Direct Volume Editing

Kai Bürger, Jens Krüger, and Rüdiger Westermann

Abstract—In this work we present basic methodology for interactive volume editing on GPUs, and we demonstrate the use of these methods to achieve a number of different effects. We present fast techniques to modify the appearance and structure of volumetric scalar fields given on Cartesian grids. Similar to 2D circular brushes as used in surface painting we present 3D spherical brushes for intuitive coloring of particular structures in such fields. This paint metaphor is extended to allow the user to change the data itself, and the use of this functionality for interactive structure isolation, hole filling, and artefact removal is demonstrated. Building on previous work in the field we introduce high-resolution selection volumes, which can be seen as a resolution-based focus+context metaphor. By utilizing such volumes we present a novel approach to interactive volume editing at sub-voxel accuracy. Finally, we introduce a fast technique to paste textures onto iso-surfaces in a 3D scalar field. Since the texture resolution is independent of the volume resolution, this technique allows structure-aligned textures containing appearance properties or textual information to be used for volume augmentation and annotation.

Index Terms—Volume editing, GPU, painting, carving, annotations.

1 Introduction

Interactive visual exploration of volumetric scalar fields is required in many different areas ranging from medicine and engineering to physics and biology. To support the exploration task, volume rendering techniques have been developed to a high degree of sophistication over the last decade. Today, direct volume rendering of data sets as large as 512^3 and beyond is possible at fully interactive rates on commodity desktop systems, and especially due to the rapid advancements in graphics hardware technology these capabilities are continually increasing.

Volume rendering is a powerful means for visualizing 3D scalar fields, and especially if used in combination with semi-automatic transfer functions and different rendering styles does it allow for an effective visual communication of complex structures in such fields and relationships between them. In practical applications, however, to improve the analysis process it is often desired to not only render the data but also to interactively edit this data. Examples thereof include the manual classification and segmentation of structures, the removal of structures to uncover regions of interest and thus to isolate important parts of the data, or the coloring of parts to emphasize relevant structures and to give extra information about them. Such mechanisms can help to effectively reveal and communicate the relevant information in 3D scalar fields and to create images that are easy to understand even by a non-experienced user.

Today we see, that the core functionality that is required to support the aforementioned mechanisms is available on recent graphics processing units (GPUs). Specifically, it is now possible to directly write into 3D textures on the GPU, and to efficiently apply local operations on the data stored in these textures, such as filtering or gradient computation. Thus, the time is ripe for opening a new area in volume visualization, which is concerned with the development of techniques for interactive volume editing. One of the research challenges here is to develop novel algorithms that are tailored to the specific GPU functionality, and which can directly be incorporated into interactive volume rendering tools to enable immediate visual feedback. This core methodology, if designed in a generic way without the restriction to a particular application, then has the potential to be used in a number of different scenarios. In particular, to support the editing process and to avoid putting the burden completely on the user, optional constraints can be integrated into these methods.

- *Kai Bürger is with the Technische Universität München, tum.3D, E-mail: buergerk@in.tum.de.*
- *Jens Krüger is with Scientific Computing and Imaging Institute, E-mail: jens@sci.utah.edu.*
- *Rüdiger Westermann is with the Technische Universität München, tum.3D, E-mail: westerma@in.tum.de.*

Manuscript received 31 March 2008; accepted 1 August 2008; posted online 19 October 2008; mailed on 13 October 2008.
For information on obtaining reprints of this article, please send e-mailto:tvcg@computer.org.

1.1 Contribution

The primary focus of this paper is the development of fast and flexible methods for user-guided volume editing, such as coloring, erasing, pasting, segmentation, and annotation. Our long-term goal is to realize a volume processing tool exhibiting similar functionality to current image processing tools, which allow the user to interactively perform a multitude of image adjustments and enhancements. To achieve interactivity, all of the algorithms proposed in this work run entirely on the GPU, and they have been integrated into a GPU-based volume ray-caster to provide immediate visual feedback. We introduce some novel ways to leverage advanced GPU functionality like geometry shaders and the possibility to directly render into 3D textures, and we effectively exploit computational and bandwidth capacities on recent GPUs. Therefore, all of the editing operations demonstrated throughout this paper were executed at frame rates of 100 fps and higher. In combination with novel mechanisms to perform these operations at sub-voxel accuracy, a framework for visibility-guided interactive volume editing is presented.

Some of the editing techniques we introduce can effectively be used for volume illustration, where the basic goal is to enhance the perception of structures in the data and the relationships between them by emphasizing important features. In particular, we extend the work on direct volume illustration by Bruckner and Gröller [5], in that we provide an alternative way to annotate structures in a volume data set and show the use of high-resolution selection volumes for sub-voxel editing effects. Even though our techniques are not restricted to volume illustration, this particular application demonstrates the high potential of using these techniques as basic methodology.

Our paper makes the following specific contributions:

- We present an efficient GPU realization of the volume painting method proposed by Bruckner and Gröller [5], and we demonstrate the use of this method for interactive volume coloring as well as structure elimination and enhancement. This method was used in Figures 1a and 1b, respectively, to color an iso-surface, to erase parts of it and to add additional structures to it.
- We extend on the idea of selection volumes and present a volume editing technique that is independent of the volume resolution. It edits on a high-resolution selection volume and can, therefore, be used to apply editing effects at sub-voxel accuracy.

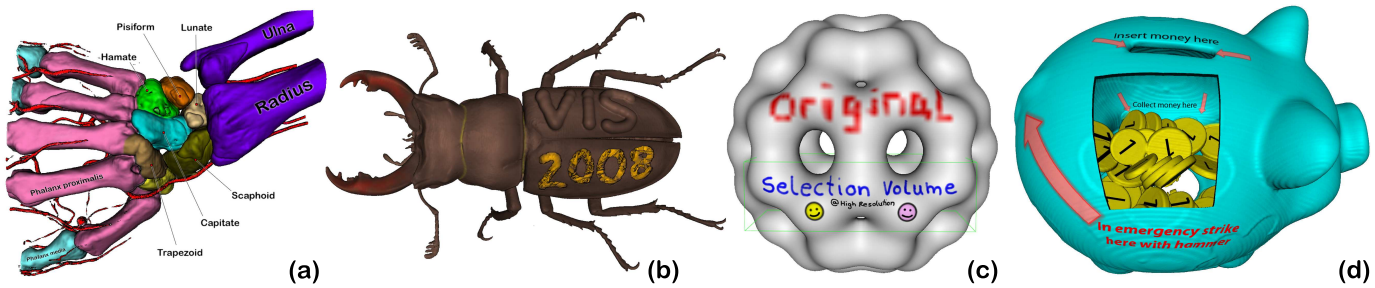

Fig. 1. We present basic methodology for interactive volume editing, such as coloring, carving, enhancement, and annotation at sub-voxel accuracy.

Figure 1c shows an example, where the upper text was painted on an iso-surface in the original volume, whereas the lower text was painted on an iso-surface in an upsampled sub-volume.

- We introduce *surface particles* to compute a local iso-surface parametrization. By using such particles, 2D textures can be mapped onto an iso-surface. This allows generating annotations that are aligned with an iso-surface, and which can effectively be used to give additional information about areal structures visible in the current view. Figure 1a shows a classified iso-surface which is enhanced by surface-aligned annotations.

- Building on the concept of surface particles, we present shape-aligned "see-through" textures to generate windowed cutaways on iso-surfaces in 3D scalar fields. By using such textures, occlusions can effectively be reduced and important internal parts of a volume can be exposed. This method was used in Figure 1d to interactively generate a cutaway view of the piggy bank data set.

The remainder of this paper is organized as follows. In Section 2 we review previous work that is related to ours. The specific volume coloring technique we use in this paper, as well as its efficient realization on recent GPUs, is presented in Section 3. Section 3.3 is dedicated to the use of this technique for structure removal and enhancement. In Section 4 we present high-resolution selection volumes and demonstrate their use for sub-voxel accurate volume editing. Section 5.1 introduces structure-aligned textures for volume augmentation and annotation. We conclude the paper with a discussion of the advantages and limitations of our work, and we present some ideas for future work.

2 Related Work

The idea to emphasize certain aspects of 3D data by using different rendering styles in the drawing of a single image of the data is at the core of non-photorealistic rendering techniques [9]. In the context of volume visualization this approach has been shown very useful to accentuate particular features in the data and thus to accommodate faster and better understanding of complex structures and the relationships between them. Over the last decade, a number of different so-called volume illustration techniques have been proposed, many of which have been integrated into GPU-based volume rendering systems to achieve interactive user-control.

Interrante et al. [16, 15] used curvature-directed strokes and dense sets of integral curves to convey surface shape. A general volume illustration rendering pipeline to enhance important features and regions was proposed by Ebert and Rheingans [8]. Viola et al. [36] suggested importance driven volume rendering to highlight interesting structures in volume data based on user-selected object importance. Different rendering styles including point stippling [27, 24], temporal domain enhancement [28], 2D texture synthesis on cross-sections of a volumetric model [29], and volumetric halos to improve depth perception of 3D structures [4] have been used to enhance the expressiveness of volume visualizations. A new approach that uses the shape of the object to be illustrated to control its rendering styles, and which also allows to adapt the objects shape to a given curve skeleton, was presented in [6].

Especially if used in combination with focus+context techniques to combine multiple aspects of the data into a single visual event [35, 13, 2, 22, 3], illustrative volume rendering has been shown to be very effective in communicating the essential information in complex volumetric data sets. An interactive system providing a toolbox of automatic illustration methods as well as focus+context mechanisms to enable selective exploration of volume data was presented by Bruckner et al. [5]. In particular, they introduced screen-space aligned annotations to add extra information about particular structures and selection volumes to locally modulate the appearance of a volume. Our work builds on these mechanisms and extends them towards a more general use for volume illustration.

Finally it should be mentioned, that there is also ongoing research on the user interfaces used to directly interact with volumetric data sets in virtual environments, including aspects of 3D haptic input devices as well as haptic rendering techniques. Even though these aspects are important in the design of a volume editing tool, they are not addressed in our current work. Instead, we abstract from the input devices used and rather focus on the editing operations triggered by the user via these devices. Due to this reason we will not attempt to review the vast body of literature related to these issues, however, in [1, 14, 30, 32] some devices and interaction models are discussed and many useful references on these subjects are given.

3 Volume Coloring

The specification of appearance properties of volume data is typically performed via color transfer functions. Based on the seminal work by Kindlemann and Durkin [17] on the design of feature-specific transfer functions that can be derived automatically from a data classification using first and higher-order statistics, such approaches have now been developed to a high degree of sophistication. Nevertheless, automated classification of volumes remains a challenging task, and semi-automatic techniques which allow the user to interactively guide the classification process often result in a more accurate assignment of appearance properties. Examples thereof include the user-guided selection of seed voxels to initialize automated region-growing [20] or more sophisticated segmentation algorithms like the random walker [10], the dual-domain approach of Kniss et al. [19, 18], or the machine-learning approach by Tzeng et al. [34], where a transfer function is iteratively refined from user-defined segmentations in 2D volume slices.

To support semi-automatic classification and segmentation of 3D volume data we now describe an interactive technique for voxel coloring. This technique works in the 3D domain, and it thus allows the user to consider the 3D shape of the structures to be colored as well as the spatial relationships between them. Figure 2 demonstrates the application of this approach for the classification of a human skull. The proposed technique has been integrated into a GPU-based volume raycaster, enabling the user to obtain immediate visual feedback about the result of the issued operations. Later in the text we show how to overcome the restriction of volume coloring to the initial volume resolution by exploiting selection volumes for sub-voxel coloring.

3.1 3D Texture Painting

Initially, a 3D scalar field of size T_x, T_y, T_z is loaded into a 3D texture—the source texture—on the GPU. Scalar values are mapped to color and opacity via a selected transfer function. If the user only wants to paint on an iso-surface in the scalar field, a one component 3D texture is used instead of a RGBA texture. Coloring always works on an additional 3D texture—the color texture—on the GPU, into which the user paints with the selected color. In iso-surface coloring this texture is initialized with a constant material color, otherwise it is initialized with the source color values. Working on such a copy allows for a special paint mode in which the paint operation resets the color by copying respective values from the source texture. In iso-surface painting, colors are reset by zeroing.

The 3D color texture is rendered using texture-based volume raycasting [23], i.e., by sampling the texture along the rays of sight and by blending color and opacity contributions according to the selected blend equation. In iso-surface rendering, sampling is performed in the source texture. Once the iso-surface is hit along a ray, the surface normal at this position is fetched from a pre-computed normal map and a local lighting model is evaluated. In this model, the color at the sample position in the color texture is used as material color.

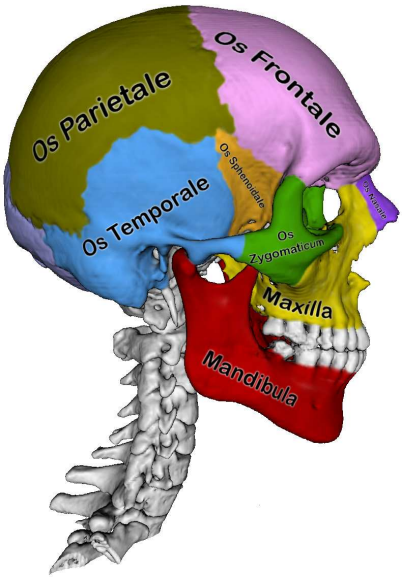

Fig. 2. An iso-surface in the visible human head data set is shown. The surface was colored to emphasize the anatomy of the human skull. Surface-aligned annotations, described in Section 5, were added.

Upon initialization, the user starts painting the volume with a virtual brush. To position the brush in 3D space we either use a simple mouse-based interface or a six degree-of-freedom input device, i.e. a PHANToM Desktop Device Premium 1 from Sensable Technologies. This also allows us to give haptic feedback to the user, for instance, if painting is on an iso-surface and force feedback indicates that the brush touches the surface. To detect a contact between a surface and the brush we simply test the brush center point for being in close proximity to the surface, i.e. by sampling the volume at this point and testing whether the value is closer to the iso-value than a given tolerance. If this is the case, force feedback along the inverse gradient direction at this point is issued.

In our work we use a sphere-like volume brush for painting, which means that voxels closer to the brush center point than the selected sphere radius are painted with the current paint color. To manipulate the color of a voxel at position P, indicated by Color_P, we use the paint equation proposed in [33, 12]:

$$\text{Color}_P = \textbf{lerp}(\text{Color}_C \ \textbf{OP} \ \text{Color}_P, \text{Color}_P, G), \tag{1}$$

The brush shape G is set such that a spherical color falloff with increasing distance to the brush center point is simulated:

$$G = \begin{cases} 0, & \text{if } |P_C - P| > m \\ F(|P_C - P|) & \text{else.} \end{cases} \tag{2}$$

Here, **OP** is one of a number of operations like REPLACE, ADD, or BLEND, which can be selected to modulate the initial volume color, P_C is the position of the center point, Color_C is the brush color, and m is the support of a user-defined falloff function F, which is used to simulate smooth color fading.

When using a volume brush to color a volume data set, the color of every voxel contained in the brush volume has to be updated according to the selected color modulation function. In principle, the color update can be performed on the CPU, requiring the modulated texture to be reloaded onto the GPU. Even if it is possible to only replace those parts of the GPU texture that were affected by the coloring operation, this strategy still results in significant bandwidth requirements due to frequent data uploads to the GPU in the course of painting. To overcome this limitation, we propose a novel technique that runs entirely on the GPU and minimizes CPU-GPU data transfer.

3.2 GPU Implementation

To efficiently update 3D texture elements that are affected by a coloring operation, we exploit novel features of current Direct3D 10 class graphics hardware. Specifically, we use the geometry shader to create geometry on the GPU, we employ new functionality to update slices of a 3D texture directly on the GPU, and we utilize instanced render calls to reduce the number of calls that have to be issued from the application program. In Figure 3 an overview of the pipeline setup for rendering into a 3D texture is shown.

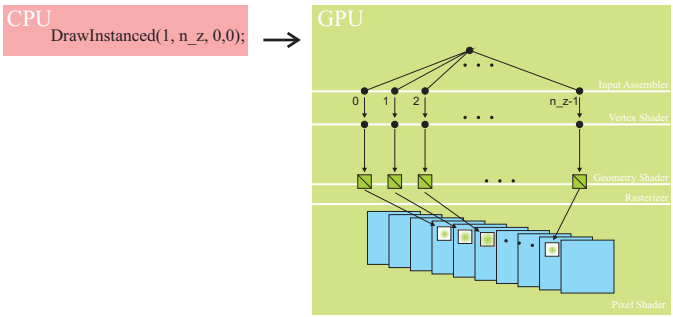

Fig. 3. Illustration of the pipeline setup for painting into a 3D texture on the GPU. A single vertex is issued by the application program, and it is duplicated by the input assembler. In the geometry shader, every point is amplified to one quadrilateral, which in turn is sent to the rasterizer. The rasterizer uses slice IDs to route generated fragments into corresponding 3D texture slices. In the pixel shader the fragments are colored with respect to the selected modulation function.

Before the painting process is started, the user selects the specific brush parameters including the cutoff radius m used in Equation 2. From this radius the extend of the brush bounding box in local texture coordinate space is computed, yielding the size $n_x \times n_y \times n_z$ of the sub-volume that is affected by the coloring operation. These values are computed on the CPU and sent to the GPU as constant shader variables. To compute the position of the brush center point P in local texture coordinates in the range [0,1], we either use the coordinate returned by the 3D input device, or, if painting is on an iso-surface, it can also be determined from the fragments depth under the mouse cursor.

The application program then renders into a viewport of size T_x, T_y. A single vertex—with a coordinate equal to P scaled by T_x, T_y, T_z—is sent to the GPU, where it is rendered as instanced geometry with instance count n_z. This causes the GPU to generate a stream of n_z vertices, all of which carry the position P and an *instance ID* running

from 0 to $n_z - 1$. These vertices are passed through the vertex shader to the geometry shader, which, for each incoming vertex, spawns a quadrilateral centered at P_x, P_y and covering $n_x \times n_y$ pixels. The ID of the 3D texture slice into which this quadrilateral is to be rendered is computed as

$$SID = P_z - \frac{n_z}{2} + IID, \tag{3}$$

where *IID* is the instance ID of every vertex. This slice ID is used by the rasterizer to direct the fragment into the corresponding z-slice of the 3D texture. In the pixel shader, for every fragment its distance to the brush center is computed and Equations 1 and 2 are evaluated. Updated color values are then written into the respective position of the 3D color texture slice, and the updated texture can immediately be used in the rendering pass.

3.3 Structure Removal and Enhancement

The method proposed in the previous section can efficiently be used to paint color into a volume. Moreover, it provides a means to interactively erase parts from the volume and to add new structures to it. Erasing is performed by painting voxels with zero opacity, thus making structures completely transparent. Even though the erasing operation is conceptually simple, it does provide a very powerful means to interactively create cutaway views. In particular it can be used when traditional volume cutaway techniques have difficulties, e.g., when occluded and occluding structures are close together and have similar material properties. Figure 4 shows such a case and a cutaway view that was generated by our method. Without using a data segmentation or a highly detailed clip geometry that can accurately separate structures from each other, in such scenarios the automated generation of a cutaway view remains a challenging task.

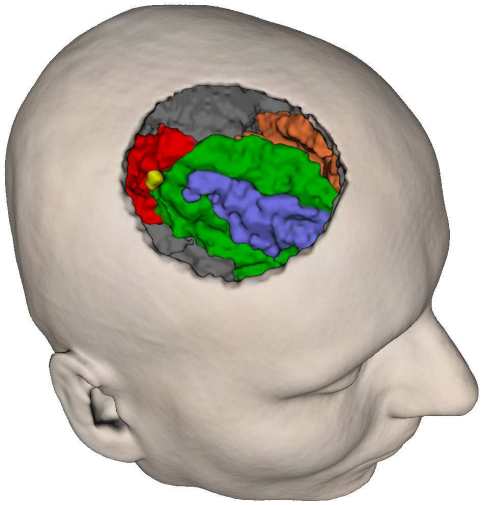

Fig. 4. Parts of bone iso-surface in a MRI data set were removed manually to reveal interior brain structures.

In the current implementation, new structures can only be added to iso-surfaces in the scalar field, i.e., if iso-surface rendering is performed. This operation is realized by a slight change of the color modulation function. Instead of replacing or modulating the colors stored in the 3D color texture, a density offset is painted into the source volume. By adding offsets of different strength and different size and shape, a number of editing effects can be achieved (see Figure 5).

When erasing or adding iso-surface structures, surface normals have to be updated accordingly. This is accomplished by a) finding all voxels in the pre-computed normal map that are contained in the brush volume, b) re-computing the normals using central differences in the source volume, and c) writing updated normals into the normal map. Steps a) and c) are performed in exactly the same way as described for volume coloring, with the only difference that the brush volume has to be slightly enlarged to capture all affected voxels.

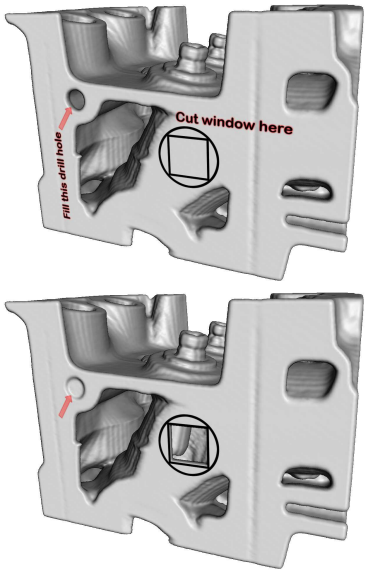

Fig. 5. Interactive volume editing was used to manually remove structures from an iso-surface and to add structures to it.

4 Selection Volumes

The volume coloring method as described so far restricts the accuracy of the coloring process to the resolution of the given volume data set. This allows one to assign voxel properties on a per-voxel basis, but the method is not capable of assigning such properties at sub-voxel accuracy. On the other hand, in particular if color painting is used to manually segment objects in the data, sub-voxel accuracy is required to determine correct segment boundaries. Similar to surface-based segmentation methods, where the mesh is not constrained to lie on voxel boundaries, our goal is to provide a much higher spatial resolution in regions where the user expects voxel-based classification to fail.

For this purpose we use selection volumes as introduced by Gröller [5], who stated, that "A selection volume specifies a particular structure of interest in a corresponding data volume. It stores real values in the range [0,1] where zero means not selected and one means fully selected.". A selection volume has the same spatial resolution as the original volume and its voxel values are used to modulate the initial data values. To make selection volumes applicable for data segmentation, we extend them in several ways: Firstly, in addition to extent and position the user can select the resolution of the selection volume. Secondly, the selection volume is "filled" with data values by resampling the source texture. It can thus be seen as an upsampled version of a sub-volume, and it is accompanied by a color volume of equal resolution to support voxel editing. Thirdly, the GPU volume ray-caster, which is used to render the original volume and the selection volume in combination, is adapted appropriately. This means, that the ray-caster not only finds the intersection points between the rays and the selection volume but also adapts the step size within this volume to its resolution. In iso-surface rendering, a uniform step size is used to avoid cracks at selection volume boundaries.

In Figure 6 we illustrate the use of selection volumes for sub-voxel classification, segmentation, and modeling. The leftmost image shows two voxel-sized structures that have been segmented manually in a selection volume. Due to the increased resolution of this volume, object boundaries can be resolved at very high accuracy. In the middle images, structures in the interior of a volume were classified by using a particular color transfer function. In the right image a high-resolution selection volume was used to obtain smooth structure boundaries. The rightmost image shows the effect of iso-surface enhancement in a high-resolution selection volume and the low-resolution base volume. Text was painted onto an iso-surface by manually adding density offsets into the respective source textures.

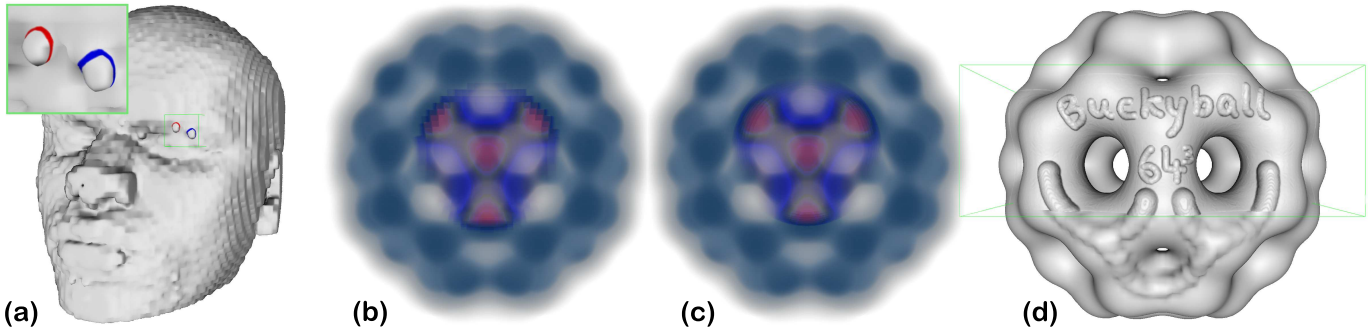

Fig. 6. The use of selection volumes is demonstrated: a) two small features are segmented at sub-voxel accuracy. b) a sub-volume at the initial and c) a much higher resolution is rendered with a different transfer function than the initial volume. d) editing effects on an iso-surface in the initial volume and a high-resolution selection volume.

4.1 Upsampling

To build a selection volume two different strategies are pursued. For direct volume rendering, voxel colors are tri-linearly interpolated in the initial color texture. For iso-surface rendering, a piecewise quadratic tensor product spline is used for resampling the source texture (see Figure 7). This results in a C^1-continuous quasi-interpolant exhibiting a smooth gradient field.

Denoting initial samples with v_i in voxel coordinates (i.e., ranging from 0 to $N-1$ for N voxels), additional samples at positions $x \in [i-0.5, i+0.5[$ are computed in two steps. First, intermediate values $A := 0.5\,(v_{i-1}, v_i)$ and $B := 0.5\,(v_i, v_{i+1})$ are computed. Then, a quadratic Bézier-spline with the control polygon A, v_i, B is constructed using the DeCasteljau algorithm. Thus, at x the associated index i has to be computed first by rounding to the next integer, i.e.,$i := \lfloor x+0.5 \rfloor$. The parameter p_i at which to evaluate the spline is then given as $p_i(x) := 0.5 + x - i$. Observing that the interpolation to compute A is collinear with the interpolation between A and v_i (and analogously for B and v_{i+1}), only two linearly interpolated fetches are necessary. These fetches can be performed by the GPU as $A' := \mathrm{lerp}\,(v_{i-1}, v_i, 0.5 + 0.5 \cdot p)$ and $B' := \mathrm{lerp}\,(v_i, v_{i+1}, 0.5 \cdot p)$, where $\mathrm{lerp}(a,b,c) := a + c \cdot (b-a)$. Finally, the second stage of the DeCasteljau algorithm to yield the final value $v_{res} := \mathrm{lerp}\,(A', B', p)$ is computed in a pixel shader.

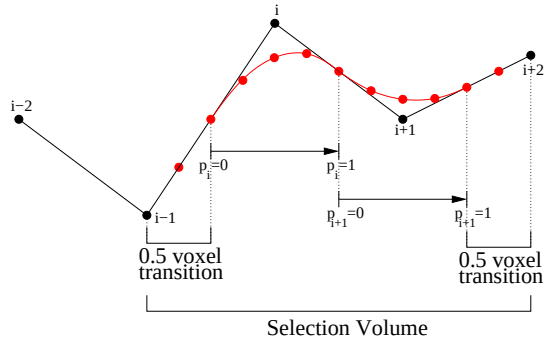

Fig. 7. The piecewise quadratic spline used for upsampling a selection volume.

Since the interpolated nodes lie halfway between the samples of the initial volume, we introduce a transition region that is half a voxel wide (with respect to the initial grid). In this region, tri-linear interpolation in the source texture is performed to guarantee C^0 continuity between the selection volume and the source volume. In the interior, the selection volume is built by tri-quadratic quasi-interpolation in the source texture, and a smooth normal map is computed on-the-fly from this volume. Figure 6(d) demonstrates the fine editing details that can be achieved by applying the operations described so far on a high-resolution selection volume.

In general, selection volumes can be used to add fine structures or color details to a 3D volume or an iso-surface in it. Selection volumes can thus be used to directly paint additional text on a surface, which provides a general means for adding surface-aligned annotations. However, as writing text on a curved surface in 3D is rather cumbersome, we propose an alternative GPU method to automatically align 2D textures containing text or other annotations on an iso-surface. For a good description of the process to be used to automatically place screen-space annotations we refer the reader to [5].

5 Surface Particles

We start our description by introducing GPU surface particles, which are used to map a 2D grid consisting of vertices and edges between them onto an iso-surface, i.e., to find a local surface parametrization. Our approach is similar in spirit to the one proposed by Ropinski et al. [31], but, in contrast, it is performed directly in 3D object space, and it operates entirely on the GPU. The 2D grid is rendered on top of the iso-surface as a textured polygon mesh. The texture contains the annotation to be used, for instance, a bit-mapped text or a pattern indicating a particular property.

A surface particle can be thought of as a particle moving on the surface. The direction of the movement is given by an external direction field that is defined by the user when placing the annotation. In any case, to move a particle on the surface we compute its trajectory $P(u)$ in a vector field $\vec{v}$, starting at an initial position (x,y,z) on the surface. This requires to solve the ordinary differential equation:

$$\frac{\partial P}{\partial u} = \vec{v}(P(u)) \tag{4}$$

with initial condition $P(0) = (x,y,z)$. To numerically solve this equation we employ classical Euler integration using a fixed step size Δu. For a thorough overview of particle tracing in vector fields let us refer here to the state-of-the-art report by Laramee et al. [25], and to [21] for the efficient implementation of particle tracing on programmable GPUs.

It is clear, that in general the numerical integration brings away the particle from the surface. Even if the vector field is everywhere defined in the local surface tangent plane, a particle is moving away from the surface in non-planar regions. To avoid this behavior, after every integration step we trace the particle back onto the surface, resulting in the following steps that have to be performed:

- **Integration** From the previous particle position, P, and the velocity at this position, $\vec{v}$, the new position $P' = P + \Delta u \cdot \vec{v}$ is computed. In the very first iteration $\vec{v}$ is set to zero.
- **Backtracing** P' is corrected by tracing the particle back onto the selected iso-surface.

- **Vector lookup** The velocity vector $\vec{v}$ at position P' is determined. This can be as simple as a texture lookup into a 3D vector field, or a 2D vector field if a surface parametrization exists, or it can be a more complex computation such as a curvature estimation.

While it is clear how to perform particle integration and vector lookup, the method to trace particles back to the surface requires some further explanation. In principle, moving it back onto the surface would require to bend the line segment connecting the current and the fixed previous particle position around the surface, thereby constraining the bending to the plane defined by this line segment and the surface normal at the previous position. Since this approach requires some exhaustive computations, we approximate it by iteratively correcting the current position towards the surface, thereby assuming the surface to be locally flat. Figure 8 illustrates this approximation for a particle that has left the surface after integration.

Back-tracing is performed by using the surface normal at the previous position, i.e. the gradient of the scalar field at this position, scaled by the difference between the scalar values at the previous and the current position. The direction of this vector determines whether the current position is inside the surface or outside. Note that using the normal at the current position is not feasible in general, since this point is not on the surface and the normal at this point may be affected by noise. Given this direction, the current particle is traced from the current position into this direction until the difference between the scalar values at the corrected position and the selected iso-value drops below a user-given tolerance. In this case we have reached the surface and terminate the correction. If the particle crosses the iso-surface, which is indicated by increasing difference between the scalar value at the particle position and the iso-value, the step size is halved and the trace is restarted at the last position.

The accuracy of the proposed method depends on the local curvature of the iso-surface. The less planar the surface is, the higher can be the length distortion of a line segment connecting the previous and the current point. The reason therefore is, that we only consider the normal at the previous point to determine the direction into which the particle is corrected. This problem could be alleviated by also considering the curvature direction in the plane spanned by the previous surface normal and the advection direction, but as the step size we use for particle integration is typically small, i.e. in the order of the voxel size, in our experiments length distortions did not result in any noticeable artifacts.

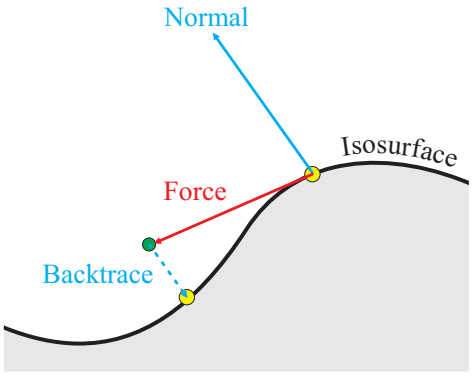

Fig. 8. One particle advection step is illustrated: Firstly, the particle is moved into the direction of the vector field (red) to an intermediate position (green). In the next step it is traced into the direction of the previous normal vector until it reaches the surface.

5.1 Volume Annotations

Volume annotations in the form of arrows and labels have a long history in hand-made technical and medical illustrations. Textual annotations are typically used in two different ways. They are either placed directly on the surface of a structure—aligning their shape to the surface shape—or they are placed in screen-space close to the image of a structure, and they are then connected to the structure with a line. In general, the former method has the advantage that annotations remain fixed to a structure when the user interacts with the volume, while free-floating labels have to be rearranged in screen-space to avoid overlapping annotations, crossing of connecting lines, or placements too far away from the structure. Free-floating annotation, on the other hand, are advantageous for pointing to small structures which do not cover enough space on screen to allow the user to read the annotation on it. Therefore, our system supports both approaches to annotate volumes, and it thus allows the user to flexibly select the appropriate choice.

By using surface particles we can now construct a regular grid, which is aligned with an iso-surface and can be textured with an arbitrary annotation. As the process is performed entirely on the GPU, the user can interactively place high-resolution annotations in the volume. To start the process, the user first selects a texture, the annotation texture, which is to be used as annotation. Then, some additional information has to be specified:

- The position on the iso-surface where the annotation is to be centered.
- The orientation of the annotation.

To specify the annotation center point the user picks a point on the iso-surface. The orientation of the annotation texture is specified by picking a second point and by interpreting the vector from the first to the second point as the u-axis of the local (u,v) surface parametrization. In the following, we will call this vector the orientation vector. Given this information, a set of surface particles is traced to generate a grid that is aligned with the surface.

At first, two surface particles are spawned at the annotation center point. One of them is traced along the orientation vector, and the other one is traced into the inverse direction. Both particles are traced for a number of equidistant steps and their intermediate positions are written into a GPU render target. Both the number of steps and the step size in voxel units can be selected by the user.

At every particle position the direction vector moving the particle along the surface is computed from the direction vector at the previous position. Starting with the normalized projection of the orientation vector into the tangent plane at the annotation center point, at every upcoming position the same procedure is performed with the previous direction vector. That is, for a particle at position P^u we compute a tangent frame consisting of three mutually orthonormal vectors: $\vec{N}$, the surface normal, $\vec{F}$ the direction vector in the local tangent plane, and $\vec{B}$, the cross product between $\vec{N}$ and $\vec{F}$. Initially, $\vec{F}$ is computed from the given orientation vector, $\vec{O}$, as follows:

$$\vec{F} = \frac{\vec{O} \times \left(\vec{N} \times \vec{O}\right)}{|\vec{O} \times \left(\vec{N} \times \vec{O}\right)|} \tag{5}$$

In the next advection step, $\vec{O}$ is set to $\vec{F}$, and the projection is with respect to the current tangent plane. Surface normals are computed by tri-linear interpolation of the gradients at adjacent voxel centers. Finally, the particle is advected using $\vec{F}$ and it is then traced back to the surface as described in the previous paragraph.

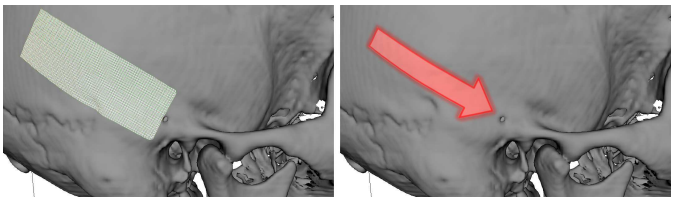

Fig. 9. Surface-aligned annotations. Left: the annotation grid. Right: an annotation texture that is mapped onto the grid.

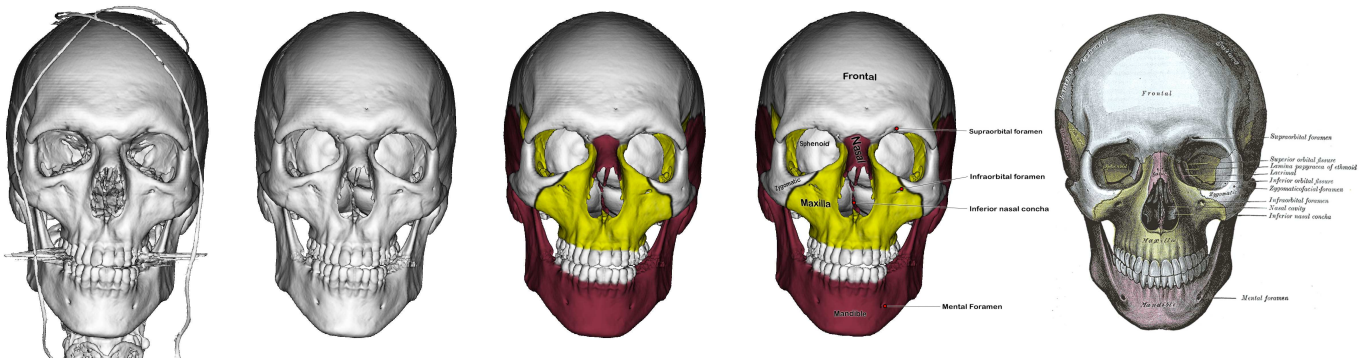

Fig. 10. Images from a volume editing session. From left to right: the initial data set, structures are removed, surface color is applied, annotations are added. The rightmost image is taken from the classical anatomy book "Gray's Anatomy" by Henry Gray [11] for comparison.

After the two surface particles that were released at the annotation center point have been traced for n steps, a number of $2n+1$ surface points are stored in a GPU render target. If these points are connected they form a line on the surface, centered at the annotation center point and oriented along the annotation direction. To expand this "line" to a full 2D grid, at every point we trace two additional surface particles into direction B and into the inverse direction. Tracing these particles for m steps results in a set of $(2m+1)\cdot(2n+1)$ points, from which a regular triangular annotation grid is built (see Figure 9). All grid points are rendered into a vertex buffer, which is then used to render the grid using an appropriate index buffer residing in GPU memory. The grid is textured with the selected annotation texture, and it is rendered before ray-casting the volume to initialize the depth buffer. To avoid depth fighting between the iso-surface and the annotation grid, the grid is slightly shifted towards the viewer.

5.2 Windowed Cutaway Views

In this paragraph, we show how to efficiently create a shape-aligned windowed cutaway section on an iso-surface by exploiting an annotation grid as introduced before. In technical illustrations, cutaways are often used to reduce occlusions and expose important internal parts. There is a vast body of literature related to this issue that we will not attempt to overview here, however, Diepstraten et al. [7] and Li et al. [26] discuss some of the mechanisms to automatically generate cutaway views and provide many useful references on this subject.

Starting with such a surface-aligned grid, we proceed in two stages. Firstly, we duplicate the mesh and displace the vertices of the copy along the inverse surface normal direction at the center vertex. The length of the displacement can be selected by the user to generate thin or thick cutaway sections. Secondly, both meshes are connected along their borders to build a closed mesh. This mesh is then used as a clip geometry as proposed by Weiskopf et al. [37], and it is directly incorporated into the texture-based volume ray-caster.

Prior to ray-casting, we render a layered depth-buffer of the mesh from the current view. During volume rendering, every ray first samples these buffers and then tests all samples along the ray for being inside or outside the mesh, i.e. by testing whether a sample is in-between a front and a back face of the cutaway mesh. Samples inside the mesh do not contribute to the final ray color, thus cutting away the volume contained in it. Figure 1(d) demonstrates the use of a shape-aligned cutaway to expose internal parts of a volume.

6 Performance Analysis

Throughout this paper we have shown a number of different effects that were generated by the proposed volume editing techniques. A typical use of these techniques is demonstrated in Figure 10, where a human skull data set was interactively processed and augmented to obtain an illustrative image as shown in "Gray's Anatomy" [11]. In the following, we investigate the performance of these techniques in more detail. Timings were performed on a 2.4 GHz Core 2 Duo processor and an NVIDIA 8800GTX graphics card with 768 MB local video memory. Image generation was done at 1280×1024 resolution. Regardless of this extreme resolution, for all models shown we achieve real-time performance with update rates of 50 fps and higher, including editing and rendering.

All brush-based editing effects like coloring, erasing, and adding, as well as resulting normal map updates, were executed in less than 3 ms up to a brush extend of 64^3 voxels. The times it takes to build a selection volume at different resolutions, i.e., from $(3\times2)^3$ to $(64\times8)^3$, is given in Table 1. As can be seen, even at a resolution as high as 128^3, GPU-based resampling is still capable of achieving interactive rates.

	Covered voxels				
Scaling	3^3	11^3	19^3	32^3	64^3
2	0.14	0.19	0.24	0.51	2.7
4	0.16	0.31	0.76	2.5	17.9
8	0.2	1.0	4.29	17.1	134.6

Table 1. Timing statistics for tri-quadratic iso-surface and trilinear color resampling. All times are given in milliseconds.

Finally, we measured the time it takes to construct a surface-aligned annotation grid by means of the method described in Section 5. Table 2 shows respective times for varying grid sizes. From these timings it can be concluded, that the proposed method is fast enough to allow for interactive placements of annotation textures on high-resolution surface structures. In particular, since the rendering of these textures only consumes an insignificant amount of time, many of them can be used simultaneously on a single object.

	Gridsize			
	11^2	21^2	41^2	81^2
Time (in ms)	1.6	2.0	3.6	14.7

Table 2. Timing of surface-aligned construction of annotation grids.

7 Conclusion and Future Work

In this paper, we have presented a number of GPU-based techniques for interactive volume editing. By efficiently using novel functionality on recent GPUs, we have developed a technique for interactive volume painting. We have further shown that this technique provides a powerful means to erase structures in a volume and thus to isolate features in it. In combination with high-resolution selection volumes these techniques can effectively be used for manual volume segmentation at sub-voxel accuracy. We have also introduced structure-aligned annotations to supplement classical free-floating annotations that are placed in screen-space, and we have demonstrated how to utilize this approach to interactively create windowed cutaway views. In particular, as all of these operations are performed in the 3D domain, with

immediate visual feedback provided, they are very intuitive to use and allow the user to quickly observe the relationships between relevant features in the data.

In the future we will further extend some of the proposed techniques: Firstly, we will develop semi-automatic volume segmentation techniques by combining manual segmentation as proposed with automatic techniques on the GPU, like the random walker approach. We believe that such a combination can considerably improve the segmentation process, both with respect to accuracy and speed. Secondly, we are aware that the construction of structure-aligned annotations as described in this work can produce distortions and even folds in highly curved regions. In the future we will investigate the use of constraint mass-spring systems on the GPU to avoid such artifacts. Thirdly, we will pursue research on the integration of focus+context approaches into direct volume editing techniques. In this way, additional visual cues can be provided to the user, resulting in an improved understanding of complex structural relationships in 3D.

REFERENCES

[1] R. S. Avila and L. M. Sobierajski. A haptic interaction method for volume visualization. In *IEEE VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 197–205., 1996.

[2] S. Bruckner, S. Grimm, A. Kanitsar, and E. Gröller. Illustrative context-preserving volume rendering. In *EuroVis*, pages 69–76, 2005.

[3] S. Bruckner, S. Grimm, A. Kanitsar, and M. E. Gröller. Illustrative context-preserving exploration of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1559–1569, 2006.

[4] S. Bruckner and E. Grller. Enhancing depth-perception with flexible volumetric halos. *IEEE Transactions on Visualization and Computer Graphics*, 13(6), 2007.

[5] S. Bruckner and M. E. Gröller. Volumeshop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization 2005*, pages 671–678, Oct. 2005.

[6] W. Chen, A. Lu, and D. S. Ebert. Shape-aware volume illustration. *Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(7):705–714, 2007.

[7] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive cutaway illustrations. *Computer Graphics Forum (Proceedings of Eurographics 2003)*, 22(3):523–532, 2003.

[8] D. Ebert and P. Rheingans. Volume illustration: Non-photorealistic rendering of volume models. In *IEEE Visualization 2000 (Conference Proceedings)*, pages 195–202, 2000.

[9] B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. AK Peters Ltd., 2001.

[10] L. Grady, T. Schiwietz, S. Aharon, and R. Westermann. Random walks for interactive organ segmentation in two and three dimensions: Implementation and validation. In *MICCAI*, 2005.

[11] H. Gray. *Gray's anatomy*. Running Press, 1901.

[12] P. Hanrahan and P. Haeberli. Direct wysiwyg painting and texturing on 3d shapes. *SIGGRAPH Comput. Graph.*, 24(4):215–223, 1990.

[13] H. Hauser, L. Mroz, G. I. Bischi, and M. E. Gröller. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, 2001.

[14] M. Ikits, J. D. Brederson, C. D. Hansen, and C. R. Johnson. A constraint-based technique for haptic volume exploration. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 35, 2003.

[15] V. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *ACM SIGGRAPH*, pages 109–116, 1997.

[16] V. Interrante, H. Fuchs, and S. Pizer. Illustrating transparent surfaces with curvature-directed strokes. In *IEEE Vis*, pages 211–218, 1996.

[17] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 79–86, 1998.

[18] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.

[19] J. Kniss, P. McCormick, A. McPherson, J. Ahrens, J. Painter, A. Keahey, and C. Hansen. Interactive texture-based volume rendering for large data sets. *IEEE Computer Graphics and Applications*, 21(4):52–61, 2001.

[20] K. Kreeger and A. Kaufman. Interactive volume segmentation with the pavlov architecture. In *PVGS '99: Proceedings of the 1999 IEEE symposium on Parallel visualization and graphics*, pages 61–68, 1999.

[21] J. Krüger, P. Kipfer, P. Kondratieva, and R. Westermann. A particle system for interactive visualization of 3D flows. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):744–756, 2005.

[22] J. Krüger, J. Schneider, and R. Westermann. ClearView: An interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2006)*, 12(5), September-October 2006.

[23] J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[24] J. Krüger and R. Westermann. Efficient stipple rendering. In *Proceedings of IADIS Computer Graphics and Visualization*, 2007.

[25] R. Laramee, H. Hauser, H. Doleisch, F. Post, B. Vrolijk, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques.

[26] W. Li, L. Ritter, M. Agrawala, B. Curless, and D. Salesin. Interactive cutaway illustrations of complex 3d models. *ACM Trans. Graph.*, 26(3):31–40, 2007.

[27] A. Lu, C. Morris, D. Ebert, P. Rheingans, and C. Hansen. Non-photorealistic volume rendering using stippling techniques. In *IEEE Vis*, pages 211–218, 2002.

[28] E. Lum and K. Ma. Hardware-accelerated parallel non-photorealistic volume rendering. In *International Symposium on Non-photorealistic Rendering and Animation (NPAR)*, June 2002.

[29] S. Owada, F. Nielsen, M. Okabe, and T. Igarashi. Volumetric illustration: designing 3d models with internal textures. *ACM Trans. Graph.*, 23(3):322–328, 2004.

[30] A. Prior. "on-the-fly" voxelization for 6 degrees-of-freedom haptic virtual sculpting. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 263–270, 2006.

[31] T. Ropinski, J.-S. Prani, J. Roters, and K. H. Hinrichs. Internal labels as shape cues for medical illustration. In *Proceedings of the 12th International Fall Workshop on Vision, Modeling, and Visualization (VMV07)*, pages 203–212, 2007.

[32] I. Russell M. Taylor. Haptics for scientific visualization. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 174, 2005.

[33] A. R. Smith. Paint. Technical Memo 7, Computer Graphics Lab, New York Institute of Technology, July 1978.

[34] F.-Y. Tzeng, E. B. Lum, and K.-L. Ma. An intelligent system approach to higher-dimensional classification of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):273–284, 2005.

[35] I. Viola, E. Gröller, M. Hadwiger, K. Bhler, B. Preim, M. Sousa, D. Ebert, and D. Stredney. Illustrative visualization. IEEE Vis 2005, Tutorial #4.

[36] I. Viola, A. Kanitsar, and E. Gröller. Importance-driven volume rendering. In *IEEE Visualization 2004 (Conference Proceedings)*, pages 139–145, 2004.

[37] D. Weiskopf, K. Engel, and T. Ertl. Volume clipping via per-fragment operations in texture-based volume visualization. In *IEEE Visualization 2002 (Conference Proceedings)*, pages 93–100, 2002.