar field

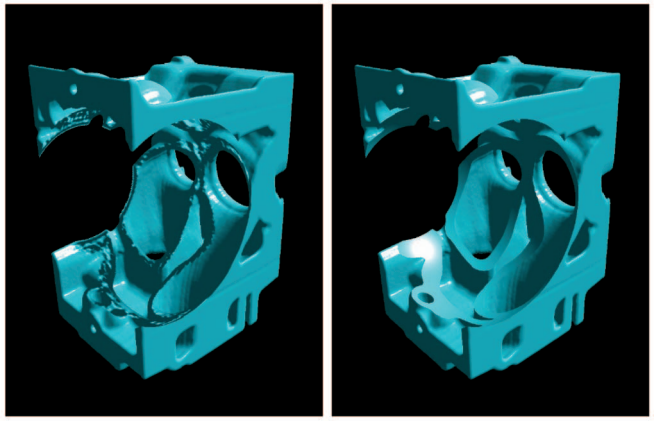


Fig. 6. Combining volume clipping and volume shading. Depth-based volume cutting with a spherical clip object is applied to a technical engine data set. No specific lighting is applied at the cutting surface in (a); (b) reveals lighting (e.g., a highlight) on the cutting surface by combining surface-based and volumetric shading.

should still influence the appearance of the clipping surface, for example, with respect to color and transparency.

Fig. 6b demonstrates how lighting the volume and the clipping surface improves the perception of the spatial structure of the clip object. For comparison, Fig. 6a shows the same data set without specific lighting on the clipping surface; it is difficult to recognize the spherical cutting surface because of the inappropriate lighting conditions.

In the remaining parts of this section, we discuss ways to combine illuminated clipping surfaces with shaded volumes in a meaningful way. Both depth-based and volumetric clipping techniques from the previous sections are extended to take into account a consistent shading of the clipping surface.

5.1 Optical Models for Volume Shading

To clarify the issues related to clipping in illuminated volumes, we begin with a brief review of the mathematical foundation of the optical models for volume rendering and volume shading. A detailed presentation of the underlying physical and mathematical models can be found in the review articles by Hege et al. [21] or Max [22]. In this paper, we follow the terminology used by Max.

By ignoring scattering, the generic equation of transfer for light can be reduced to the simpler emission-absorption model or density-emitter model [23]. The respective volume rendering equation can be formulated in the form of the differential equation

$$\frac{\mathrm{d}I(s)}{\mathrm{d}s} = g(s) - \tau(s)I(s). \tag{5}$$

The amount of radiative energy is described by radiance $I(s)$ and its derivative with respect to the length parameter s is taken along the direction of the light flow. The source term $g(s)$ describes the emission of light from the participating medium (and later reflection as well). The extinction coefficient $\tau(s)$ defines the rate that light is attenuated by the medium. The optical properties are determined by a transfer function that assigns a value for the extinction coefficient and the source term to each scalar value. Radiance and optical properties are written as scalars, which is appropriate for luminance-based gray-scale images. A wavelength-dependent behavior or multiple wavelength bands (for example, red, green, and blue) can be taken into account by computing theses quantities for each wavelength separately.

The volume rendering equation in the differential form (5) can be solved for radiance by integrating along the direction of light flow from the starting point $s = s_0$ to the end point $s = D$,

$$I(D) = I_0\, e^{-\int_{s_0}^{D} \tau(t)\,\mathrm{d}t} + \int_{s_0}^{D} g(s)\, e^{-\int_{s}^{D} \tau(t)\,\mathrm{d}t}\,\mathrm{d}s.$$

The term I_0 represents the light entering the volume from the background at the position $s = s_0$; $I(D)$ is the radiance leaving the volume at $s = D$ and finally reaching the camera. With the definition of transparency,

$$T(s) = e^{-\int_{s}^{D} \tau(t)\,\mathrm{d}t},$$

we obtain the volume rendering integral

$$I(D) = I_0\, T(s_0) + \int_{s_0}^{D} g(s) T(s)\,\mathrm{d}s. \tag{6}$$

Scattering of light from outside a voxel of the participating medium has to be included to introduce greater realism into this optical model. In a simple volume shading model, the external illumination is assumed to unimpededly reach a voxel from an outside light source, neglecting any volume absorption on the way to the voxel. Volume scattering at a voxel can be computed similarly to local illumination models known from surface rendering, such as the Phong or the Blinn-Phong models. In volume shading, the gradient of the scalar field serves as the normal vector for these local illumination models because the gradient is identical to the normal vector on an isosurface through the respective point in space. In this way, volume shading produces an effect similar to an illuminated isosurface. Local illumination is included in the volume rendering integral (6) by extending the source term to

$$g(\vec{x}, \vec{\omega}) = E(\vec{x}) + S(\vec{x}, \vec{\omega}).$$

The nondirectional emissivity $E(\vec{x})$ is identical to the source term in the pure emission-absorption model. The additional scattering term $S(\vec{x}, \vec{\omega})$ depends on the position $\vec{x}$, the direction of the reflected light, $\vec{\omega}$, and the properties of the light source.

In slice-based volume rendering, the integral in (6) is approximated by a Riemann sum over n equidistant segments of length $\Delta x = (D - s_0)/n$. The approximation yields

$$I(D) \approx I_0 \prod_{i=1}^{n} t_i + \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j, \tag{7}$$

where

$$t_i = e^{-\tau(i\Delta x + s_0)\Delta x} \tag{8}$$

is the transparency of the ith segment and

$$g_i = g(i\Delta x + s_0)\Delta x \tag{9}$$

is the source term for the ith segment. (Note that the factor Δx for g_i is erroneously missing in [22]). It is important for the subsequent discussion that both the discretized transparency t_i and source term g_i depend on the segment length Δx. In texture-based volume rendering, fragments are assigned opacities $\alpha_i = 1 - t_i$ and gray values (or colors) g_i. The opacity and source term have to be adapted if the slice sampling distance Δx is changed.

5.2 An Optical Model for Clipping in Illuminated Volumes

For a consistent combination of volume clipping and volume shading, the following requirements should be met: First, the illumination of the clipping surface should allow the viewer to recognize the orientation and shape of this surface. Second, the optical properties of the clipping surface should be compatible with the optical properties of the scalar data field at the same position to avoid discontinuous changes in the visual appearance. Third, volume shading in regions distant from clipping surfaces should not be affected by the clipping algorithm. Fourth, the generated images should be independent of the chosen sampling rate Δx—up to improvements of accuracy by a refinement of the Riemann sum. In particular, from a mathematical point of view, the image of a clipped volume has to converge to a reasonable limit value for $\Delta x \to 0$.

Let us consider a first approach to combining clipping and volume shading. The idea is to apply unmodified clipping techniques (e.g., from the previous sections) to the shaded volume. Illumination is based on the gradient of the scalar field and on the optical properties assigned by the transfer function. Afterward, the shaded clipping surface is layered on top of the volume, applying illumination based on the normal vector of the surface. The optical properties and, thus, the material properties of the clipping surface are defined by the same transfer function and scalar values as for volume shading. Furthermore, the same local illumination model is applied as for volume shading. The surface of the clip object can be pictured as a "skin" surrounding the actual volume, as illustrated in Fig. 7a.

This approach meets the first three requirements for a successful combination of clipping and volume shading. First, the lighting computation at the "skin" is based on the normal vector on the clipping surface. Therefore, the orientation of the cut can be recognized. Second, the appearance of the "skin" and the neighboring volume is compatible because identical material properties are applied. Third, volume shading away from the clipping surface is not affected at all.

Unfortunately, this first approach is not invariant if the sampling rate Δx changes. In the discretized form of (9), the source term g_i converges to zero for $\Delta x \to 0$, i.e., the color contribution of a 2D surface (of infinitely small thickness) converges to zero as well. Similarly, the transparency t_i from (8) converges to one for $\Delta x \to 0$ unless τ is infinite. (The special case of completely opaque, surface-like regions in the volume—where transparency is zero and τ and g are delta distributions—is automatically handled by the "skin" and the following approach alike.) The underlying problem is that a continuous volumetric description of a 3D volume is combined with an infinitely thin 2D hypersurface. In the original, continuous representation by the volume rendering integral (6), the contribution of terms g and τ is zero if their support is of zero length.

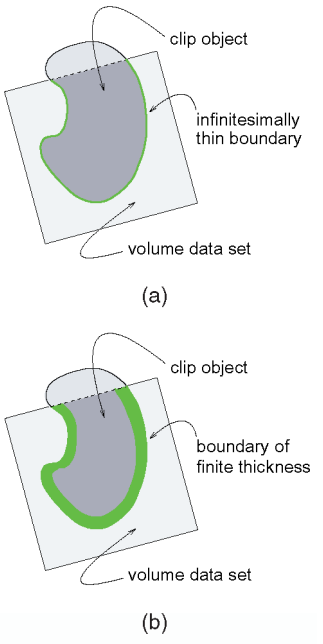


Fig. 7. Combining volume shading for a volumetric data set with surface-based shading on the boundary of the clip geometry. Image (a) shows an infinitesimally thin "skin" around the clip object; image (b) illustrates the extension of the surrounding layer to finite thickness.

This problem can be overcome by modifying the first approach in a way that the "skin" is broadened to a finite thickness. The volume is covered by a thick layer whose illumination is governed by the normal vectors on the clipping surface. The influence of the orientation of the surface reaches into the volume for some distance, which could be pictured as "impregnating" the shaded volume with lighting information from the surrounding clipping surface. Fig. 7a and Fig. 7b compare both the original "skin" approach and the modified "impregnation" approach. The "impregnation" reaches only into the remaining volume; it may not change the geometric clipping (i.e., the visibility of voxels), but only the illumination in the layer region.

The transition between clipping surface and interior volume is introduced into the volume shading model by extending the source term to

$$g(\vec{x}, \vec{\omega}) = E(\vec{x}) + w(\vec{x}) S_{\mathrm{srf}}(\vec{x}, \vec{\omega}) + (1 - w(\vec{x})) S_{\mathrm{vol}}(\vec{x}, \vec{\omega}),$$

with the weight function $w(\vec{x})$, the surface-based illumination term $S_{\mathrm{srf}}(\vec{x}, \vec{\omega})$, and the volumetric scattering term $S_{\mathrm{vol}}(\vec{x}, \vec{\omega})$. From construction, the thickness of the layer is limited and, therefore, the support of the weight function $w(\vec{x})$ has to be compact (i.e., closed and bounded). Moreover, the function values $w(\vec{x})$ have to be from $[0, 1]$ to ensure a convex

combination of the two illumination terms. We propose restricting the model to a uniform thickness of the layer. In this way, one important parameter—the thickness of the "impregnation"—is introduced. It is assumed that the thickness parameter is specified by the user. Another choice concerns the functional behavior of the weight itself. A linear behavior with respect to the distance from the clipping surface (i.e., a linear ramp), a step function (i.e., $w = 1$ if inside the boundary layer and $w = 0$ elsewhere), or a more sophisticated function could be used. A continuous or even smooth weight function has the advantage that the illumination shows a continuous or smooth transition between the "impregnation" layer and the remaining parts of the volume.

One further issue is the evaluation of the surface-related illumination part, $S_{\mathrm{srf}}(\vec{x}, \vec{\omega})$, at positions away from the clipping surface. In particular, what is the normal vector in those regions? Our definition of a surface-based normal vector in the "impregnation" layer makes use of the concept of parallel transport. For each point in the layer, the closest point on the clipping surface is determined. The normal vector is evaluated at that point on the surface and then transported to the respective location within the layer. The material parameters can be directly computed from the optical properties of the scalar data set at any point in the layer.

The "impregnation" approach meets all four requirements for an appropriate combination of clipping and volume shading. By introducing a finite boundary region, the contribution of the illuminated clipping surface to the volume rendering integral (6) is finite. The discretized version of the integral (7) converges to this contribution for successive refinements of the sampling rate. Furthermore, volume shading in regions distant from clipping surfaces is not affected by clipping because of the compact support for the weight function. The first two requirements are met due to the same arguments as for the "skin" approach.

5.3 Combining Clipping and Volume Shading in an Object-Space Approach

The above mathematical discussion of the "impregnation" approach leads to the following algorithm for combining clipping and volume shading. All considerations are based on object space. We assume that the clipping object is defined as the isosurface of a signed distance volume for the isovalue zero (analogously to volumetric clipping in Section 4). We furthermore require that the distance field is based on the Euclidean norm in 3D Euclidean space and that the clipping surface is non-self-penetrating and smooth.

With these assumptions, a direct mapping of the above mathematical formulation can be accomplished. The generic weight function $w(\vec{x})$ can be expressed in terms of the distance volume $v_{\mathrm{distance}}(\vec{x})$ and an alternative weight function $\tilde{w}(d)$ according to

$$w(\vec{x}) = \tilde{w}(v_{\mathrm{distance}}(\vec{x})). \tag{10}$$

The weight function $\tilde{w}(d)$ has support $[0, d_{\max}]$, where $d_{\max}$ describes the thickness of the "impregnation" layer. Function values $\tilde{w}(d)$ are from $[0, 1]$. By the above construction, $w(\vec{x})$ models a uniform thickness of the layer.

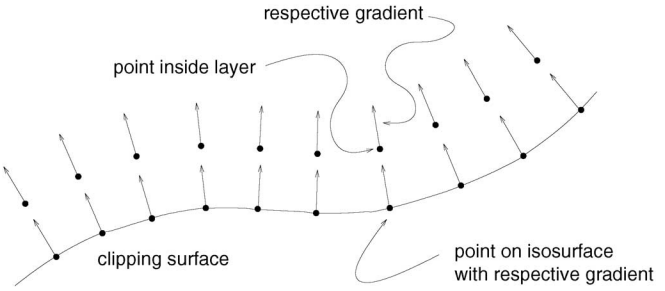


Fig. 8. Gradients in distance field.

Surface-based illumination also makes use of the distance volume. Not only is the signed distance stored, but the gradient of the distance field as well. The gradient is identical to the normal vector of a respective isosurface, i.e., the gradient represents the normal vector on the clipping surface. Furthermore, even in a small neighborhood (ϵ neighborhood) around the smooth clipping surface, the gradient reflects the normal vector that could be obtained by parallel transport from the isosurface. This specific property of a distance volume is caused by the fact that the gradient represents the direction to or from the closest point on the isosurface. This behavior is illustrated in Fig. 8. The maximum size of the neighborhood depends on the curvature of the clipping surface; the neighborhood shrinks with increasing curvature. If the "impregnation" layer is contained in these local neighborhoods, the surface-shading term $S_{\mathrm{srf}}(\vec{x}, \vec{\omega})$ can be computed inside the layer by using the gradient of the distance field and the optical properties of the scalar field in that region.

Therefore, a quite generic formulation is available to combine clipping and volume shading. This object-space approach can be incorporated in a wide class of volume rendering algorithms, such as ray casting or splatting. In this paper, we rather focus on texture-based rendering and thus demonstrate how the previously described depth-based and volumetric clipping techniques can be extended to allow for consistent volume and surface shading.

5.4 Volumetric Clipping and Volume Shading

Volumetric clipping from Section 4 already models the clipping object by means of an isosurface through a distance field. Accordingly, the above formulation can be directly mapped to volumetric clipping. The following minor changes need to be included in the original clipping algorithm:

First, the texture representations of the volume data set and the distance field are extended to include both scalar values and gradients. For example, an RGBα texture format can be employed to store the scalar value in α and the gradient in RGB. The gradient fields serve as a basis to evaluate the local illumination model.

Second, the fragment operation unit is reconfigured to evaluate local illumination for the original data set and the distance field. The material properties for both volumes are based on the entries in the data set and the transfer function. Since lighting is not precomputed, the user may change lighting parameters, such as material properties or color and position of the light source, without redefining the

RGBα textures for the volume data set and the distance field.

Third, the weight function $\tilde{w}(d)$ is evaluated per fragment to implement (10). The distance d is obtained from the distance field. The thickness of the "impregnation" layer can be adapted by scaling d. A linear ramp behavior for $\tilde{w}(d)$ can be realized by a linear interpolation with subsequent clamping at 0 and 1; a step function is implemented by a comparison operation. More sophisticated weighting functions need respective numerical operations. The final function value $\tilde{w}(d)$ serves as weight for a linear interpolation between the lighting computation for the original data set and the distance field (from Step 2). An even simpler comparison operation is sufficient provided $\tilde{w}(d)$ is a step function. It is important that all these computations can be accomplished by fragment operations on modern GPUs. Our implementation, for example, is based on the register combiner unit of the GeForce 3/4.

All other parts of the original clipping algorithm are not affected. In particular, actual clipping is still based on a comparison between the entry in the distance field and the isovalue of the clip geometry, followed by an α test. Moreover, the number of texture fetches is not increased —two textures are still sufficient for the data set and the distance field.

5.5 Depth-Based Clipping and Volume Shading

The ideas from the above object-space approach need some modifications before they can be applied to depth-based clipping. The main problem is that a measure for the distance from the clipping surface is not available. We propose an approximation of our basic optical clipping model which does not require any distance information.

The reason for introducing an "impregnation" layer of finite thickness was the fact that transparency t_i and source term g_i in (8) and (9) depend on the slice distance. In particular, an infinitely thin layer will have no effect on the final image when the sampling rate is very high. This observation leads to the following modified approach:

The clipping boundary is not modeled as a thick layer but as a 2D hypersurface. Shading inside the volume is based on transparency values and source terms that are adapted to the slicing distance Δx. Surface-based shading on the clip geometry, however, yields modified contributions to transparency

$$t_{\mathrm{srf}} = e^{-\tau(s_{\mathrm{srf}})\Delta x_{\mathrm{srf}}},$$

and source term

$$g_{\mathrm{srf}} = g(s_{\mathrm{srf}})\Delta x_{\mathrm{srf}},$$

where s_{srf} describes the location of the clipping surface and Δx_{srf} represents the thickness of the original "impregnation" layer, which is not dependent on the sampling rate. Once again, the optical properties are defined by the volume data set and the transfer function. To put it another way, geometrically the clip boundary is treated as a 2D surface, whereas the contribution to the rendering integral comes from a thick virtual layer.

This approach results in the following approximations of the original distance-based model: First, the terms $g(s_{\mathrm{srf}})$ and $\tau(s_{\mathrm{srf}})$ are evaluated only at the clipping surface and, thus, the surface-based transparency and source term are constant inside the "impregnation" layer. Second, the weight function $\tilde{w}(d)$ is implicitly set to a step function. Third, the volume rendering integral has double contributions from the "impregnation" layer—both from the above terms, t_{srf} and g_{srf}, and erroneously from the slice-based sampling of the volume data set. Fourth, the layer is rendered as a surface of infinitesimal thickness.

The effects of these approximations depend on the thickness of the layer, the rate of change of scalar values around the clipping surface, and the opacity. In typical applications, the approximations are not noticeable to the user because the layer thickness is rather small and, more importantly, the opacity is quite high in regions that have strong illumination. A clipping surface of high opacity essentially covers parts of the volume that are further away.

On the other hand, the following benefits are introduced by surface-based shading: First, surface shading has higher visual quality than volume shading because of a better representation of normal vectors. Second, the normal vector is always well-defined because no parallel transport is employed. The normal vector field on the clipping surface does not even need to be continuous and, thus, clipping geometries with sharp edges and creases are possible.

For the actual implementation, the following modifications are introduced into the original depth-based clipping techniques from Section 3. Essentially, pure slice-based volume rendering is extended to hybrid volume and surface rendering. The volume and the surface parts are interleaved according to the depth structure of the clip geometry and so is the order of surface and volume rendering. Accordingly, multipass rendering approaches have to be used; single-pass rendering for convex volume cutting (Sections 3.4 and 3.5) is not suitable because the surface part is contained within the volume part and cannot be separated. All other techniques from Section 3, however, are prepared to alternatingly render regions of the clipped volume and the surrounding clipping surfaces.

Volume rendering and volume clipping are directly adopted from Section 3, with volume shading being added. Surface rendering is based on the tessellated representation of the clip object. Material properties are obtained from the 3D scalar data set and the transfer function, i.e., texture coordinates are assigned to the vertices of the surface in order to access the 3D texture of the data set. The normal vector is computed from the surface model. Fragment operations compute the modified source and transparency terms to obtain the contribution of the clipping surface to the image. Finally, this contribution is combined with the volume parts by blending into the frame buffer.

6 Results and Discussion

Fig. 9 illustrates the different volume clipping methods of this paper. The underlying test data set is of size 128^3 and shows a piecewise linear change of scalar values along the vertical axis, which are mapped to yellow and blue colors. The boundary of the cubic volume has a different scalar value that is mapped to a brownish-range color. Lighting is

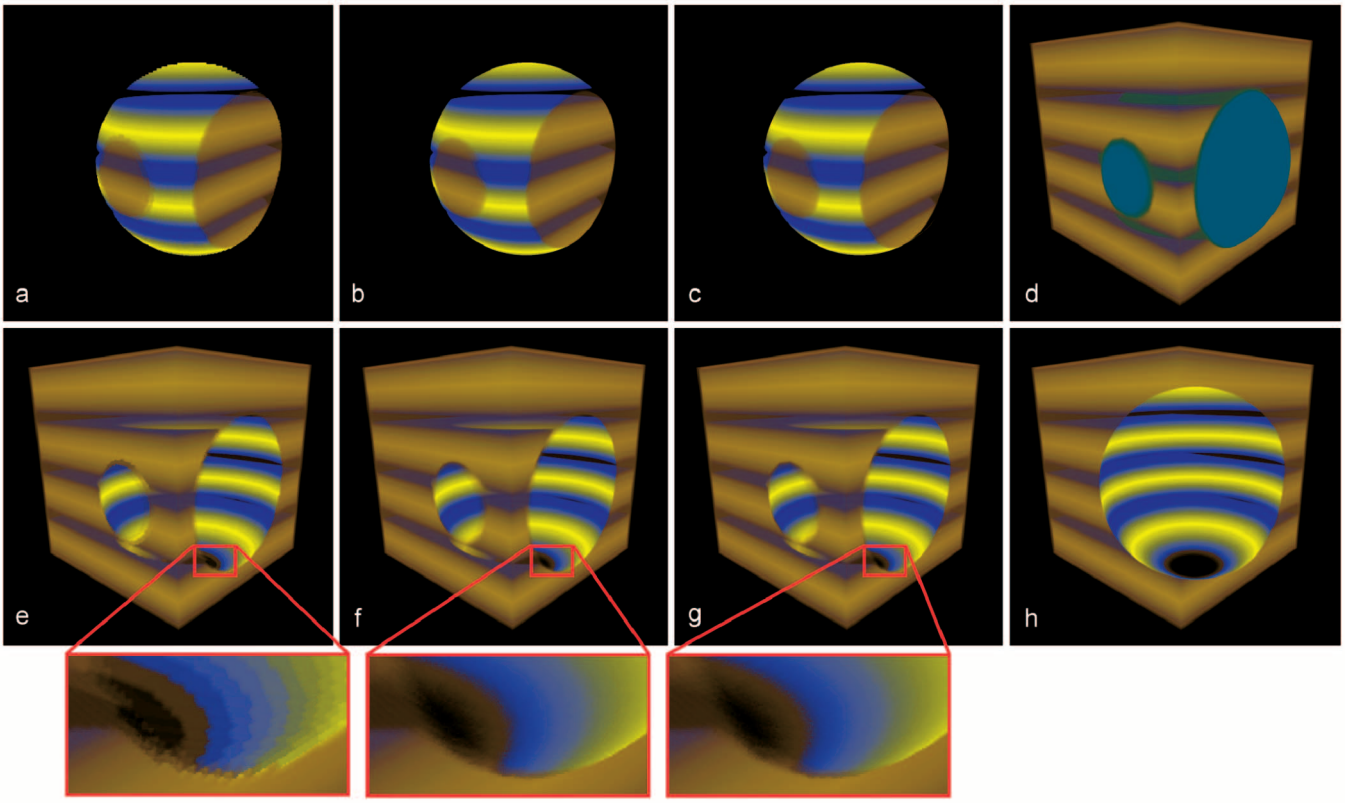


Fig. 9. Comparison of volume clipping techniques.

disabled and texture-based rendering with preclassification based on paletted textures is used.

The first three images in the top row of Fig. 9 are produced by volume probing, the images in the second row are produced by volume cutting. Fig. 9a, Fig. 9b, Fig. 9e, and Fig. 9f show clipping based on a volumetric texture—as described in Section 4. The clipping texture has the same resolution as the data set. In Fig. 9a and Fig. 9e, nearest neighbor sampling is applied; in Fig. 9b and Fig. 9f, a trilinear interpolation within the distance field is used. Artifacts are clearly visible for nearest neighbor sampling but not for trilinear interpolation. Fig. 9c and Fig. 9g demonstrate depth-based clipping as presented in Sections 3.3 and 3.4. Depth-based clipping produces visualizations of high quality because clipping is performed with per-pixel accuracy. Depth-based clipping from Section 3.5 (via shadow tests) achieves the same visual quality as in Fig. 9c and Fig. 9g because a similar basic clipping mechanism is used. For Fig. 9h, only a single clip surface is implemented according to the depth-based approach of Section 3.1. This technique erroneously removes parts of the volume data set around the closest vertical edge. In an interactive application, the introduced errors are quite disturbing; the spatial structure is very hard to grasp because clipping properties become view-dependent in this simple approach. Finally, Fig. 9d shows the clip geometry in the form of an opaque sphere.

In Fig. 10, a cube-shaped voxelized clip object is used for volume cutting within the data set of an engine block. Trilinear interpolation is applied in Fig. 10a, nearest neighbor sampling in Fig. 10b. Due to trilinear interpolation, the corners and edges of the cubic clip object are smoothed out unless the faces of the cube are identical with voxel boundaries in the clip volume. Therefore, the original clip geometry cannot be reproduced in Fig. 10a. An example of a significant difference between both methods is marked by red arrows. For this specific example of an axis-aligned cube, nearest neighbor sampling gives good results. However, a generic clip object that contains both smooth regions and sharp features can only be accurately handled by depth-based clipping.

Fig. 11 shows clipping in a nonilluminated medical MR data set. Fig. 11a uses a voxelized spherical clip object and Fig. 11b a voxelized cylindrical clip object. The clipped part

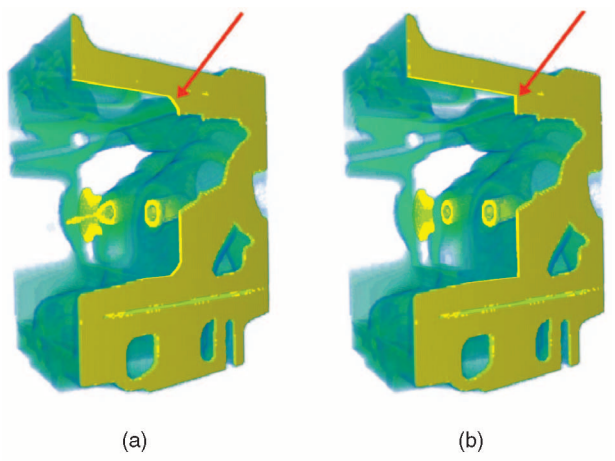
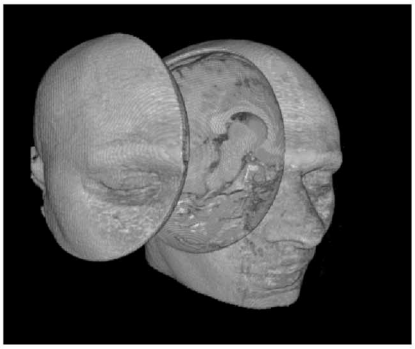
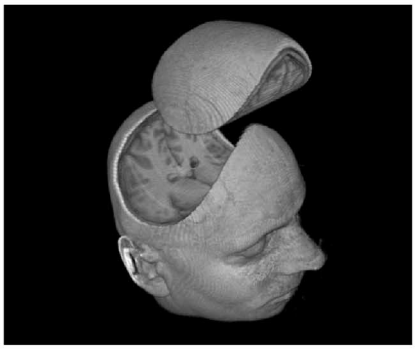


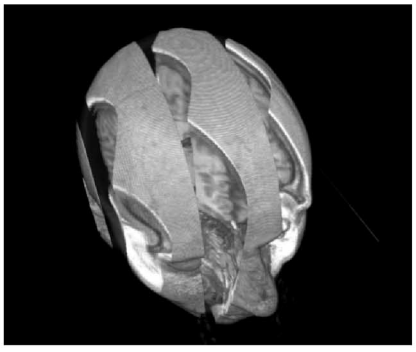
Fig. 10. Comparison of trilinear interpolation (a) and nearest neighbor sampling (b) for a voxelized cube-shaped clip object.



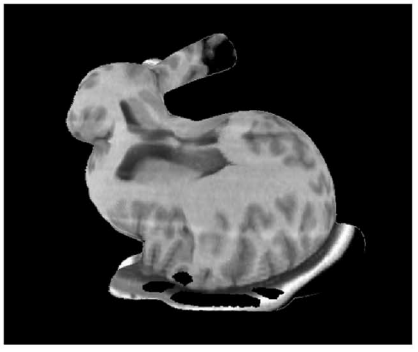
(a)



(b)



(c)



(d)

Fig. 11. Clipping for an MR data set. Volume visualization is employed without any lighting. In images (a)-(c), voxelized clip objects are used; image (d) illustrates depth-based clipping with the Stanford bunny as clip object.

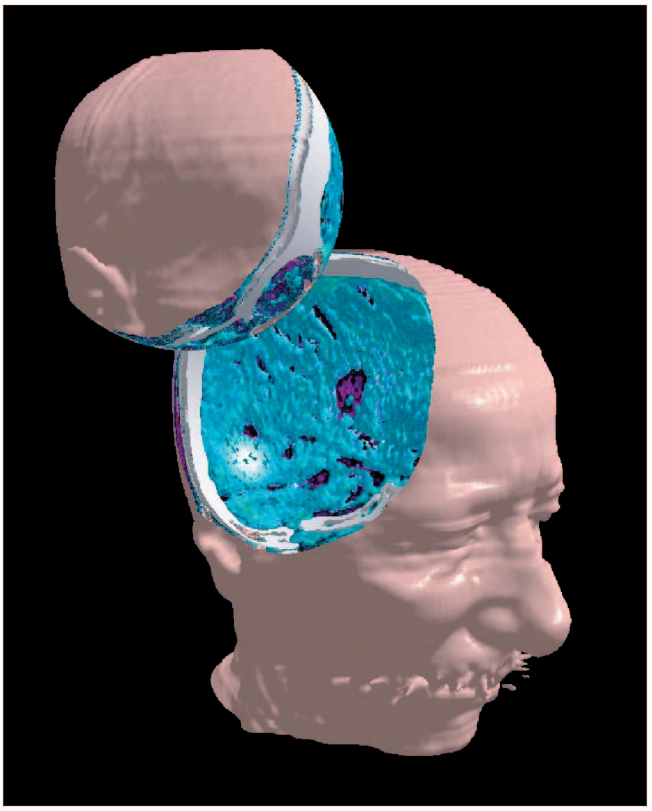


Fig. 12. Depth-based clipping in an illuminated CT data set.

of the volume is moved away from the remaining parts of the volume. Fig. 11c demonstrates that volumetric clipping is capable of processing rather complex clip geometries and topologies. Fig. 11d shows that depth-based volume clipping allows for geometries with a large number of triangles—such as the Stanford bunny.

Figs. 12 and 13 demonstrate the consistent combination of clipping and volume shading. Depth-based clipping via shadow testing (according to Section 3.5) is applied to a medical CT data set in Fig. 12. A specular highlight on the clipping surface is apparent and allows the viewer to recognize the orientation of the surface. Here, the transfer function is chosen to render the head almost opaque, i.e., the image resembles surface-based graphics. In contrast, the visualization of the orbital data set in Fig. 13 reveals both transparent and opaque structures. Fig. 13a shows the original data set. Fig. 13b and Fig. 13c are based on depth-based clipping by means of shadow testing, whereas Fig. 13d is based on volumetric clipping. This comparison demonstrates that the quality of the specular highlight is very high for surface-based shading in Fig. 13b and Fig. 13c. In contrast, the highlight in Fig. 13d is based on the gradients of the clip volume and shows artifacts caused by an inaccurate reconstruction of the gradients and by quantization. In Fig. 13b, an extended version of surface shading is applied to specifically mark the clipping surface for a fast recognition of its shape, position, and orientation. An additional ambient lighting term makes the clipping object visible as a dark-gray surface.

The implementation of our volume renderer is based on C++, OpenGL, and GLUT. It is platform-independent and

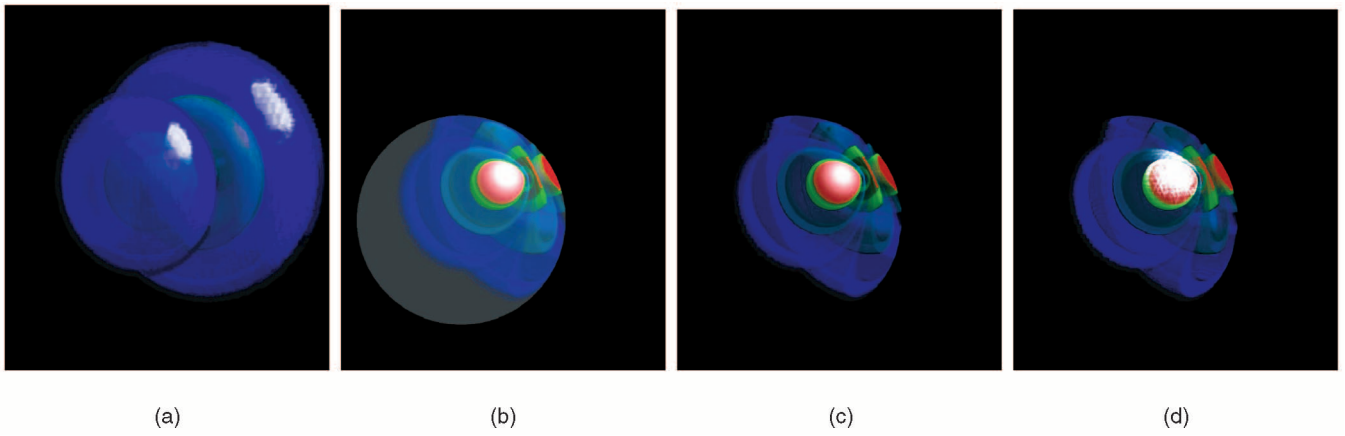


Fig. 13. Clipping in an illuminated orbital data set. Image (a) is without clipping; in (b) and (c), depth-based clipping is applied; image (d) is based on volumetric clipping. The clipping surface is additionally marked in dark gray in (b).

runs on Linux and Windows PCs. Our implementation makes use of NVidia's OpenGL extensions [24] for the GeForce 3/4. The performance measurements in Table 1 were conducted on a Windows XP PC with an Athlon XP 1800+ CPU and an NVidia GeForce 4 Ti 4600 graphics board with 128 MB texture memory. The size of the data set was 256^3 and the viewport sizes were 512^2 or $1,024^2$. A slicing approach for a 3D texture was chosen, preclassification by paletted textures was applied, and volume shading was disabled.

The performance numbers show that clipping based on a voxelized geometry causes rather modest additional rendering costs. The performance of depth-based clipping depends on the chosen implementation. On the GeForce 3/4, the shadow-testing approach is faster than the approach based on view frustum clipping. The main reason is the larger number of texture stages required by depth clipping. Because of the rather restricted functionality of the GeForce 3/4, a shift of depth values needs three texture stages. On the other hand, a lookup in a depth texture combined with a shadow test can be handled by a single texture stage. Another reason is that data can be directly copied from the z buffer into a depth texture, whereas the high-resolution texture for the depth-clipping approach can only be filled by data from main memory. Moreover, the early z test can improve the performance of shadow-based probing. Depth-based cutting by means of shadow testing is slower than shadow-based probing because a second depth texture has to be used. For volume shading, the above relative performance behavior with respect to different clipping approaches remains roughly the same; however, the overall performance is reduced due to the additional texture and fragment operations for post-classification and illumination. All mentioned effects on the rendering performance are dependent on the specific architecture of the GeForce 3/4 and might very well change on other GPUs.

TABLE 1
Performance Measurements in Frames per Second

Viewport size	512^2	1024^2
No clipping	28.0	8.8
Voxelized clip object	15.3	5.2
Depth-based probing (depth clipping)	8.5	2.5
Depth-based cutting (depth clipping)	7.1	2.2
Depth-based probing (shadow test)	18.5	5.6
Depth-based cutting (shadow test)	12.2	3.4

From our experiments with the different clipping techniques, we have learned that each method has specific advantages and disadvantages. The best-suited technique has to be chosen by the type of application and the requirements stated by the user.

The main advantage of the volumetric clipping approach is that it provides explicit control over each voxel of the clip volume. Therefore, arbitrarily complex objects can be used as clip geometry. Furthermore, only modest additional rendering costs are introduced due to the availability of multiple parallel texturing units—as long as the memory bandwidth is not the limiting factor. Moreover, the required per-fragment operations are available on most current GPUs.

The main disadvantage of the volumetric clipping approach is a possibly inaccurate reproduction of the clip geometry. Nearest neighbor sampling of the clipping texture results in visible, jaggy artifacts and a poor image quality. Trilinear interpolation, on the other hand, does not allow for arbitrary sharp geometric features of the clip object. For volume shading, the image quality can further suffer because of the sometimes inaccurate reconstruction of normal vectors from an interpolated gradient field. Moreover, the voxelization approach prohibits changing the shape of a clip object in real time. Affine transformations of the clip object, however, can be performed at interactive frame rates. Another problem is the additional texture memory requirements for the clipping volume. However, the size of the clipping texture can be chosen independently of the size of the volume to save texture memory.

An advantage of depth-based volume clipping is the high quality of the resulting images since clipping is performed with per-pixel accuracy. High image quality is even more noticeable for clipping in shaded volumes because the illumination of the clipping surface is derived from surface-based information and not from a volumetric gradient field. Aliasing problems related to texture mapping cannot occur because there is a one-to-one mapping

between pixels on the image plane and the texels representing the clip geometry. Arbitrary changes of the clip geometry cause no additional costs because no voxelization is required. The rendering performance is comparable to the voxel approach and allows interactive applications. Our depth-based volume clipping methods benefit from the fact that the clip geometry has to be processed only once per frame. This is an important difference from the method by Westermann and Ertl [11], which needs to update the values in the stencil buffer for each slice by rendering the whole clip geometry. This approach does not scale well for complex clip geometries or large volume data sets with many slices.

The main problem of depth-based volume clipping is the costly treatment of concave and complex clip geometries by means of multipass rendering. Therefore, we often restrict ourselves to single-pass depth-based algorithms even if concave clip geometries are used. From our experience, artifacts that might occur for concave objects used in conjunction with these single-pass methods are negligible in many applications. Once a high enough opacity is collected along a light ray, voxels further away from the viewer do not contribute to the final image. Often the artifacts caused by concave objects are located in these "hidden" regions.

7 Conclusion and Future Work

We have introduced clipping methods that are capable of using complex clip geometries. All methods are suitable for texture-based volume rendering and exploit per-fragment operations on the graphics hardware to implement clipping. The techniques achieve high overall frame rates and therefore support the user in interactive explorations of volume data. Depth-based clipping analyzes the depth structure of the clip geometry to decide which regions of the volume have to be clipped. Depth-based clipping is particularly well suited to produce high-quality images. In another approach, a clip object is voxelized and represented by a volumetric texture. Information in this volume is used to identify clipping regions. This approach allows us to specify arbitrarily structured clip objects and is a very fast technique for volume clipping with complex clip geometries. Furthermore, an optical model has been proposed to combine clipping and volume shading for consistent shading of the clipping surface. We have demonstrated how this model can be efficiently incorporated in the aforementioned clipping techniques.

In future work, it could be investigated how visualization applications can benefit from sophisticated volume clipping. In particular, medical imaging could be enhanced by including clipping geometries extracted from segmentation information. In this context, volume tagging could be of specific interest. Here, a voxelized "clip" object would not only contain information on the visibility property, but more detailed information such as an object identification number. This ID could help to apply different transfer functions to different regions of the data set. As another variation, the "clip" volume could contain continuously varying α values to allow for a smooth fading of volumetric regions. Furthermore, appropriate methods to visualize dense representations of 3D vector fields by means of volume clipping could be investigated. Future work could also deal with interaction techniques and user interfaces to specify clip geometries effectively.

From a technical point of view, we plan to adapt the clipping techniques to preintegrated volume rendering [25] and incorporate more advanced volume shading models. For example, efficient texture-based methods for volume shadows [26] and translucency [27] could be combined with volumetric clipping. Finally, the ideas of this paper could be adapted to the increasing functionality of future graphics hardware. In particular, the implementation of depth-based clipping could make use of floating-point textures and advanced fragment operations on future GPUs.

Acknowledgments

The authors thank Martin Kraus, Stefan Röttger, and Manfred Weiler for fruitful discussions and the anonymous reviewers for helpful comments to improve the paper. Special thanks to Manfred Weiler and Bettina Salzer for proof reading. This work was, in part, supported by the Landesstiftung Baden-Württemberg.

References

[1] H. Pfister, B. Lorensen, C. Baja, G. Kindlmann, W. Schroeder, L.S. Avila, K. Martin, R. Machiraju, and J. Lee, "Visualization Viewpoints: The Transfer Function Bake-Off," *IEEE Computer Graphics and Applications,* vol. 21, no. 3, pp. 16-23, May/June 2001.

[2] G. Kindlmann and J. Durkin, "Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering," *Proc. 1998 Symp. Volume Visualization,* pp. 79-86, 1998.

[3] J. Kniss, G. Kindlmann, and C. Hansen, "Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets," *IEEE Visualization Proc. 2001,* pp. 255-262, 2001.

[4] B. Fröhlich, S. Barrass, B. Zehner, J. Plate, and M. Göbel, "Exploring Geo-Scientific Data in Virtual Environments," *IEEE Visualization Proc. 1999,* pp. 169-174, 1999.

[5] W.R. Volz, "Gigabyte Volume Viewing Using Split Software/Hardware Interpolation," *Proc. 2000 Symp. Volume Visualization,* pp. 15-22, 2000.

[6] D. Weiskopf, K. Engel, and T. Ertl, "Volume Clipping via Per-Fragment Operations in Texture-Based Volume Visualization," *IEEE Visualization Proc. 2002,* pp. 93-100, 2002.

[7] K. Akeley, "Reality Engine Graphics," *SIGGRAPH 1993 Conf. Proc.,* pp. 109-116, 1993.

[8] B. Cabral, N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," *Proc. 1994 Symp. Volume Visualization,* pp. 91-98, 1994.

[9] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl, "Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping," *IEEE Visualization Proc. 1999,* pp. 233-240, 1999.

[10] A. Van Gelder and K. Kim, "Direct Volume Rendering with Shading via Three-Dimensional Textures," *Proc. 1996 Symp. Volume Visualization,* pp. 23-30, 1996.

[11] R. Westermann and T. Ertl, "Efficiently Using Graphics Hardware in Volume Rendering Applications," *SIGGRAPH 1998 Conf. Proc.,* pp. 169-179, 1998.

[12] P. Hastreiter, H. Çakmak, and T. Ertl, "Intuitive and Interactive Manipulation of 3D Datasets by Integrating Texture Mapping Based Volume Rendering into the OpenInventor Class Hierarchy," *Bildverarbeitung für die Medizin - Algorithmen, Systeme, Anwendungen,* T. Lehman, I. Scholl, and K. Spitzer, eds., pp. 149-154, Universität Aachen, Verl. d. Augustinus Buchhandlung, 1996.

[13] U. Tiede, T. Schiemann, and K.H. Höhne, "High Quality Rendering of Attributed Volume Data," *IEEE Visualization 1998 Proc.,* pp. 255-262, 1998.

[14] P. Bhaniramka and Y. Demange, "OpenGL Volumizer: A Toolkit for High Quality Volume Rendering of Large Data Sets," *Proc. Volume Visualization and Graphics Symp. 2002*, pp. 45-53, 2002.
[15] J. Diepstraten, D. Weiskopf, and T. Ertl, "Transparency in Interactive Technical Illustrations," *Eurographics 2002 Conf. Proc.*, pp. 317-325, 2002.
[16] C. Everitt, "Interactive Order-Independent Transparency," white paper, NVidia, 2001.
[17] A. Mammen, "Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique," *IEEE Computer Graphics and Applications*, vol. 9, no. 4, pp. 43-55, July 1989.
[18] P.J. Diefenbach, "Pipeline Rendering: Interaction and Realism through Hardware-Based Multi-Pass Rendering," PhD thesis, Univ. of Pennsylvania, 1996.
[19] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice.* Reading, Mass.: Addison-Wesley, 1990.
[20] F. Durand, "3D Visibility: Analytical Study and Applications," PhD thesis, Université Joseph Fourier, Grenoble I, July 1999.
[21] H. Hege, T. Höllerer, and D. Stalling, "Volume Rendering—Mathematical Models and Algorithmic Aspects," Technical Report TR 93-7, ZIB (Konrad-Zuse-Zentrum), Berlin, 1993.
[22] N. Max, "Optical Models for Direct Volume Rendering," *IEEE Trans. Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99-108, June 1995.
[23] P. Sabella, "A Rendering Algorithm for Visualizing 3D Scalar Fields," *Computer Graphics (SIGGRAPH '88 Conf. Proc.)*, vol. 22, no. 4, pp. 51-58, Aug. 1988.
[24] *NVIDIA OpenGL Extension Specifications*, M.J. Kilgard, ed., NVIDIA Corp., 2001.
[25] K. Engel, M. Kraus, and T. Ertl, "High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading," *Proc. Eurographics/SIGGRAPH Workshop Graphics Hardware 2001*, pp. 9-16, 2001.
[26] J. Kniss, G. Kindlmann, and C. Hansen, "Multidimensional Transfer Functions for Interactive Volume Rendering," *IEEE Trans. Visualization and Computer Graphics*, vol. 8, no. 3, pp. 270-285, July-Sept. 2002.
[27] J. Kniss, S. Premoze, C. Hansen, and D. Ebert, "Interactive Translucent Volume Rendering and Procedural Modeling," *IEEE Visualization Proc. 2002*, pp. 109-116, 2002.

Daniel Weiskopf received the MS degree in physics and the PhD degree in theoretical astrophysics from the University of Tübingen. He is a research assistant at the Visualization and Interactive Systems Institute at the University of Stuttgart. His research interests include visualization, computer animation and simulation, virtual environments, and special and general relativity. He is a member of the IEEE Computer Society and ACM SIGGRAPH.

Klaus Engel received the PhD degree from the University of Stuttgart, Germany. He received the Diplom (Masters) of computer science from the University of Erlangen in 1997. From January 1998 to December 2000, he was a research assistant in the Computer Graphics Group at the University of Erlangen-Nuremberg. From 2000 to 2003, he was a research assistant at the Visualization and Interactive Systems Institute at the University of Stuttgart. Since 2003, he has been a researcher for Siemens Corporate Research in Princeton, New Jersey.

Thomas Ertl received the master's degree in computer science from the University of Colorado at Boulder and the PhD degree in theoretical astrophysics from the University of Tübingen. Currently, he is a full professor of computer science at the University of Stuttgart, Germany, and the head of the Visualization and Interactive Systems Institute (VIS). His research interests include visualization, computer graphics, and human computer interaction in general, with a focus on volume rendering, flow visualization, multiresolution analysis, and parallel and hardware accelerated graphics for large datasets. He is the coauthor of more than 150 scientific publications and he served on several program and paper committees as well as a reviewer for numerous conferences and journals in the field. He is member of the IEEE Computer Society, ACM SIGGRAPH, and Eurographics.