Interactive Volume Exploration on the StudyDesk

Werner Wohlfahrter
Vienna University of Technology
Favoritenstrasse 9-11/5
A-1040 Vienna, Austria
ww@cg.tuwien.ac.at

L. Miguel Encarnação
Fraunhofer Center for Research in Computer Graphics (CRCG)
321 South Main Street, Providence, RI 09203, USA
mencarna@crcg.edu

Dieter Schmalstieg
Vienna University of Technology
Favoritenstrasse 9-11/5
A-1040 Vienna, Austria
dieter@cg.tuwien.ac.at

Abstract

We present the combination of an interaction–rich virtual environment and a volume-rendering library which enables to work in an easier and more natural way on volume data. This paper describes how we combined these packages and points out the advantages of such a combination. We also give a short survey of the state of the art in volume rendering libraries and we explain why we selected SGI Volumizer as the volume rendering library for our volume VR system.

Keywords

3D interaction, virtual reality, augmented reality, user interface, volume data

1 Introduction

Volume data is becoming increasingly important in the scientific and medical field because of the use of MRI, CT and ultrasound. Volume data are usually very large and difficult to handle but need to be analyzed in an efficient way. It is therefore important to think about easier and more natural ways to interact with volume data. Our approach is a combination of an interaction–rich VR system (StudyDesk) and a volume-rendering system (SGI's OpenGL Volumizer). We selected SGI OpenGL Volumizer as the most suitable volume rendering library for our interaction research, over VolPack, 3DDataMaster, Amira, TeleInVivo, VoxelView and VTK. Our solution comprises a powerful interaction system for exploring volume data that allows the user to drag, scale, or cut the volume in a very natural way using two–handed interaction. It also provides a method for "freezing" and "unfreezing" cutting planes into different displays by using lookup tables, and the ability to extract an arbitrary slice out of the volume data.

2 Related Work

The following sections give you a brief introduction to StudyDesk, the VR interaction framework we use, and Mirror Tools, which are an add–on to StudyDesk.

2.1 StudyDesk

Studierstube [7, 22, 18] is an augmented reality framework based on OpenInventor, which supports two–handed interaction using a tracked personal interaction panel (PIP) in the one hand and a tracked pen in the other. The PIP is a see–through plastic palette onto which virtual menus and controls are mapped. The pen, also designed as a see–through device, is used to point to certain objects. A user works like a painter who holds a painters palette in his non–dominant hand and a brush in his dominant hand (Figure 1). Similar two–handed interaction has proven useful in medical applications such as surgical planning [8]. The VR interaction environment that uses Studierstube in combination with the Virtual Table described in [21] is called StudyDesk which was employed for this work.

2.2 Mirror Tools

Bimber et al. [1, 2, 16] extended the capabilities of the PIP by adding a semi–reflective, semi–transparent foil onto it using it not only as a transflective pad, but also as a reflective pad. This means that the volume projection is calculated by using an

eyepoint reflected over the PIP plane. The volume therefore appears distorted on the table but correct on the reflective pad. This technique provides two advantages:

- The tracked 3D space in front of the table can be enlarged by mirroring the eyepoint.
- Normally only one person can be tracked, and the others must wear untracked glasses. The projection is calculated only for the tracked person to whom it appears perfect, but to the others the projection is distorted. Bimber et al. showed that this distortion is small when using the mirror setup.

3 Volume Rendering

As an introduction to 3.4 studied volume rendering packages, we give a short overview of the state of the art in volume rendering, in which we compare different libraries.

Kaufman's [10] survey of Volume Rendering divides it into two parts: surface rendering techniques and direct volume rendering techniques.

3.1 Surface Rendering Techniques

Surface rendering techniques approximate a surface contained within volumetric data using primitives, which can be rendered using conventional graphics accelerator hardware. A surface can be defined by applying a binary segmentation function, $B(v)$, to the volumetric data. $B(v)$ evaluates to 1 if the value v is considered part of the object, and evaluates to 0 if the value v is part of the background. Thus the surface is the region where $B(v)$ changes from 0 to 1. With continuous interpolation functions, a surface, known as an iso-valued surface or an isosurface, may be defined by a single value.
First proposed by Herman and Liu in 1979, the *Opaque Cubes* method (also known as Cuberille) was one of the first widely used visualizing methods for volume data, and it simply visualizes the cells that are part of the isosurface.

Schroeder, Martin and Lorensen [23] describe *Color Mapping* as a visualization technique which maps scalar data into colors and displays them. The scalar mapping is implemented by indexing scalar values into a color lookup table. They also discuss *Contour Tracking* in which the eye often separates similarly colored areas into distinct regions, effectively constructing the boundary between these regions. These boundaries correspond to contour lines of constant scalar value. Examples of 2D contour displays include weather maps shown with lines of constant temperature (isotherms), or topological maps drawn with lines of constant elevation.

The most important surface rendering algorithm is *Marching Squares*, used for 2D data, and *Marching Cubes* [15], used to create isosurfaces out of 3D data. Given a volume dataset and a threshold value, the isosurface passing through the points of the volume dataset having this value can be reconstructed by using the *Marching Tetrahedra* [4] algorithm, which is a per cell approach similar to the marching cubes algorithm. Another marching cubes–related algorithm is called Dividing Cubes. In that case every intersected voxel is subdivided until the divided voxel is as large as a pixel of the final image or smaller. This voxel will be rendered as a point.

3.2 Direct Volume Rendering Techniques

Direct volume rendering techniques are divided into *Image Order* and *Object Order* direct volume rendering techniques.
Image order is also known as ray–casting, backward–mapping, and pixel–space projection. To render the image, all pixels of the image are calculated by looking what voxels contribute to it. This is done by casting a "virtual ray" and evaluating the voxels that are "hit".
Object order is also known as forward–mapping and voxel–space projection. To render the image, all volume elements are traversed and projected into image space. In other words, each voxel is accessed to calculate the contribution it makes to the image.

Slicing is a volume visualization method that has been used in medicine for many years. Every CT or MRI scan is a set of image "slices" of a human. This method is important for our work and will be described in greater detail later.

A more recent method, *Splatting* or *Footprint evaluation* (perspective variant), completes a front–to–back object–order traversal throughout the dataset. It calculates (per plane or slice) what the contribution of the voxels in that plane are, and thus what the contribution of the plane itself is to the image. This contribution is attenuated by a often pre–calculated kernel (usually a Gaussian Filter), and finally projected into the image plane.

Lacrute and Levoy [13, 14] introduced *Shear Warp Factorization*. Their method is based on a factorization of the viewing matrix into a 3D shear parallel to the slices of the volume data, a projection to

form a distorted intermediate image, and a 2D warp to produce the final image.

3.3 Applied Volume Rendering Criteria

A crucial part of our work was to find the most suitable volume rendering package. Our environment and the targeted functionality to analyze volume data define certain meeting points that a volume rendering library should provide.
Because Studierstube is based on OpenInventor, we prefer a volume rendering library that is based on OpenGL, making it easy to integrate and advantageous in terms of performance. To feel comfortable working with such system, realtime rendering is necessary. StudyDesk is driven by an SGI workstation that defines the hardware platform. Stereo vision is a supported feature of Studierstube. To get correct stereoscopic projection of the volume data, perspective rendering is needed. Exploring the inside of a volume is a major requirement for analyzing volume data. This can be done by cutting the volume or storing textures of an arbitrary plane. Access to the underlying data is very important to retrieve and control these textures.

3.4 Studied Volume Rendering Packages

VolPack [17] is a portable software library for volume rendering, developed by the University of Stanford. Since the library does not use any specialized hardware, it is portable to most platforms and still achieves very fast rendering times. VolPack is based on the Shear–Warp algorithm. The library does currently not support perspective projections and clipping planes.

3D Data Master [25] is part of the 3D DataSuite from TGS. 3D Data Master, an extension to OpenInventor, can deal with different types of meshes to represent the data. These meshes are divided into *Surface Meshes*, to store 2D data, and *Volume Meshes*, to store 3D data. Surface and volume meshes can either be *structured grids* or *unstructured meshes*. 3D DataMaster uses color mapping, marching cubes and marching tetrahedra to render the data. This software package was a quick development on top of OpenInventor. TGS no longer recommends using it as a volume rendering solution, and instead recommends their new product Amira.

Amira [26] is the new strategic volume rendering software offered by TGS. Unfortunately, Amira, which is based on the OpenInventor graphics toolkit, does not include an API (TGS is working on that). Amira uses marching cubes to generate isosurfaces but does not exactly specify their algorithm for direct volume rendering and data representation. The package relies on fast hardware–accelerated OpenGL 3D graphics. TGS strongly recommends using hardware texture mapping, since many visualization tools in Amira rely on it and some features require it. Amira might be a very good volume–rendering application but is not yet programmable and therefore not usable in our case.

TeleInVivo [27] is a commercially available volume–visualization application developed at Fraunhofer CRCG. It is based on a volume rendering framework developed in house. TeleInVivo uses the parallel projection Shear–Warp algorithm for rendering. It is currently implemented as a pure software renderer without using the OpenGL library because the combination of Shear Warp and OpenGL is still a problem. TeleInVivo has DICOM–3 read capabilities and can handle any imaging modality stored in that form. In addition, it has built–in capabilities for freehand acquisition of 3D ultrasound using the MircoScribe arm from Immersion Corp. TeleInVivo also renders multiply segmented and labeled volumes (i.e. digital atlases). A key feature is the collaborative mode that allows remotely situated personnal to share views of a volumetric dataset. TeleInVivo is a pure software renderer, which makes it hardware independent on the one hand, but relatively slow on the other. The Shear Warp algorithm is implemented for parallel projection only.

VoxelView [28] was designed and built by VitalImages as a visualization software package especially for the medical imaging market. Unfortunately, there is no information available about algorithms or internal data structures used in this package. VoxelView uses shared memory and semaphore to share the data with other programs. It is possible to manipulate the data over the shared memory and display it in VoxelView, but it is not possible to use their volume–rendering engine only. VoxelView uses SGI as its strategic platform. It supports DICOM, Analyze and TIFF, and it can load datasets like GE, Siemens and Toshiba. Perspective projection is used to render the volume data. Isosurfaces are supported as well. VoxelView provides a functionality called tracing contours, which allows users to trace a contour on each slice and get the newly generated geometry displayed in combination with the volume data. VoxelView is well implemented and offers various functions, but unfortunately it is not freely available and has no API. VitalImages re-

cently released a volume–rendering software package followed–up called Vitrea.

The Visualization ToolKit (VTK) [23, 11] is an open source, freely available software system for 3D computer graphics, image processing, and visualization. VTK includes a C++ class library (with more than 500 classes), and several interpreted interface layers including Tcl/Tk, Java, and Python. VTK has been implemented on nearly every Unix–based platform and PC (Windows NT and Windows95). The graphics model in VTK is at a higher level of abstraction than rendering libraries like OpenGL or PEX. In VTK, applications can be written directly in C++, Tcl, Java, or Python. In fact, using the interpreted languages Tcl or Python with Tk (and even Java with its GUI class libraries) it is possible to build useful applications fast. VTK is a very well developed volume–rendering library. It is easy to use and powerful, but slow because it uses ray casting as its volume rendering technique. As such, it is unreasonable to use this library for our work without having a Mitsubishi VolumePro [19] board at hand.

OpenGL Volumizer [24] is the volume rendering library from Silicon Graphics. This volume rendering library will be described in more detail because this is the library of choice for the presented application system.
It is built, as the first part of the name suggests, on top of OpenGL, and it works very well with OpenInventor. OpenGL Volumizer uses a technique similar to ray casting, called volume slicing, to leverage the texture–mapping hardware that many workstations now have. Ray casting can be performed in ray–order or sampling–surface order using planes or spherical surfaces parallel to the line of sight. Volume slicing and ray casting are equivalent in the following ways:

- Ray casting under orthographic projection is equivalent to taking a series of slices along planes parallel to the viewport and compositing them.
- Ray casting under perspective projection is equivalent to sampling along a series of concentric spherical shells centered at the eye.

The result is volume rendering according to texture mapping, as shown in Figure 2.

Usually in ray casting, each point on a ray that is projected from the eye position through the volume is processed sequentially by the CPU, therefore slowing the process down. In Volumizer, all points on a plane orthogonal to the line of sight are computed sequentially in the texture–mapping hardware (Figure 4).

Some image databases contain more voxel data than can be stored in a machine's texture mapping hardware. To take advantage of hardware acceleration, voxel data is broken up into subsections called bricks. A brick is a subset of voxel data that can fit into a machine's texture–mapping hardware. Bricks are regular hexahedra (boxes) with non–zero width, height, and depth. Displaying a volume requires paging the appropriate brick data into texture memory.

For further information, please consult the OpenGL Volumizer Programmer's Guide [5].

4 Implementation

The best way to combine OpenGL Volumizer and StudyDesk is to encapsulate Volumizer into OpenInventor nodes, which then become part of the used OpenInventor scene graph (Figure 3). Since Volumizer acts like normal OpenInventor nodes, StudyDesk does not have to change in any way. Figure 5 shows the relationship between the software pieces used.
A status object stores all Volumizer and application–related states. Every state change will also change the status object and therefore affect the application's behavior. A state change is provoked by the following functions.

- **Loading volume data**
  The application can either read a 3D tiff or a number of 2D raw format files, which is important because most medical devices store their volume data in that format. The volume data are divided and stored as bricks whose size depends on the hardware used. One brick stores exactly the amount of data that can be served by the texture memory. The collection of bricks is stored in the main memory and select bricks are downloaded into the texture memory on demand. Having a large texture memory increases the size of the bricks and decreases the download effort from the main memory to the texture memory.
- **Scaling the volume**
  As described above, OpenGL Volumizer uses the Volume Slicing technique to render the volume data, and the texture mapped onto the slices is calculated from the voxel data. Enlarging the volume by moving a sliderbar that is mapped onto the PIP increases both the

number of slices and the size of each slice, which provokes more texture calculation and slows down the application.

- **Lookup table**
  Using Volumizer's lookup table functionality, which controls the opacity and colors of the volume, enables us to display the data that lies in a specified range (compare Figures 11 and 12). We use a linear lookup table because it is hardware optimized and goes from 0.0 to 1.0, starting at $mean - tolerance/2$ and finishing at $mean + tolerance/2$. Mean and tolerance can be set using sliders on the PIP similar to those used for scaling the volume. The lookup table is activated by pressing the activate/deactivate lookup table button on the PIP (Figure 10).

- **Dragging the volume**
  One way we analyze an object in the real world is by grabbing it, rotating it (if possible) and taking a closer look to get more details about the interesting parts. We aimed for having the same ability in our environment to explore the data in a similar way. Dragging the volume is done by copying the pen's tracking position to a transform node that is located between the volume dragger node and the volume (Figures 3, 8, 9).

- **Cutting the volume**
  The PIP is a tracked plastic pad that looks like a clipboard. By flipping it over, the PIP functionality changes from *display mode* to *cut mode*, which means that the menus mapped onto it disappear and the PIP works like a cutting plane when sweeping it through the volume. While cutting the volume with the PIP held in the nondominant hand, it is possible to simultaneously move the volume using the pen in the dominant hand. There is anecdotal evidence that it is easier to find the interesting areas in the volume using both hands because one hand is usually trained for different tasks. The dominant hand is usually used for precise work where as the non dominant hand mostly takes the supporting rule [8]. The cut is technically done by using an OpenInventor clip plane node, which gets the tracker coordinates and orientation of the PIP. The clip node is placed before the volume dragger node, that is used to drag the volume to provide two–handed interaction while cutting the volume described above (Figures 3, 7).

- **Freezing cutting planes**
  To "freeze" certain cutting planes, an additional mode has to be set before flipping over the plane. If the perfect cutting position is reached, clicking the button on the pen freezes the current cutting plane (Compare Figures 12, 13). This is a very natural way of defining a volume of interest. Internally a copy of the OpenInventor clipping plane is stored after the volume–Dragger node and before the volume node. By doing so, the clipping plane is fixed to the volume, regardless of whether the volume is dragged or scaled, until the plane is unfrozen (Figure 3).

- **Unfreezing cutting planes**
  To "unfreeze" a cutting plane, the PIP must be flipped back over into the *display mode*. A simple click of the pen button unfreezes the last cutting plane, which technically means that the last stored OpenInventor clip node is deleted from the scene graph (Figure 3).

- **Mirroring**
  To activate the mirror functionality, described in 2.2, the PIP must be turned over. The application is in clipping (transflective) mode if the user looks through it to the table. It is in mirror (reflective) mode if the user looks at the PIP and seeing a correct reflection of the volume projected on the table (because the PIP is working like a real mirror) (Figure 14).

- **Extracting arbitrary slices**
  Extracting an arbitrary slice of data out of the volume is important for medical or geological applications. Arbitrary slices are generated by extracting data off the volume. The complexity involved in getting these slices out of the volume depends on the texture–mapping hardware used. With three–dimensional texture–mapping hardware, the slicing plane can be clipped using Volumizer's clip function, and the resulting polygons can be drawn with texturing enabled. Without three–dimensional texture–mapping hardware, computing a tri–linearly filtered oblique slice through a volume is more challenging (see [5] for more details). Though it is complex it can be done using OpenGL Volumizer's Multi–Planar Reconstructions (MPR) which calculate the slice and store the data in a separate data structure.

5 Results

The two–handed interaction provided by StudyDesk is well suited for handling volume data and provides a very easy and natural way of manipulating it. For example, we found that defining the best cutting position in the volume is much easier when using both hands then using multiple sliders.

The way we get arbitrary slices out of the volume is fast and easy and could, therefore, be of interest to researchers handling data in the fields of geology or medicine.

Bimber et al.'s [2, 1, 16] mirroring fits very well in that area especially since it minimizes the distortion for the non–tracked viewers. Imagine a group of doctors discussing a CT, MRI or ultrasound scan of a patient. If some doctors see a distorted view, their ability to comment on the scan is limited. This method greatly reduces the distortion problem.

Fast hardware is necessary to render the volume and geometry data together, in stereo, without delay. We currently run our application on an SGI Octane, using a Barco Baron Virtual Table, Cristal Eyes and an Ascension Flock of Birds tracking system. The delays that occur while interacting with a 64^3 volume are hardly noticeable. Of course the delay lengthens when working on a 128^3 volume. Today working on a 256^3 volume and bigger is not practical in this environment. However, this may change as faster CPUs and graphics cards become available.

Future work will thus focus on analyzing the application behavior on faster hardware and studying the interaction improvements achieved by integrating speech recognition and force feedback. After integrating force feedback, we will try to integrate volume deformation capabilities to simulate, for example, a virtual cut [12, 29].

Figure 1: The transparent PIP and pen props (taken from [21]).

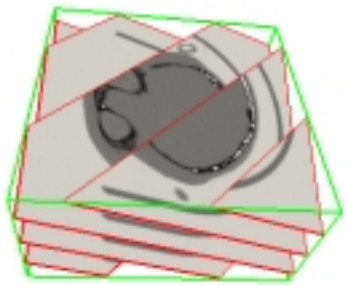

Figure 2: Volume Rendering as Texture Mapping (taken from [5]).

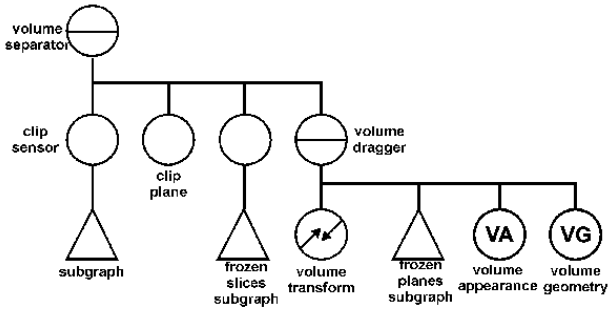

Figure 3: Simplified OpenInventor subgraph.

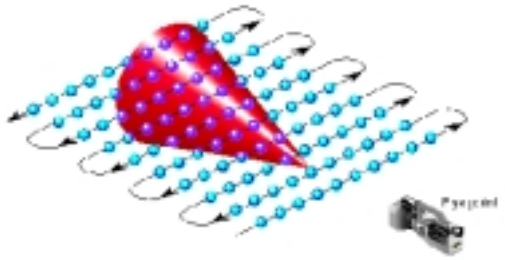

Figure 4: Volume slicing (taken from [5]).

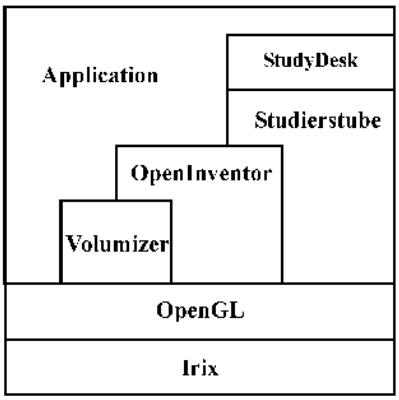

Figure 5: Layer diagram of our System.

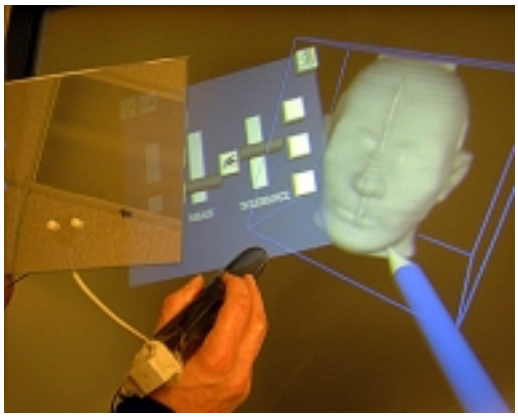

Figure 6: The Studydesk virtual environment.

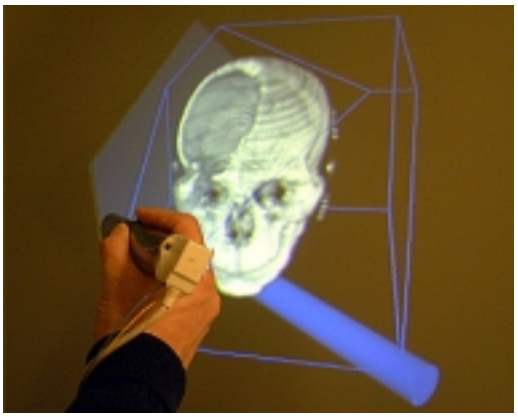

Figure 7: Cut through the head.

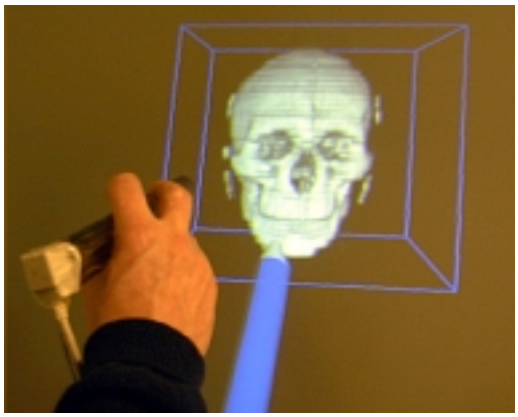

Figure 8: Start dragging the volume.

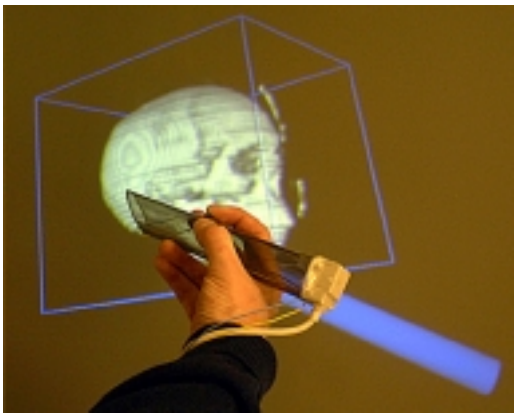

Figure 9: Dragged volume in final position.

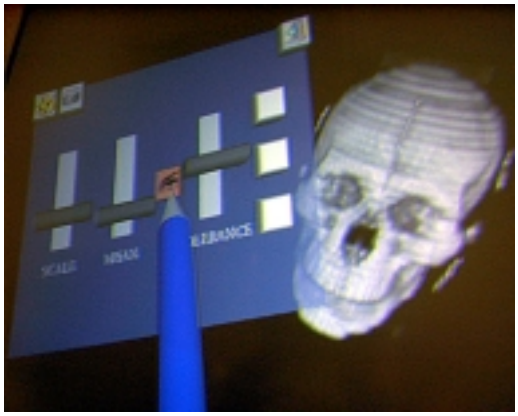

Figure 10: Activating lookup table.

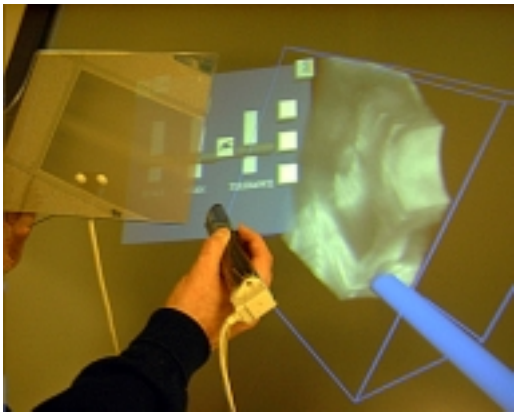

Figure 11: Ultrasound data of an foetus.

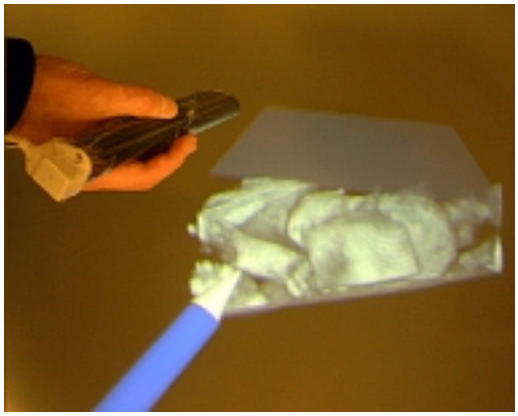

Figure 12: Ultrasound data after activating the lookup table.

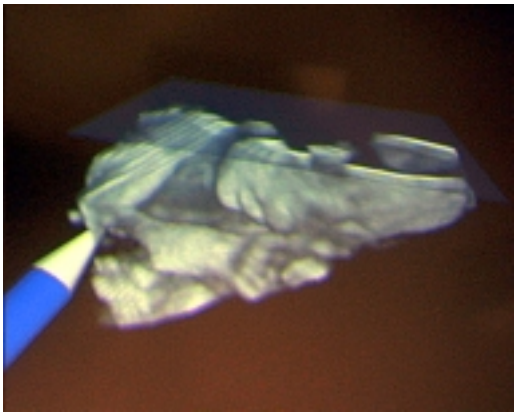

Figure 13: Foetus after clipping off all interfering data.

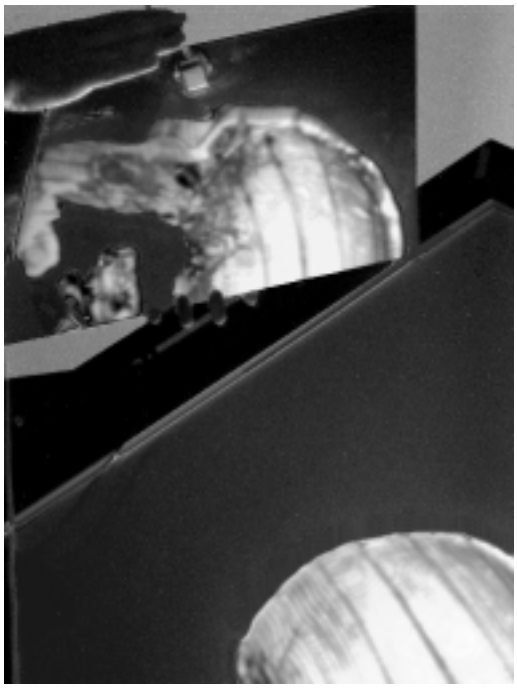

Figure 14: Mirroring: Correct reflected head on the PIP (upper left) and distorted projection of the head on the table (lower right).

References

[1] O. Bimber, L. M. Encarnação, and D. Schmalstieg. Augmented reality with back–projection systems using transflective surfaces. *Proceedings Eurographics*, 19(3), October 2000.

[2] O. Bimber, L. M. Encarnação, and D. Schmalstieg. Real mirrors reflecting virtual worlds. *IEEE VR Conference, New Brunswick, N.J.,USA*, March 2000.

[3] D. Blythe. Volume visualization with texture. *SIGGRAPH*, Course 17 Advanced Graphics Programming Techniques using OpenGL:174–182, 1998.

[4] P. Cignoni, C. Montani, and R. Scopigno. Tetrahedra based volume visualization. In Hans-Christian Hege and Konrad Polthier, editors, *Mathematical Visualization*, pages 3–18. Springer-Verlag, Heidelberg, 1998.

[5] G. Eckel. OpenGL Volumizer programmer's guide. *Silicon Graphics Inc.*, 1998.

[6] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics - Principles and Practice*. Addison-Wesley System Programming Series, second edition, 1990.

[7] M. Gervautz and Zs. Szalavári. A two handed interface for augmented reality. *Computer Graphics Forum (Proceedings of EUROGRPHICS'97)*, 16(3):335–346, 1997.

[8] J. C. Goble, K. Hinckley, R. Pausch, J. W. Snell, and N. F. Kassell. Two–handed spatial interface tools for neurosurgical planning. *IEEE Computer*, 28(7):20–26, 1995.

[9] M. B. Haley. Incremental volume rendering using hierarchical compression. Master's thesis, Faculty of Science at the University of Cape Town, 1996.

[10] A. Kaufman. Volume visualization: Principles and advances. *SIGGRAPH*, 1997. Part Course notes Papers 9-1.

[11] Kitware, Inc.
URL: http://www.kitware.com. *Visualization Toolkit (VTK)*, 2000.

[12] Y. Kurzion and R. Yagel. Volume deformation using ray deflectors. *The 6th Eurographics Workshop on Rendering*, pages 21–32, June 1995. Dublin.

[13] P. G. Lacroute. *Fast Volume Rendering using a Shear–Warp Factorization of the viewing transformation*. Departments of electrical engineering and computer science, Stanford University, September 1995. Dissertation.

[14] P. G. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Proceedings SIGGRAPH*, pages 451–458, July 1994.

[15] W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface reconstruction algorithm. *Proceedings SIGGRAPH*, (21):163–169, 1987.

[16] D. P. Mahoney. Mirror, mirror in the hand. *Computer Graphics World*, pages 18–19, March 2000.

[17] University of Stanford.
URL: http://www–graphics.stanford.edu/-software/volpack/vp_userguide.html. *VolPack*, 1995.

[18] Vienna University of Technology.
URL: http://www.cg.tuwien.ac.at/research/-vr/studierstube. *Studierstube*, 2000.

[19] Real Time Visualization.
URL: http://www.rtviz.com. *VolumePro 500*, 2000.

[20] G. Sakas and A. Pommert. Advanced applications of volume visualization methods in medicine. *Eurographics '97 State of the Art Reports, Budapest*, pages 101–143, 1997.

[21] D. Schmalstieg, L. M. Encarnação, and Zs. Szalavári. Using transparent props for interaction with the virtual table. *ACM Symposium on Interactive 3D Graphics (I3DG'99)*, pages 147–153, April 1999.

[22] D. Schmalstieg, A. Fuhrmann, M. Gervautz, and Zs. Szalavári. Studierstube – an environment for collaboration in augmented reality. *Virtual Reality – Systems, Development and Applications*, 3(1):37–49, 1998.

[23] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. Prentice Hall, 2nd edition, 1998.

[24] Silicon Graphics, Inc.
URL: http://www.sgi.com/software/-volumizer. *OpenGL Volumizer*, 2000.

[25] Template Graphics Software, Inc. URL: http://www.tgs.com/3DMS/index–c++.html. *3D DataMaster*, 2000.

[26] Template Graphics Software, Inc. URL: http://www.tgs.com/amira/. *Amira*, 2000.

[27] The Fraunhofer Center for Research in Computer Graphics, Inc. (Fraunhofer CRCG). URL: http://www.crcg/edu. *TeleInVivo*, 2000.

[28] Vital Images, Inc. URL: http://www.vitalimages.com. *VoxelView*, 2000.

[29] R. Yagel, D. Stredney, G. J. Wiet, P. Schmalbrock, L. Rosenberg, D. J. Sessanna, and Y. Kurzion. Building a virtual environment for endoscopic sinus surgery simulation. *Computer Graphics*, 20(6), December 1996.