An Interactive Visualization and Navigation Tool for Medical Volume Data

O. Sommer, A. Dietz, R. Westermann and T. Ertl

Computer Graphics Group
Universität Erlangen-Nürnberg, Germany*

Abstract

Interactive direct volume rendering by hardware assisted 3D texture mapping has become a powerful visualization method in many different fields. However, to make this technique fully practicable convenient visualization options and data analysis tools have to be integrated. For example, direct rendering of semi-transparent volume objects with integrated display of lighted iso-surfaces is one important challenge especially in medical applications. Furthermore, explicit use of multi-dimensional image processing operations often helps to optimize the exploration of the available data sets. On the other hand, only if interactive frame rates can be guaranteed, such visualization tools will be accepted in medical planing and surgery simulation systems. In this paper we propose a volume visualization tool for large scale medical volume data which takes advantage of hardware assisted 3D texture interpolation and convolution operations. We demonstrate how to use the 3D texture mapping capability of high-end graphics workstations to display arbitrary iso-surfaces which can be directly illuminated to enhance the spatial relations between objects. Back-to-front 3D texture slicing is used to simultaneously display semi-transparent material densities. Using this approach similar image quality can be achieved as with conventional software-based ray-casting techniques, but the rendering process is accelerated to a considerable extent. In order to enable on-the-fly data analysis, first approaches using hardware assisted convolution operations have been integrated. An implementation of the proposed method based on the OpenInventor rendering toolkit is described offering interactive frame rates at high image quality including sophisticated user interactions.

1 Introduction and Related Work

A number of techniques have been developed to directly visualize 3D scalar fields on rectilinear Cartesian grids such as data sets from medical imaging modalities. In order to optimize the exploration process an important challenge is the integration of different rendering algorithms which allow simultaneously representing arbitrary material quantities such as soft tissue or sharp boundary regions.

Two widely used volume visualization methods which are often applied concurrently in medical applications are surface fitting [16] and direct volume rendering [9]. However, due to the large amount of voxels to be processed and geometric primitives which may be generated both techniques require time-intensive computations making it difficult to achieve interactive visualization.

Besides the use of dedicated volume rendering hardware [6, 10, 17] the most impressive frame rates at high image quality are definitely obtained by taking advantage of hardware assisted 3D texture interpolation on modern high-end graphics workstations [3]. On the other hand, it is well known that direct volume ray-casting offers the highest flexibility in terms of integrated feature enhancement, e.g. the reconstruction of iso-contours and the simulation of realistic lighting and shading effects. Additionally, there is no need to generate intermediate representations such as polygonal meshes if rendering of iso-contours is desired.

Another important trend is the development of graphics hardware that is able to support image processing algorithms like discrete convolution operations on 2D or 3D images. Of course, it is not yet clear how these operations can be seriously used in medical applications, but on the other hand, it is quite easy to simulate classical edge detection or noise removal algorithms which can then be applied interactively.

This paper proposes a visualization system that takes advantage of specialized graphics hardware to obtain flexible rendering and data analysis options at optimal speed. The following major issues are addressed by this tool:

- **Performance Tuning**: We make optimal use of hardware resources by exploiting 3D texture interpolation for the acceleration of the volume rendering process. A selection of classical image processing algorithms is implemented based on fast convolution operations.
- **Direct Volume Rendering**: We retain the full flexibility of volume ray-casting, in particular, we do not need an intermediate polygon representation for the visualization of surface features.
- **Contour Lighting**: We use realistic lighting effects to enhance the understanding of the spatial orientation of iso-surfaces and their relation to semi-transparent volume structures.
- **User Interface:** Since the whole visualization tool is embedded into the OpenInventor rendering toolkit it offers highest flexibility in terms of user manipulation and navigation. Arbitrary objects which are specified in the Inventor file format can be included.

The basic problem in volume visualization is the extraction and rendering of the information content of three dimensional structures contained in the volume in a way that allows also for accurate analysis. A general analytic description of the volume visualization process has been proposed by Krüger [11], who showed that all known direct volume rendering approaches can be understood as specializations of a transport theory model of the propagation of light in materials. Different approximations to the underlying transport equation have been developed [2, 5, 8, 18, 15] which differ substantially in the physical phenomena they account for and in the way the emerging numerical models are solved.

One fundamental difference lies in the order of the numerical evaluation of the arising integral equations. This can be done either in object-space order, where for each voxel the projection onto the image plane is computed and visualized, or in image-space order, where for each pixel those voxels are determined and rendered which contribute to the final pixel intensity. Image-space methods, i.e. ray-casting, are known to produce superior quality images for the cost of repeated re-sampling of the data. Although several

*Lehrstuhl für Graphische Datenverarbeitung (IMMD9), Universität Erlangen-Nürnberg, Am Weichselgarten 9, 91058 Erlangen, Germany, Email: `oesommer@informatik.uni-erlangen.de`

acceleration techniques have been proposed such as adapting the integration step size to the variance in the data [4, 14] or efficiently stepping through the volume [23] none of these methods allows for interactive image generation rates. Since voxel based projection methods [13, 21, 22] in general avoid the numerically complex re-sampling of the original domain, the expected frame rates are much higher than those of typical ray-casting techniques. Expressing the rotation of the volume objects by 2D shears and a final image warp and exploiting coherence and parallelism accelerates the process considerably [12].

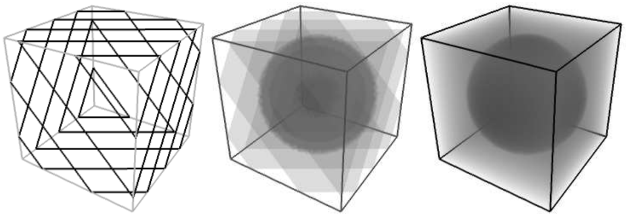

Figure 1: Volume rendering by 3D texture slicing.

Recently, the use of 3D texture mapping hardware [1], now also available in desktop graphics workstations and PCs, has become a powerful visualization option for direct volume rendering [3]. The rectilinear volume data is first converted to a 3D texture. Then, a number of planes perpendicular to the viewer's line of sight are clipped against the volume bounding box. The resulting polygons are projected onto the viewing plane using adequate blending operation to realize back-to-front or front-to-back compositing. Prior to the drawing procedure texture coordinates in parametric object space are assigned to each vertex of the clipped polygons. During rasterization the slice that is cut out of the 3D texture according to the texture coordinates is mapped onto the generated fragments (see Figure 1). Since this process is supported by specialized graphics hardware the time it consumes decreases considerably compared to a software realization. Thus, interactive frame rates can be achieved.

However, there is one major limitation of this method: It is not possible to integrate realistic lighting and shading effects which are known to enhance the perception of spatial relations between objects. On the other hand, even in medical applications where the expert often navigates through highly complex structures it is important to further improve the viewer's spatial sensation. Furthermore, it should not be ignored that also in medical data sets there are opaque boundary regions which are best visualized by means of lighted iso-surfaces. In Figure 2 an example is given which compares the visualization of an iso-surface of the skin of a 128^3 human aneurysm CTA-scan. On the left the rendering was performed with 3D texture mapping and appropriately adjusted transfer functions while on the right a volume ray-casting technique with integrated lighting was applied. Note, that although about 1200 slices were used to generate the left image it can easily be seen that the visual sensation of fine details and the spatial relation of arbitrary structures is considerably improved by the lighting and shading in the right image.

Of course, combination methods could be used which first extract a polygonal surface by means of a marching cubes type algorithm [16] which is then integrated with texture based volume rendering in a two-pass procedure. However, especially for large data sets, the memory and computation time requirements of even the most advanced iso-surface algorithm do not allow for an interactive modification of the iso values. Additionally, the amount of geometric primitives which may be generated can hardly be rendered in an acceptable amount of time.

Recently, first approaches for combining hardware accelerated volume rendering via 3D texture maps and volume lighting were presented. Van Geldern [19] stores the sum of precomputed ambient and reflected light components into the texture volume and performs standard 3D texture compositing as described above. The obvious drawback to this technique is the need for reloading the texture memory each time any of the lighting parameters (including rotation of object) changes. On the contrary Haubner et al. [7] precompute the voxel gradients and store a quantized representation of the orientation as the 3D texture map together with the volume density. Lighting is achieved by indexing into an appropriately replicated color table. However, smoothly shaded surfaces are difficult to achieve due to the limited quantization of the normal orientation and the intrinsic hardware interpolation problems.

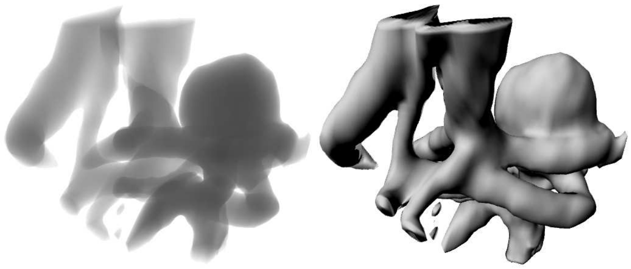

Figure 2: Texture slicing vs. direct contour lighting.

The basic idea of our approach lies in the utilization of hardware supported 3D texture interpolation to re-sample the data available on a rectilinear grid. In this way the numerically most critical part of traditional volume ray-casting techniques can be avoided. We propose a method to access the re-sampled texture values which allows detecting arbitrary iso-surfaces. Once their location in object space is determined various illumination effects can be rendered. In a second rendering pass we perform 3D texture slicing to integrate direct rendering of semi-transparent density volumes.

Another feature of the developed visualization system is the integration of specific data analysis algorithms. Up to now most of the available volume visualization tools based on 3D texture interpolation fail to offer additional techniques to analyze the data appropriately. Specific structures can only be enhanced or eliminated by modifying the transfer functions in a trial-and-error process. Actually, we integrated hardware supported convolution operations into the present approach. These operations can be applied on arbitrarily chosen clip planes and thus allow interactively performing edge detection or noise removal algorithms on selected portions of the data.

In the remaining sections we discuss the basic ideas of our approach. Once we have outlined the underlying concepts and algorithms we also describe the way they are realized within the Open-Inventor toolkit. This is important since no tool will be accepted in practical applications that does not provide an intuitiv user interface with convenient manipulation and navigation options. Finally, some examples are given to demonstrate the basic functionality.

2 Textured Sweep-Planes

So far, all known methods using 3D texture maps for the direct rendering of scalar volume data operate in a back-to-front or front-to-back order parallel to the image plane. For a number of equally spaced clip planes the planar cross-sections between these planes and the volume bounding box are computed and drawn into the framebuffer. Multiple polygons mapping onto the same screen region are blended according to the fragment opacity.

Apart from the advantages of this approach there is one obvious disadvantage which disables direct lighting calculations. Since texture mapping occurs during rasterization at the end of the graphics pipeline and incoming fragments are immediately accumulated with pixel values already drawn, it seems to be impossible to access interpolated texture samples. On the other hand, in order to determine smooth iso-surfaces and to compute realistic lighting effects we do need these values and their location in object space.

To retain the whole flexibility offered by direct volume ray-casting techniques while taking advantage of real-time 3D texture interpolation we propose a slightly different approach.

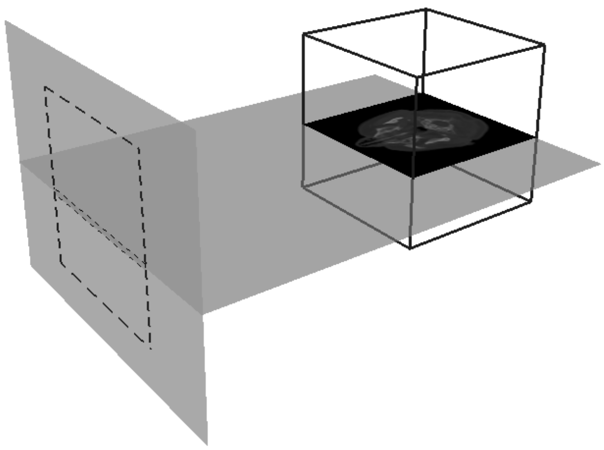

Figure 3: Texture slicing orthogonal to the viewing plane.

Instead of slicing the 3D texture map perpendicular to the viewer's line of sight we perform the same procedure but with cutting planes orthogonal to the viewing plane as outlined in Figure 3. The number of cross-sections we cut out of the texture is equal to the number of scanlines defined by the actual viewport definition.

This approach is very similar to the algorithm proposed in [6] which was integrated into a special purpose architecture for interactive volume rendering. However, it is interesting to see that the same algorithm can be efficiently implemented on general purpose graphics workstations and also low-cost PCs. Additionally, we will show that even if we only aim at rendering opaque iso-surfaces and semi-transparent density volumes without lighting effects we can save a lot of memory resources.

At a first glance slicing the texture orthogonal to the viewing plane does not allow us to gain anything. Since the cross-sections are parallel to the scanlines rendering with respect to the actual viewing transformation would just project each textured polygon onto one unique scanline. However, no meaningful images would be produced, because compositing along the viewing direction is not possible within one polygon.

But note that one cross-section generated in this way covers all interpolated data samples which would have been re-sampled by a standard volume ray-casting approach for the corresponding scanline. Thus, the entire interpolation procedure is performed by an efficient use of the available hardware resources. But it is still an open problem how to retrieve the texture samples in the cutting plane from the graphics pipeline in order to simulate an image space driven approach.

2.1 Adjusted Projection

We attack this problem by temporarily adjusting the global viewing transformation. Let us assume that the screen coordinate system is given by $\vec{X}$ and $\vec{Y}$ and that the viewing direction is given by $\vec{D}$. If the observer eyepoint is rotated 90 degrees around the $\vec{X}$-axis positioned in the center of the texture slice, then the former screen space $\vec{Y}$-axis and the viewing direction are interchanged in the new viewing system (see Figure 4). In terms of orthographic parallel projections, the projection onto the viewing plane is changed into a projection onto the slice plane.

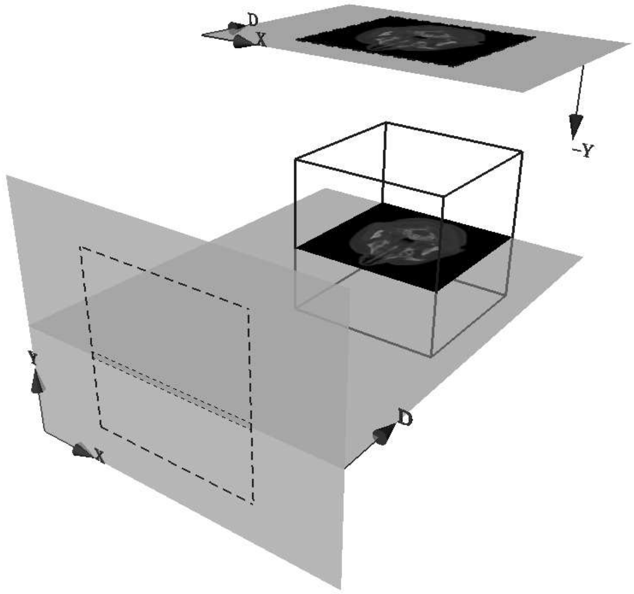

Figure 4: Changing the viewing system.

In the new viewing system we now perform a standard rendering of the polygon defined by intersecting the bounding box of the volume with the slice plane. Appropriately assigning texture coordinates results in the pixel intensities generated in the framebuffer during rasterization directly corresponding to the material values which would have been interpolated within one scanline. In order to access the texture samples we read the pixel values from the framebuffer into local main memory. All the information we need to process an arbitrary scanline is now available at the expense of one framebuffer operation. Of course, this operation is quite expensive, but it is still much faster than the interpolation of all data samples without hardware support.

The important point is that the volume re-sampling procedure, which accounts for about 60-70% of the total time of a standard ray-caster, is completely avoided. It is interchanged with the framebuffer operation. In contrast to the volume rendering pipeline as proposed by Levoy [14] (see Figure 5) two different data and computation flows are now possible. We should also mention that in general only a small fraction of the entire framebuffer has to be read. This will be discussed below.

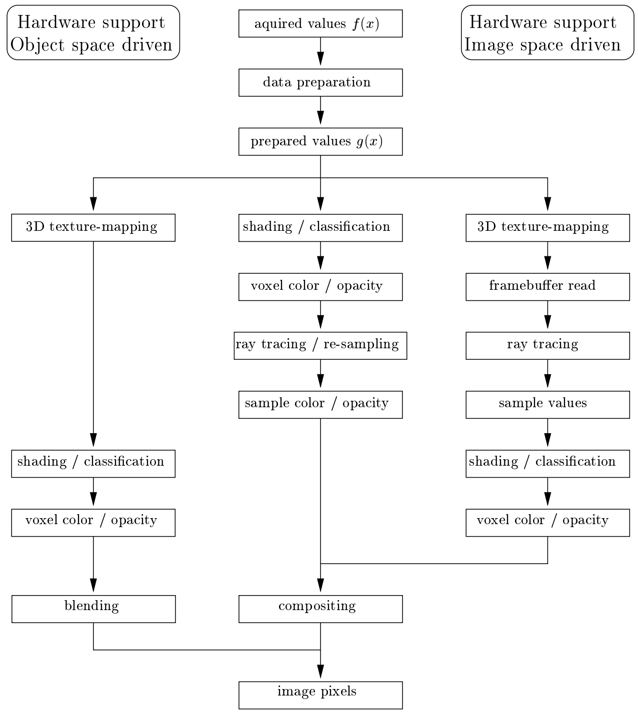

Figure 5: Overview of volume rendering pipelines.

2.2 Volume Integration

We are now ready to render the volume scanline by scanline. After reading the frambuffer all re-sampled data values which are necessary to process an entire scanline are available in main memory. Each column of the rectangular memory segment in which the pixel data is stored holds the material samples which would have been reconstructed along one ray of sight emanating from the viewing plane within the active scanline.

The amount of light impinging on the view plane at a certain position can be simulated by evaluating the volume rendering integral

$$I(t_0, t_1) = \int_{t_0}^{t_1} q(t) e^{-\int_{t_0}^{t} \alpha(s) ds} dt$$

along each ray. It sums up the contributions of the volume emission $q(t)$ along the ray, which is scaled by the optical depth according to the volume absorption $\alpha(s)$. Traditionally, the evaluation of the integral is performed using an Eulerian sum: the ray is split into segments of equal length over which the source term and the opacity are assumed to be constant. The continuous integral evaluates into a discrete sum over the segments along each ray:

$$I(t_0, t_1) \approx \sum_{k=0}^{n} q_k \alpha_k \prod_{i=0}^{k-1} (1 - \alpha_i). \quad (1)$$

Usually, the volume emission and absorption are assigned to each voxel before the integration is performed, or both values are obtained from a transfer function which maps the re-sampled material values to a RGB-color and a α-component.

In the latter case, as long as we assume an orthographic projection, the volume integration along a ray of sight collapses to the traversal of columns in the rectilinear 2D framebuffer segment in main memory. For each re-sampled texture value equation (1) is evaluated. This corresponds exactly to blending the texture slices in back-to-front order. Of course we have to spend more time in the software compositing, but on the other hand arbitrary acceleration techniques like α-termination or β-acceleration can be applied.

3 Iso-Contour Extraction

Maybe the most important drawback of volume projection techniques via 3D texture maps is that the visual appearance of iso-contours can hardly be enhanced by realistic illumination effects. Iso-contours, in general, can be polygonalized, by determining the cross-section between the surface and the volume cells [16]. Once the polygon model has been generated it can be rendered in a graphics pipeline. Arbitrary lighting and shading effects can be produced.

Basically, surface fitting techniques and direct volume rendering with 3D texture maps can be integrated quite easily. But neither the emerging memory overhead nor the time needed to reconstruct arbitrary surfaces is acceptable for large data sets. On the contrary, in volume ray-casting no intermediate surface representation is generated. The surface is directly visualized by successively testing whether the data samples along a ray meet certain criteria. The common approach which does not always yield accurate results is to traverse the ray until an isovalue is hit. Then the surface normal at this point is computed and arbitrary lighting or shading models can be evaluated. In [14] a different procedure was employed. Prior to re-sampling the material values are shaded and classified with respect to an isovalue and the local greyscale gradient. Different material types can be enhanced or suppressed in this way.

Our present approach offers the whole flexibility volume ray-casting does. Classification and shading as proposed in [14] can be applied to the acquired or prepared scalar values, but also the visualization of iso-surfaces by isovalue testing can be integrated straight forwardly. While the sweep-plane is traversed each data value is compared to the specified isovalue. If a hit with the iso-contour is determined the normal is calculated and the contour at the actual location is shaded.

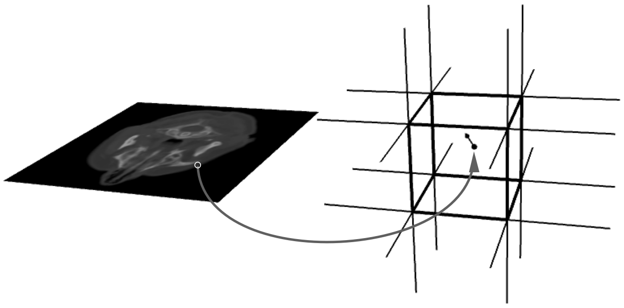

Figure 6: Gradient approximation on original data.

Very fast results can be achieved by directly computing the normal from the values already in the sweep-plane buffer. On the other hand, the generated images show typical block artifacts which emerge from the discretization of the buffer, and additionally two further sweep-planes have to be stored in memory in order to access the top and bottom neighbors.

Instead we transform the location of each data sample which belongs to a contour back into the original voxel array (see Figure 6). Then, the normal at this point is interpolated from the normals at the eight nearest neighbors. This yields smooth results and is also less memory intensive. Note that the normals we approximate at the discrete grid points are temporarily stored until a complete scanline is processed, thus avoiding multiple approximations of the same normal.

4 Convolution Operations

Particularly in image processing the application of discrete convolution operations on the available pixel values is a commonly used data analysis option. Depending on the desired result different kinds of convolution kernels are applied which offer distinct choices to enhance or suppress specific features. For example, difference operators to detect edges or simple average operators to perform noise reduction are often applied separately or one by another to improve the overall understanding of the data. However, although fast software realizations exist which efficiently perform discrete convolution operations on 2D images, in general their use is limited to non-realtime applications due to the numerical complexity of the filtering process. On the other hand, special purpose hardware exists, now also available on modern graphics workstations, which enables the convolution of large scale images with arbitrary filter kernels interactively.

In particular, the newer SGI machines provide extensions which allow hardware supported convolution of multi-channel pixel data. While the data is sent through the rendering pipeline, e.g., drawing from main memory into the framebuffer, convolution of the data takes place before it gets written to the framebuffer. Thus, the convolution of arbitrary slices from the volume data can be performed quite easily. In the present work the key idea was to extent this functionality to arbitrary clip planes passing through the volume.

Whenever a clip plane is activated and convolution is enabled the pixel values within the clip plane are convolved with a filter kernel that can be chosen from a pre-defined toolbox. Several kernels have been implemented, e.g., sobel, median, laplacian, Maar-Hildreth, blurr etc. In this way it is possible to detect edges, to sharpen the clip plane image or to suppress noise within. Of course, it is not obvious whether it really makes sense to perform the convolution on arbitrarily sliced images from the data. Since the data values within the clip plane are interpolated from the discrete voxel values, filter operations can lead to results which may be in some sense misleading. However, since the operations are completely interactive their application can often help to enhance the overall understanding (see Figure 7).

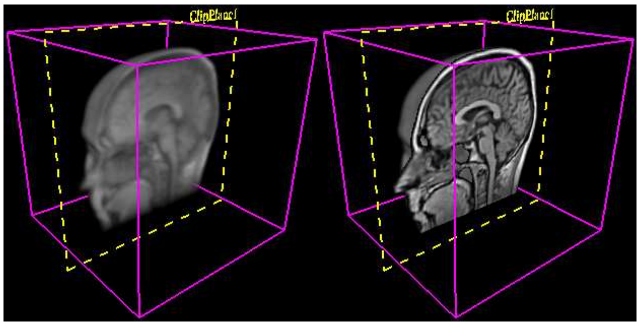

Figure 7: Clip plane without convolution and with enabled high-pass filtering.

In order to perform the convolution the clip plane extent has to be retrieved. This is accomplished in three steps. First, we request the clip plane equation from the Open-GL state. Then, the viewing transformation is adjusted such that we are looking orthogonal to the clip plane. Finally, the plane is clipped against the volume bounding box and the obtained polygon is textured with the 3D volume and projected onto the viewport. Now we have all the interpolated voxel values within the clip plane in the framebuffer. Reading the framebuffer, enabling convolution and drawing the image back into the framebuffer leads to the desired result.

5 Integration in OpenInventor

The presented algorithm was implemented using OpenInventor, an object oriented graphics toolkit built on top of OpenGL, which has become a defacto standard for interactive modeling, rendering and manipulation of 3D scenes [20]. One part of the work presented here is the complete integration of texture mapping based volume rendering into the OpenInventor framework in order to obtain the whole flexibility and functionality offered by the toolkit. By introducing a new class the volume renderer is represented as a separate object within the hierarchical structure of the scene graph. This allows convenient application of built-in manipulators, sensors, editors and other predefined classes, methods and features (light sources, anti-aliasing, stereo mode, perspective/parallel rendering, fly, walk, trackball).

In particular, the design of the new volume object takes advantage of the OpenInventor structuring mechanism of node kits which organizes the newly implemented nodes as separately managed subgraphs (see Figure 8). Our `SoVolumeKit` is subclassed from `SoBaseKit` and contains clip planes with a geometric representation which can be accessed from the OpenInventor standard manipulators and a `SoVolume::SoShape` node. The internal structure is designed to support the handling of multiple volumes. During rendering an object of type `SoGLRenderAction` traverses the scene graph and asks all objects to render themselves by calling their local `GLRender` method. Within this method of `SoVolume` all object specific OpenGL calls are performed.

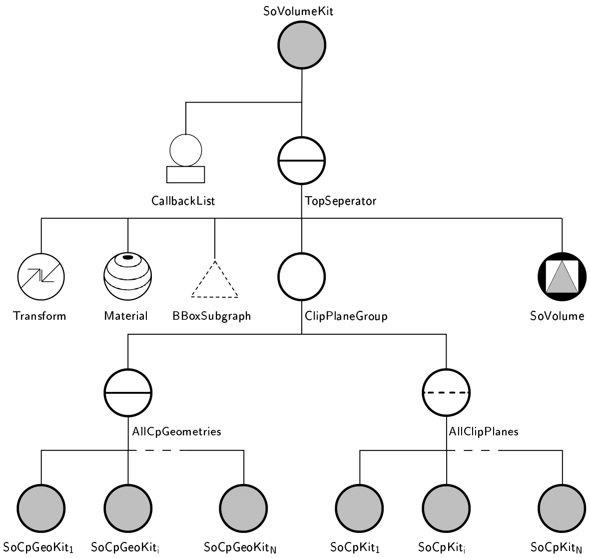

Figure 8: Subgraph structure of SoVolumeKit: It contains a geometric representation of the bounding box, the subgraph of up to six clip planes, and the SoVolume node which is responsible for the render action.

We can arbitrarily switch between traditional back-to-front 3D texture projection and our new technique of ray-casting through textured sweep-planes implementing realistic lighting effects. If the expensive lighting mode is enabled we automatically switch back to 3D texture projection while the camera position is modified returning to lighting mode after a certain time-out. For the lighting calculation during ray traversal we use the lighting parameters (position, direction, etc.) of the interactively placed OpenInventor lights. These and other parts of the traversal state like color, clipping planes, transformation matrices etc. can be accessed from the `SoState` object delivered by the `SoGLRenderAction` and by direct OpenGL calls.

Since we have to cut multiple slices out of the volume orthogonal to the active viewport we need a separate render area in which the textured polygons can be drawn from the adjusted view point. Choosing the back buffer has several disadvantages. First, objects already drawn into the back buffer would be destroyed. Second, the actual render area could be too small or could be overlayed by other windows, which makes calls to `glDrawPixels` fail. Therefore, we decided to use the `SGIX_pbuffer` extension which provides a part of the physical framebuffer which can be directly accessed by the graphics hardware, but which is not displayed on the screen. Furthermore, p-buffers can be locked exclusively by a certain application against other access.

In order to minimize the amount of p-buffer needed, only the bounding region that is covered by the volume object is taken into account during slicing. First, we determine the bounding box extent, and then we slice that extent scanline by scanline. Arbitrarily translated objects can be handled in this way.

Our approach easily extends to perspective projections. All which has to be done is to modify the current projection matrix in such a way that an additional rotation of 90 degrees around the x-axis is applied after the perspective projection. Also, the textured sweep-planes must be inclined according to the chosen perspective for each scanline. However, this procedure does not produce correct results on all types of graphic hardware since perspective correction during 3D texture mapping on the slicing planes is not always supported.

6 Results

The results were computed on a Silicon Graphics Indigo^2 Maximum Impact with a 250 Mhz R4400 processor, 128 MB memory and TRAM option. Our experiments were run on different data sets to demonstrate the impact of the data resolution to certain parts of our algorithm and also to show the functionality of the implemented visualization tool. Two data sets were used: a human head MRI-Scan with 128^3 voxels and a $256^2 \times 128$ CTA-Scan of an aneurysm also from a human head.

Table 1 shows accurate timings for all distinct parts of the algorithm. Basically, we distinguished between four different tasks: (1) All operations within the graphics pipeline (**GrPipe**) including framebuffer access. (2) Volume re-sampling by tri-linear interpolation (**Sample**). (3) Mapping via the transfer function and compositing (**Comp**). (4) Gradient calculations in the iso-contour reconstruction (**Grad**). Finally, the overall times are given (**All**).

Table 1: Timings for 128^3 human head data set in seconds. One texture map was used. Image resolution was 400×400.

Mode	Density		Iso	
	HW	SW	HW	SW
GrPipe	1.01	—	1.01	—
Sample	—	42.4	—	15.1
Comp	32.2	33.4	—	—
Grad	—	—	1.8	1.9
All	33.21	75.8	2.71	17.0

First, the front-to-back compositing of material color and opacity values was applied (**Density**). This is equivalent to the standard volume rendering technique using 3D texture maps. Second, we focused on a specific contour surface belonging to a given iso value (**Iso**). Within each column we compared the results of the hardware (**HW**) assisted approach with its software (**SW**) pendant. Note that re-sampling is almost negligible in our approach, and that we do not need to process the whole volume if the iso value is changed. This is done in shell-rendering, where those voxel which belong to a certain iso-surface have to be classified in advance.

We can see that the framebuffer operations indeed dominate the overall times during surface rendering. For each scanline a $400 \times 128 \cdot \lfloor\sqrt{3}\rfloor$ framebuffer array was read and traversed in main memory. This corresponds to a step size of one voxel size, which has also been chosen in the reference method. No time for volume re-sampling is used which is in fact the dominant time in a standard ray-casting method. We see that gradient calculations significantly slow down the overall times. This overhead can be avoided if a second texture is stored in which the gradients are coded as RGB-values. From each slice which is read out of the framebuffer we can now determine the hit with the surface and also the gradient at this position. However, this approach doubles the memory requirements, but on the other hand, the time which is needed to perform the gradient calculations is almost negligible in this case.

We should mention that in order to optimize our algorithm we completely avoid the software compositing in the actual implementation. If the iso-contour is determined and written to the framebuffer we store the z-buffer values and perform a second rendering path using the traditional back-to-front 3D texture projection. Consequently, in the first column of Table 1 the overall time decreases to approximately 0.3 seconds. This method was used to generate the images in the color page below.

Our second experiment was run on the 256^2x128 aneurysm CT-scan. Due to the limited texture memory of our target architecture we first had to split the volume into distinct blocks. In order to obtain a texture slice belonging to a certain scanline all bricks have to be reloaded into texture memory. This slows down the rendering process considerably. The overall time increased by about a factor of 4.

The last two images in the color plate below show additional examples out of an interactive session with the presented visualization tool. The time needed to render the head data set was approximately 3.0 seconds. Rendering the bricked aneurysm took about 8.7 seconds.

7 Discussion

We have extended the volume rendering techniques via 3D texture maps by combining hardware assisted texture interpolation with realistic illumination effects for shaded iso-surfaces. The whole flexibility of front-to-back volume ray-casting is maintained in this way, i.e. simultaneous visualization of soft tissue and solid iso-contours can be achieved. Of course, we can not compete with interactive frame rates as achieved by 3D texture slicing techniques, but on the other hand, the integration of iso-surface reconstruction and hardware assisted convolution operations allows almost interactively analyzing large data sets. In contrast to other approaches which use extended color maps and precomputed voxel gradients to simulate direct volume lighting, we save memory, get smooth contours and avoid the numerical re-calculations during movements. One drawback is the handling of multiple texture maps. Successively reloading texture maps for each scanline processing slows down the algorithm.

References

[1] K. Akeley. RealityEngine Graphics. *Computer Graphics, Proc. SIGGRAPH '93*, 27(4):109–116, July 1993.

[2] J. Blinn. Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *Computer Graphics, Proc. SIGGRAPH '82*, 16(3):21–30, July 1982.

[3] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In A. Kaufman and W. Krüger, editors, *1994 Symposium on Volume Visualization*, pages 91–98. ACM SIGGRAPH, 1994.

[4] J. Danskin and P. Hanrahan. Fast Algorithms for Volume Ray Tracing. In *Transactions 1992 workshop on Volume Visualization*, pages 91–98, 1992.

[5] B. Drebin, L. Carpenter, and P. Hanrahan. Volume Rendering. *Computer Graphics*, 22(4):65–74, August 1988.

[6] T. Guenter. Virim: A massively parallel processor for real-time volume visualization in medicine. In *Proceedings of 9th EG Hardware Workshop*, pages 103–108. Eurographics Assiciation, Addison-Wesley, 1994.

[7] Haubner, M. and Krapichler, Ch. and Lösch, A. and Englmeier, K.-H. and van Eimeren W. Virtual Reality in Medicine - Computer Graphics and Interaction Techiques. *IEEE Transactions on Information Technology in Biomedicine*, 1996.

[8] J. T. Kajiya and B. P. Von Herzen. Ray Tracing Volume Densities. *Computer Graphics*, 18(3):165–174, July 1984.

[9] A. Kaufman. *Introduction to Volume Visualization*. IEEE Computer Society Press, 1991.

[10] Knittel, G and Straßer, W. A Compact Volume Rendering Accelerator. In Kaufman, A. and Krüger, W., editor, *1994 Symposium on Volume Visualization*, pages 67–74. ACM SIGGRAPH, 1994.

[11] Krüger, W. The Application of Transport Theory to the Visualization of 3-D Scalar Data Fields. In A. Kaufman, editor, *Visualization 90*, pages 273–280, Los Alamitos, CA, 1990. IEEE, IEEE Computer Society Press.

[12] P. Lacroute and M Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform. *Computer Graphics, Proc. SIGGRAPH '94*, 28(4):451–458, 1994.

[13] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. *Computer Graphics*, 25(4):285–288, July 1991.

[14] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, March 1988.

[15] M. Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[16] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[17] H. Pfister and A. Kaufman. Cube-4 - A Scalable Architecture for Real-Time Volume Rendering. In R. Crawfis and Ch. Hansen, editors, *1996 Symposium on Volume Visualization*, pages 47–54. ACM SIGGRAPH, 1996.

[18] P.A. Sabella. A Rendering Algorithm for Visualizing 3D Scalar Fields. *Computer Graphics*, 22(4):51–58, August 1988.

[19] A. Van Geldern and K. Kwansik. Direct Volume Rendering with Shading via Three-Dimensional Textures. In R. Crawfis and Ch. Hansen, editors, *1996 Symposium on Volume Visualization*, pages 23–30. ACM SIGGRAPH, 1996.

[20] J. Werneke. *The Inventor Mentor, Programming Object-Oriented 3D Graphics with OpenInventor*. Addison-Wesley, release 2 edition, 1994.

[21] L. Westover. Footprint Evaluation for Volume Rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[22] J. Wilhelms and A. Van Geldern. A Coherent Projection Approach for Direct Volume Rendering. *Computer Graphics*, 25(4):275–284, July 1991.

[23] R. Yagel and Z. Shi. Accelerating Volume Animation by Space Leaping. In G.M. Nielson and Bergeron D., editors, *Visualization 93*, pages 62–69, Los Alamitos, CA, 1993. IEEE, IEEE Computer Society Press.

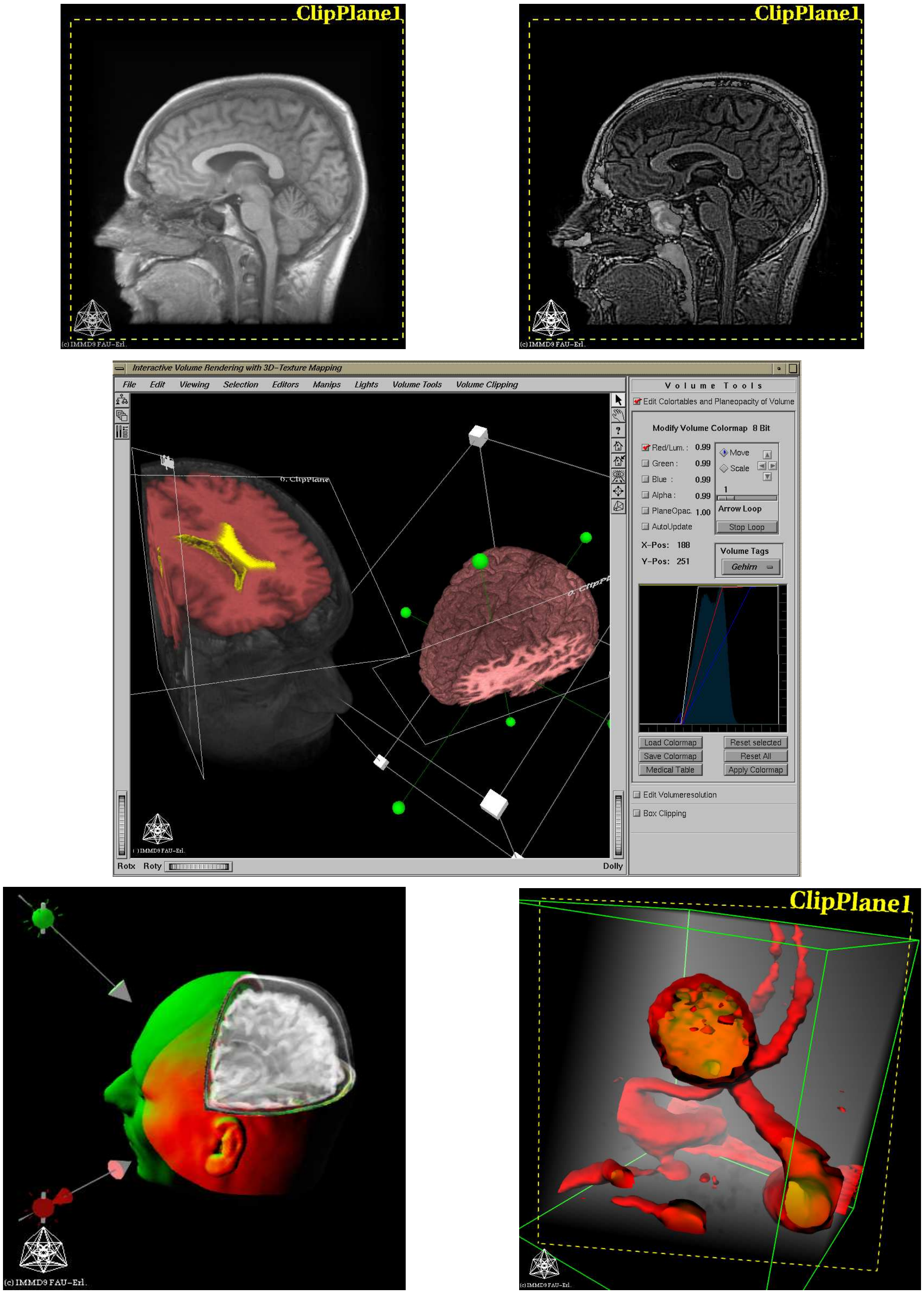

Figure 9: Example images from an OpenInventor session. They show the filtering of a clip plane extent with a high-pass filter, multiple clip planes, lighted iso-surfaces and integrated display of semi-transparent material and opaque surfaces.