Volume 0 (1981), Number 0 pp. 1–11

Distributed Software-Based Volume Visualization in a Virtual Environment

A.L. Fuhrmann*, R.C. Splechtna*, L. Mroz†, H. Hauser*

*VRVis research center, †TIANI MedGraph

Abstract

In this paper we present our integration of volume rendering into virtual reality, combining a fast and flexible software implementation of direct volume rendering with the intuitive manipulation and navigation techniques of a virtual environment. By distributing the visualization and interaction tasks to two low-end PCs we managed to realize a highly interactive, yet inexpensive set-up. The volume objects are seamlessly integrated into the polygonal virtual environment through image-based rendering. The interaction techniques include scalar parameterization of transfer functions, direct 3D selection, 3D highlighting of volume objects and clipping cubes and cutting planes. These methods combined with the interaction and display devices of virtual reality form a powerful yet intuitive environment for the investigation of volume data sets. As main application areas we propose training and education.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism(Virtual reality); I.4.10 [Computer Graphics]: Image Representation(Volumetric);

1. Introduction

The advantages of a virtual environment as user-interface for visualization applications have been demonstrated previously: Cruz-Neira [CN93] demonstrated the application of her CAVE^{TM}virtual environment in scientific visualization, our own adaptation of a commercial visualization system for virtual reality (VR) [FLS97], and many others [vDFL*00] have demonstrated the useful combination of Virtual Reality (VR) and visualization.

Swan et al. [Swa00] demonstrated a computational steering system, also using a virtual environment for visualization and user interaction. Within a cave 3D output is shown from time-varying data, also facilitating volume rendering as well as a VR user interface to enable steering of the simulation process.

Especially in the field of volume visualization, the application of direct manipulation widgets – geometric objects exhibiting interactive behavior – makes navigation and manipulation intuitive, efficient and informative [KKC01]. The use of 6 degree-of-freedom (DoF) input hardware allows effective control of these widgets and stereoscopic display devices deliver the necessary feedback for 3D interaction and ease the interpretation of the volume data.

Opposed to smart software solutions for fast and interactive volume rendering, special-purpose hardware is another option to gain high frame-rates. The VolumePro board [PHK*99], for example,can be used as an additional graphics board on the PC platform, performing fast volume rendering. The newer VP1000 version is able to integrate high quality (orthographic) volume rendering into OpenGL scenes. However the cost of the hardware forbids its integration into an inexpensive setup.

2. Objective

The main objective of this paper is to demonstrate our seamless integration of software-rendered volume visualization into a low-cost PC-based virtual environment. The possibility to directly interact with volume and polygonal objects alike, and the application

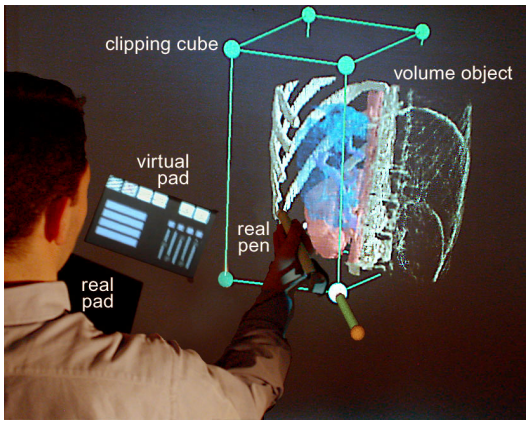

Figure 1: *Using the pen to position the clipping cube. The clipped part of the "Torso" data set is displayed as non-photorealistic contour rendering. The virtual pad displays the slider controls of the cube widget.*

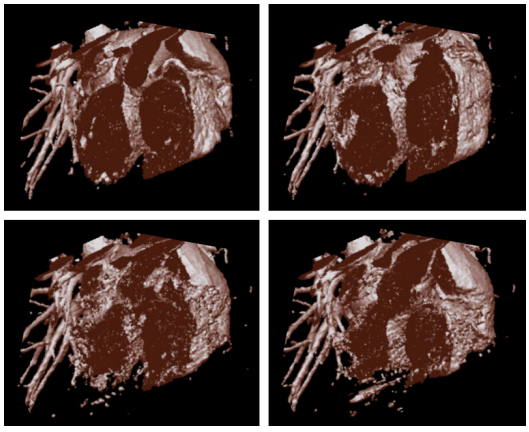

Figure 2: *Four phases of the animated "heart" data-set, front removed with cutting-plane.*

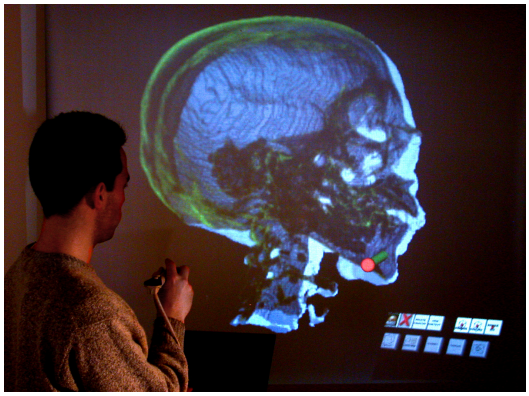

Figure 3: *The "head" data-set with threshold set to bones surface. Volume in front of the cutting-plane is displayed as green contours.*

of polygonally represented widgets, like cutting planes or light sources, on the volume data makes for a more intuitive and efficient workflow than conventional interaction modes.

In the next section of this paper, we describe the following properties of our system in detail:

- **Real-time volume visualization** Depending on the complexity of the volume data set, and on the rendering modes – e.g. surface shaded, maximum intensity projection or non-photorealistic – we reach interactive frame (stereo pair) rates of up to 25fps for a resolution of 128^3 or approximately 11fps for 256^3. The virtual environment itself delivers independently of this 60fps.
- **Interaction in the virtual environment** Using stereoscopic displays and six degree-of-freedom (6DoF) interactions devices, we realize a set of sophisticated manipulation methods: direct 3D selection of volume sub-structures with real-time graphical feedback and focus-and-context visualization via oblique and axis-aligned clipping planes.
- **Heterogeneous display** Applying image-based rendering techniques, we mix volume rendering and polygonal rendering to seamlessly integrate volume visualization and interaction widgets.

3. Integrating Volume Visualization into Virtual Reality

We aimed to integrate software volume rendering with intuitive virtual reality interaction and display techniques into a low-cost PC-based virtual environment (VE). Our environment for volume rendering in virtual reality combines two in-house developed libraries: the Real Time Volume Rendering Library (RTVR) and the Studierstube virtual environment [SFH*02]. We already have employed VR-techniques in a Visualization setting [FLS97,FG98], and applied our experiences in this new case. By distributing the volume rendering to a separate server, a fast PC with enough memory for the volume data sets, we added flexibility and scalability to the system (section 3.5). This server communicates with the client, implemented as Studierstube application, via a LAN. Images sent by the server are integrated into the polygonal renderings of the virtual environment as *image-based primitives*.

3.1. RTVR - Real Time Volume Rendering Library

The usual approach to providing volume rendering within virtual environments is to exploit the texture-mapping capabilities of 3D rendering hardware. SGI's volumizer [Sil] library may be used to obtain volume rendering functionality on SGI hardware. On the PC

platform a couple of efficient approaches for volume rendering have been published, which exploit the capabilities of inexpensive 3D gaming cards [KKC01, EKE01]. Although the performance of volume rendering using texture mapping hardware is usually superior to software-based approaches, there are some drawbacks:

- To achieve high quality and functionality of volume rendering proprietary extensions of OpenGL have to be used. Shader code required to implement a rendering technique on NVidia chips differs significantly from code for ATI chips or SGIs. Depending on implemented features of a specific hardware platform (like the number of texturing units) the performance of a specific approach to volume rendering may vary significantly.
- The amount of texture mapping memory available on 3D PC cards is quite limited. Depending on the desired features of the volume rendering, further information has to be stored for each voxel in addition to the scalar data value. To perform shaded rendering a gradient volume has to be stored - using 3×8 bit gradient vectors or a quantized representation of it (12-16 bit). Opacity modulation by gradient magnitude - used for enhancement of boundary surfaces - requires to store precomputed gradient magnitude (8 bit). The display of segmented volumes requires access to segmentation information during rendering. Flexible and high quality rendering techniques require 5 and more bytes of texture memory for each voxel of the volume [KKC01]. Thus, rendering a 512^3 volume with 16-bit data values (medical data sets usually use 12 bit voxels) requires 256 MB of texture memory. Sufficient memory is only provided by the latest generation of PC graphics cards (like some GeForce FX cards).

Using a pure software approach overcomes the dependency on specific hardware features and restrictive limitations to volume size and rendering functionality. Schulze and Niemeier [SNL01] presented an approach for utilizing perspective shear-warp for volume rendering within a CAVE environment. The CAVE setup requires to perform perspective projection for volume rendering, which is slower than parallel projection, and allows to render only small data sets (32^3–64^3) at frame rates suited for tracked environments.

RTVR (Real Time Volume Rendering Library) is a framework of Java classes for interactive volume rendering [MH01]. High rendering performance is achieved by working with sets of voxels representing structures of interest within a volume, instead of rendering the entire monolithic volume. For this reason, rendering performance mainly depends on size and structure of the visualized features of the volume, rather than on the size of the volume itself. For typical visualizations of 256^3 data sets from medicine this approach allows to render up to 35 frames per second on a single CPU (See section 5).

RTVR uses two-level volume rendering [HMBG00] an extension of standard 3D visualization in such a way that it allows to do object-aware volume rendering. Starting out with a segmented volume, for each and every object within the data, an individual rendering mode – ranging from direct volume rendering through compositing, maximum-intensity projection, and surface rendering, to non-photorealistic rendering (NPR) of contours only – can be assigned. On a global level, renderings of objects are combined using compositing based on accumulated transparencies.

The subdivision of the volume into "objects", structures of interest, allows simple inclusion of segmentation information and fusion of data from multiple volumes. Each object is rendered using its own set of rendering parameters, like transfer function, shading model, non-photorealistic shading [CMH*01a]), and compositing modes, like direct volume rendering, maximum intensity projection. Thus it is possible to choose the most appropriate rendering method for each object, depending on the visualization goal and the structure of the object itself. Clipping can also be performed on a per-object basis, which allows effective use of clipping for revealing structures contained within other objects. For providing context information, RTVR allows clipped parts of objects to be rendered using a separate set of rendering parameters. For example, showing clipped data with very low opacity, or displaying just contours of objects using a non-photorealistic shading technique.

In summary, four major reasons can be given why a RTVR has been chosen to provide volume rendering functionality instead of a hardware based approach:

- The ability to fuse data from multiple, reasonably large volumes
- Efficient rendering of time-varying data
- Performance comparable to and even higher than methods based on texture mapping for typical visualization scenarios.
- High flexibility with respect to visualization parameters, e.g. two-level volume rendering.
- Stereo pairs of images can be rendered in parallel, since RTVR supports multi-threading on PCs with multiple CPUs or hyperthreading.

3.2. Studierstube

Studierstube [SFH*02] is a multi-user augmented environment, which we already applied to problems in the area of scientific visualization [FLS97, FG98]. It

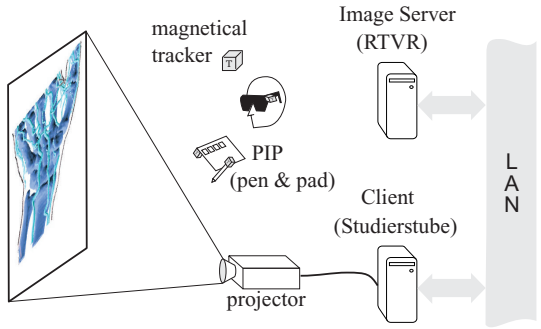

Figure 4: *The hardware set-up used for the illustrations in this paper. For documentation purposes, a front-projection, monoscopic installation was used.*

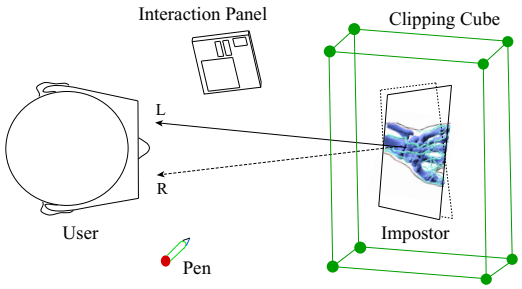

Figure 5: *Impostor rendering of volume data (top view). Two quads act as impostors, displaying different views of the volume object to the user's eyes. Interaction elements (pen, pad, & clipping cube) are modelled as polygonal geometry.*

implements basic interaction methods like positioning objects by dragging them with a 6-DoF pen as well as conventional 2D interaction elements, like sliders, dials, and buttons for parameterizations of visualization methods. These purely virtual interface elements are positioned on the *Personal Interaction Panel* (PIP [SG97]), a handheld tablet (figure 1). Users hold this board in their non-dominant hand while they make adjustments to the interface elements with the same pen they use for 6 DOF interaction (figure 1).

Studierstube supports a wide range of input and output devices and is thus configurable for many different hardware setups. For reasons described in section 3.4 we selected a projection setup. The user stands in front of a stereo-projection screen, wearing polarized glasses and interacts with the virtual environment using the magnetically tracked tablet and pen combination described above (figure 1). The same magnetic tracker (an Ascension Flock of Birds) is used to track the user's head position, so that the projection can be updated accordingly when the user moves.

Studierstube relies on polygonal rendering via OpenGL. A previous implementation of volume visualization in Studierstube [WSE00] employs OpenGL Volumizer [Sil], a library distributed by SGI, which uses textured polygons for volume visualization. The disadvantages of this approach are well-known: even when the OpenGL 3D texture extension is supported on the particular graphics hardware, aliasing artifacts appear. Furthermore, advanced rendering modes like surface shading or non-photorealistic contour rendering cannot easily be simulated using this approach.

We circumvented these problems by combining polygonal rendering with direct volume rendering, using an image-based approach.

3.3. Image-Based Rendering of Volume Objects

We apply *image-based rendering* to integrate the volume visualization into our polygon-based OpenGL rendering of our virtual environment. This means that the images sent from the server are used as textures on a quad which is used as an impostor for the volume object. The server receives the two directions – one for each eye – for which the volume has to be rendered. The size of the impostors does not have to be varied, since normal perspective rendering of the quads takes place afterwards.

Actually we have to use two impostors: one for each of the two stereoscopic views. These two quads are turned around their center so that their surface normal always faces the respective eye of the user (figure 5). When the user changes his (or her) position, the head-tracker (see section 3.2) reports this change to the impostors, which update their orientation accordingly and send a request for new volume renderings to the server. As soon as the image pair is delivered, they are displayed on their respective quads. The main problem besides the rendering rate of the server was the network delay (lag, section 3.5), which resulted in a delayed response when manipulating the volume data.

This update process happens asynchronously in a background thread, which has the advantage of not impairing responsiveness of the virtual environment: during the update time – which, depending on the complexity of the volume data set and the rendering

mode can vary from 20ms up to 500ms – the environment does not freeze but continues to update according to the users interactions, the pen graphical feedback keeps moving smoothly.

The disadvantage of this decoupling between image update and impostor display becomes visible when turning the volume object quickly: The impostors reorient themselves in real-time, strictly speaking with the update rate of the polygonal environment, which in our case comes to approximately 60Hz. The textures on the quads, on the other hand, get updated only every few frames. When moving slowly, this does introduce noticeable artifacts, but when a quick rotational change is introduced, artifacts depending on the re-orientation of the impostor's geometry in the image plane can be extremely annoying.

To reduce these artifacts, which appear as sawtooth-like rotational movements of the volume object, one has to minimize the rotation of the quad in the image plane. This can be done by performing an *incremental update* of the quad's orientation: instead of calculating a rotation matrix between the initial orientation of the quads normal and the viewing direction (leaving one degree of rotational freedom undefined), we calculate a correctional rotation between the normals direction in the previous frame and the new viewing direction. The rotational axis for this update fixed in the image plane. This minimizes the rotation of the impostor in image-space and the resulting artifacts. Since the incremental is always taken between the correct new position and the previous result, errors do not add up.

There are changes of viewpoint where the fast update of the impostor images is not so important: movement directly along the direction to the impostor (front to back) and rotations in the image plane (tilting the head left or right). In these cases, lower update rates can be tolerated (or faster movements can be performed for a given update rate).

3.4. Pseudo-Perspective

Like other fast volume rendering implementations (e.g. VolumePro [PHK*99]), RTVR implements parallel projection only. While a straightforward implementation of correct perspective projection in RTVR is possible, the computational overhead would reduce frame rates in most cases below the limit acceptable for VR applications.

Most VR set-ups, on the other hand, depend on correctly rendered perspective images. To solve this dilemma, we integrated the volume data as pseudo-perspective images: The texture maps for the impostors are rendered in parallel projection, while the impostors geometry (quads) are rendered perspectively distorted. So the volume objects appear, while inherently parallel projected, larger or smaller depending on the distance between the user's viewpoint and the impostor. Since the impostors maps are rendered separately for each eye, the volume object can be viewed as stereo.

In our projection setup, this proved to be a viable solution. Head-mounted displays on the other hand tempt the user to approach the objects or even put the viewpoint inside the volume, a case where pseudo-perspective breaks down.

The discrepancies between the pseudo-parallel projected impostors and the correct perspective of the interface becomes only noticeable when the users viewpoint is close to the volume object, e.g. about one impostor diameter away.

3.5. Networking

We use a client-server approach for two reasons: Firstly, a network connection provides a flexible coupling between the Java (RTVR) and C++ (Studierstube) parts of the application. Secondly, delays in the volume rendering process do not impair interaction mechanisms in VR if the two processes are executing asynchronously. Furthermore, we gain added scalability: by distributing client and server-processes on different machines, multi-user environments can be implemented without reduction in performance.

One distinction between this system and our previous experience in distributed visualization [FLS97] lies in the distributed rendering approach we use here. While the virtual environment handles all of the user's interaction and navigation, volume rendering is performed on another PC and the rendered images are transferred via the network connection.

We implemented an asynchronous protocol for the image updates: every change of visualization parameters – e.g. viewpoint or transparency – results in an update-request message being sent to the server. During volume rendering, the server ignores the packets. Only when an image pair has been completed, the server processes the youngest received message and starts a new rendering. This strategy has a lower latency than a complete request/acknowledge cycle for each image.

Since we need low network latency as well as high throughput, we implemented the communication between Studierstube and RTVR with UDP/IP, the connectionless, unreliable cousin of TCP/IP [Ste98]. UDP provides unbuffered, message-oriented communication, thereby minimizing protocol overhead. The

lower reliability (essentially only of concern when used in a WAN) was in our case of lesser concern, since the continuous transmission of images from the RTVR server guarantees idempotent updates.

4. VR Interactions with Volume Objects

For full interaction with the volume object from within the virtual environment, we need essentially three different input streams:

- *Symbolic input* e.g., via a keyboard, is where VR normally falls short: gesture input or speech recognition are still relatively unstable and computationally expensive. Luckily, we need symbolic input only when selecting a new data set, which can be done by selecting a file in a standard "File open" dialog on the PIP.
- *Scalar parameterization* of the volume visualization needs input elements capable of adjusting continuous or discrete input values. This is described in detail in section 4.1.
- *Direct 3D interaction* makes the advantages of interaction in VR apparent, interaction of inherently three-dimensional properties like position, distance, or rotation to be precise. The 6DoF input devices normally employed in VR settings naturally excel at direct interaction. A detailed description of the direct interaction techniques we apply can be found in the following sections.

4.1. Scalar Parameterization

We use the PIP (section 3.2) as primary input device for all scalar parameterization (as opposed to direct 3D interaction, see section 4). Examples for such interaction are adjustment of transparency or iso-surface thresholding. These two parameters for example can be adjusted by moving sliders like the ones depicted in figure 1. These sliders mimic the action of classical 2D graphical user interface elements in 3D.

Additional input elements are dials, which can be used for making precise adjustments over wide value ranges and 3D color selectors. Standard buttons are used to switch between rendering and interaction modes (e.g. from "cutting plane" to "manipulation").

All these widgets can be placed on the pad and are always literally "at hand" when the user needs them.

4.2. Navigation

The simplest direct interaction technique is *navigating* the volume data. When switched to navigation mode, the user can move the object by simply placing the pen inside its bounding box and pressing the pen's button.

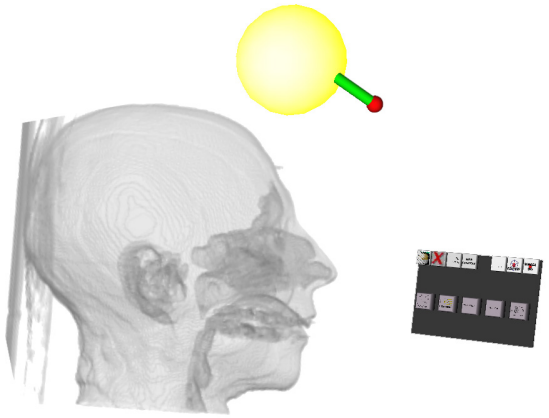

Figure 6: *Positioning the lighting widget with then pen. The "head" data set is displayed as transparent surface rendering with the threshold at the air/tissue boundary. The head cavities are clearly visible. The virtual pad of the PIP displays the buttons for mode selection.*

As long as the button stays pressed, the objects follow faithfully all motions and rotations of the pen. The only restrictions in movement are imposed by the tracker's range and the pseudo-perspective rendering (section 3.4).

4.3. Lighting

Changing the light direction is also very simple: in *lighting mode*, a sphere indicates the position of the light source (figure 6). This sphere can be moved in the same way as the object. Two restrictions apply: again due to pseudo-perspective rendering, only the direction of the sphere relative to the center of the volume object matters, not the distance, and the light source only affects the volume object. The latter is actually a feature, not a bug, since wrong lighting could make the PIP pad unreadable.

4.4. Cutting Plane & Clipping Cube

One visualization technique that we have already employed [FG98] is *focus-plus-context* (section 4.6). One implementation is the selective clipping of the object to focus on a certain part.

Clipping planes are implemented in OpenGL and allow hardware accelerated clipping of polygonal objects. This approach, of course, does not suffice for our volume objects, since it would only clip the impostor quad. Since RTVR already supports axis-aligned as well as oblique clipping planes, we only had to support the necessary interaction elements on the client.

A *Clipping Cube* is represented by a wireframe cube with spheres at the corners (figure 1). These spheres light up when the pen enters them and can be moved arbitrarily. This resizes the cube non-uniformly while keeping it axis-aligned. The faces of the cube indicate the six clipping planes inside the volume object.

The action to be taken for the parts of the object outside the cube can be set differently: if real clipping is needed – when only the cuts through the object are of interest – the outside parts can be set to "invisible". If a real focus+context visualization is to be performed, a more subtle approach changes the outside parts to transparent and/or switches to non-photorealistic contour rendering (figure 3).

The *Cutting Plane* works like a single face of the clipping cube: it divides the volume object into focus and context along its plane. The main difference is its arbitrary alignment, which can be used to place oblique cuts through the volume. As with the clipping cube it can be used to change the display of the "cut" parts of the volume (figure 7).

As interaction method a direct application of the pad was obviously an intuitive way to place the plane. After switching to "cutting" mode, the pad itself acts as a cutting plane. After placing the plane in the right position, a click with the second pen button ends the "cutting" mode, freezing the cutting plane in place.

We had to use an additional button, since we use the primary pen button for moving the volume object during the interaction. This means that in "cutting" mode, real *two-handed interaction* can be employed: one hand moves the cutting plane via the pad and the other moves the object via the pen.

4.5. Direct Selection of Volume Substructures

As mentioned before (section 3.1), RTVR uses segmented objects to further subdivide a volume data set. In the case of our "Torso" data-set (figure 1), for example, skeleton, kidneys, and stomach are separately segmented objects. For all these objects, we can separately – or in groups – specify visualization parameters. We can select for a focus+context visualization all objects we are not primarily interested in and switch them to semi-transparent or NPR. We can also select them and clip their front using the cutting plane method, while the object we are interested in – e.g. the kidneys, figure 7 – remains unaffected.

One (trivial) method to implement this selection was a row of checkboxes on the pad (figure 1, left on the virtual pad), which can be toggled to indicate the selection state of their associated volume object. This approach is easily implemented and works even from a distance, but is not very intuitive, especially when convoluted objects – like vessels – can not always be easily distinguished in places. A more intuitive approach would be the direct selection of an object by clicking the pen inside it.

Occlusion

The combination of image-based rendering with polygon-based interaction widgets yields some problems. While the stereoscopic cues work fairly well in integrating the volume object into the polygonally defined surroundings, e.g., clipping cube, pen & pad, the occlusion between volume and polygonal objects is difficult to get right. Since the volume object is only represented as a single quad in the z-buffer of the client, occlusion can normally only occur between this plane and any widget. Especially when trying to select parts of the volume object directly by clicking inside the volume, this arbitrary occlusion would be confusing. The pen would half of the time be hidden behind the impostor plane, which would make the selection of these parts of the volume object extremely difficult. Even if we where to integrate z-depths into the impostor [Sch97], we could still not correctly handle occlusion between pen and semi-transparent volume objects.

To enable direct interaction with the volume objects, we employ a volume rendered pen inside the volume data set. As soon as the pen enters the bounding box of the volume depicted on the impostors, we hide the polygonal pen and substitute it by a volume rendered cylinder. This pen is correctly occluded even by semi-transparent volume objects. The only disadvantage is that this pen now is only updated by the impostor update rate.

Highlighting

This highlighting can only be performed in cooperation with the server, since only the server knows which volume object the current pen coordinates belong to. The client sends the pen position continuously as long as it is inside the volume objects bounding box. The server responds by highlighting the sub-object that touching the current pen position. Highlighting can be any combination of rendering parameters and modes and is specified on a per-object basis. In most cases we simply reduce transparency and lighten the color of the sub-object. When the object is rendered NPR, we additionally switch the rendering mode to surface shaded.

4.6. Focus+Context Visualization

A typical example of how Focus+Context (F+C) visualization is realized is depicted in figure 1: the user

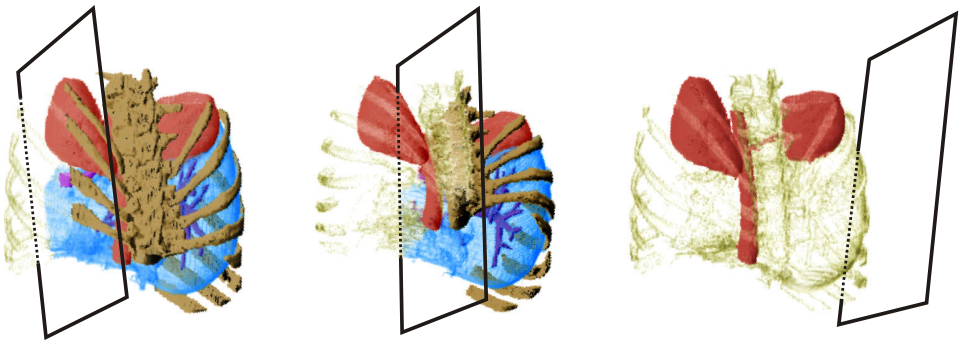

Figure 7: *Moving a cutting plane (illustrated as black rectangle) with the PIP. Context shown as non-photorealistic rendered contour. Only the not selected parts are clipped, the kidneys (red) remain shaded.*

has switched to "clipping cube" mode on the PIP, as is now dragging one corner (highlighted sphere, lower right) to the left, thereby resizing the clipping cube widget and moving one of its faces to bisect the data set of a human torso at the spine. The context part (i.e. the part outside the cube widget) is displayed as non-photorealistic rendering, showing only the contours of the clipped data.

Non-photorealistic rendering has been recently introduced to volume visualization [RE01, CMH*01b]. Bringing more input into the transfer function mapping, for example, the viewing vector, advanced effects like contour rendering can be achieved, independently from which direction the data is investigated. In our case non-photorealistic rendering proved to be especially useful for rendering the context in a focus+context mode of volume rendering (focus being rendered as surface, context as contour, for example).

Figure 7 shows a more sophisticated case, where only the selected segmented objects are clipped. Using the PIP as direct interaction tool, a cutting plane is moved through the visualization. The kidneys (red), which have been previously unselected by clicking the pen inside them, are not clipped, but remain shaded even when the half-space defined by the plane covers the whole torso (figure 7, right).

In this example we see two different ways of F+C visualization: first, the direct selection of the focus object; second, the geometric placement of the plane. By placing the plane along the spine, bisecting the torso (figure 7, middle), we visualize the left kidney in focus, thereby further refining the previous selection.

4.7. Time-Varying Data

The way RTVR handles volume data for rendering - storing sets of voxels which represent objects instead of the entire volume - allows to efficiently render time varying data avoiding the usual problems with memory bandwidth. By keeping in memory only voxels belonging to objects of interest more time-steps can be kept in RAM than possible with most other software renderers. Since RTVR performs purely software rendering, it also does not suffer bandwidth limitations of the same order as most hardware volume renderers, which depend on loading data (up to 5 bytes per voxel for high-quality shaded rendering) into texture memory for each new time-step. While extremely memory-consuming, animated volume data from simulations or sophisticated CT systems (section 5) can be visualized and interacted with in the same way as static volume data, as can be seen in figure 8. Our animated heart data-set (section 5) consists of 12 animation phases, each stored as separate volume object (figure 2).

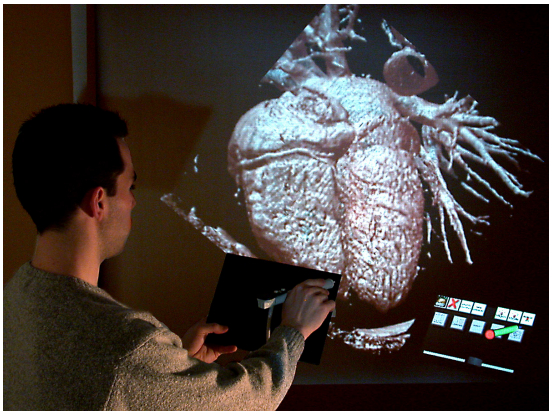

Figure 8: *The time-varying* heart *data set consists of 12 animation phases.*

5. Application and Discussion

We evaluated our system using several medical computer tomographic volume data sets:

- *head*: human head, only one volume object.
- *skull*: human head, two segmented objects: skin and skull.
- *torso* lower human torso, four segmented objects: bones, liver, liver vessels, kidneys.
- *heart* human heart, one animated object consisting of twelve animation phases.

The hardware set-up consisted of two PCs: as server we used a 1.8GHz Xeon with 1GB RAM, connected via a 100Mbit LAN to the VR client on a 1.4GHz Athlon XP with 512MB RAM and a Nvidia geForce3 graphic card. We used a DTrack optical tracker and a VRex stereoscopic Projector with a resolution of 1024×768 for use with polarizing glasses. For documentation purposes all the photos in the paper were taken with monoscopic display to avoid artifacts.

A comparison of the realized update rates for the different data sets is shown in table 1. The monoscopic update rates are approximately twice the stereoscopic rates shown in the table. These rates are the updates of the volume objects only, the polygonal components of the environment – widgets, pen, pad – were updated with video rate (i.e. 60Hz).

With the *head* data-set we tested variations in threshold, transparency and lighting. Setting the threshold to display the air/tissue boundary and raising transparency provided a visualization of the internal cavities of the head (figure 6). Adjusting transparency and threshold was easily done with the sliders, and the lighting direction adjusted by positioning the "light source" widget (section 4.3).

The *skull* data set consists of two segmented objects. We generated figure 9 by making the skin transparent and using the cutting plane only on the skull. Note the non-photo realistic contours of the skull.

The *torso* data set contains several segmented volume objects: liver, liver vessels, kidneys, and bones (spine and ribs). We used the torso for testing more complex interaction: direct 3D selection of objects and clipping of selected objects. One aspect of our implementation proved to be especially useful in these complex cases: the server retains the previous state for all not selected objects. When changing any of the parameters – e.g. position of cutting plane or threshold – of the selected objects, the other objects kept their state, allowing to successively adjust different parameters of different object sets.

Our only animated data-set consists of twelve phases of a beating *heart* with contrast agent, captured by a CT with ECG synchronization. Here the frame-rate dropped significantly (see table 1), but we were nevertheless able to work interactively in placing the cutting plane (figure 8).

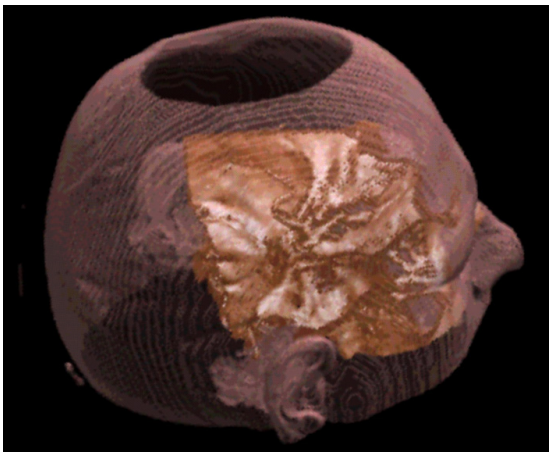

Figure 9: *Screenshot: The skin of the "skull" data set is displayed as transparent surface over bones clipped with the clipping-cube widget.*

As can be seen from table 1, the rendering time depends more on the number of the visualized objects within the volume and on the visualization methods used than on the size of the volume itself. Typical setups for medical data which include one or more objects rendered as surfaces, and small structures (for example vessels) rendered using a truly volumetric rendering method can be rendered at frame rates as shown in 1, thus allowing interactive stereo projection. Using separate computation servers for left and right eye images acceptable frame-rates could be possible even for visualizing objects from 512^3 volumes.

As expected, the solution for occlusion between polygonal widgets and volume objects described in section 4.5 worked only as a compromise: when the pen entered a volume object, the conflict between the occlusion cue – i.e. the pen still hiding the front of the object – and the stereoscopic depth cue was considered an irritation by most users. Nevertheless our solution was better than leaving the occlusion to the impostor geometry, thereby hiding the pen completely in the back half of the object.

The direct selection of segmented volume objects described in section 4.5 works only well in stereoscopic mode with correctly calibrated head-tracking, otherwise errors in depth perception lead to "overshooting" behavior when selecting small features. This was especially noticeable while producing images and videos for the documentation in monoscopic mode.

data set	objects	resolution	updates/s
head	1	128^3	33.7–35.1
skull	2	256^3	16.3–17.0
torso	4	256^3	9.3–10.6
heart	12(phases)	256^3	11.1–12.3

Table 1: *Volume Rendering updates for different data sets. Updates are for stereo pairs per second, min/max values given.*

6. Future Work

As mentioned in section 3.2, the feature of Studierstube to provide viewpoints for multiple users can be integrated by essentially distributing the volume rendering process on one server per user. A further speedup – with only slight modifications to the network protocol – can be reached by using one RTVR server per eye, effectively doubling the frame rate.

Further input widgets can be provided both in 2D on the PIP or in 3D (6 DoF): a sophisticated widget for defining the transfer function, like the one developed by Kniss et.al [KKC01], can be integrated easily into Studierstube.

Since RTVR exhibits several shortcomings - only parallel projection, CPU intensive - when applied to virtual reality, we are currently working on an integration of our GPU-based volume renderer into VR.

We are planning to test our system as supplemental teaching aid in basic anatomy in cooperations with the Vienna General Hospital. Preliminary demonstrations for medical students have resulted in positive feedback, and the backprojection setup allows us to demonstrate the volume models to groups of up to 30 students.

7. Conclusions

We have presented a distributed volume visualization system integrated in a virtual environment. The implementation applied pure software-rendered volume visualization on a server PC. The client part of the system enhanced the image-based rendering of the volume data supplied by the server by 3D interaction widgets for focus+context visualization and transfer function definition. The client-side visual feedback was realized through polygonal widgets and via highlighting of the segmented volume objects.

We discussed the integration aspects of the image-based volume objects into the polygon based virtual environment and its interaction methods and concluded the paper with some examples how our system works on medical CT data.

Acknowledgements

This work has been performed in the scope of the basic research at the VRVis Research Center (`http://www.VRVis.at/`) in Vienna, Austria, which is funded by the Austrian governmental **K plus** program. The authors also thank **TIANI MedGraph** (`http://www.tiani.com/`), i.e., one of VRVis' partner companies, for providing CT data to evaluate our system, UNC at Chapel Hill for the head data set and B. Fuhrmann and R.F. Tobler for their help with this paper. Last, but not least we thank Berk Özer, who implemented most of the system.

References

[CMH*01a] Csébfalvi B., Mroz L., Hauser H., König A., Gröller E.: Fast visualization of object contours by Non-Photorealistic volume rendering. In *Proceedings Eurographics 2001* (2001), pp. 452–460.

[CMH*01b] Csébfalvi B., Mroz L., Hauser H., König A., Gröller E.: Fast visualization of object contours by non-photorealistic volume rendering. In *EG 2001 Proceedings*, Chalmers A., Rhyne T.-M., (Eds.), vol. 20(3) of *Computer Graphics Forum*. Blackwell Publishing, 2001, pp. 452–460.

[CN93] Cruz-Neira C.: Scientists in wonderland: A report on visualization applications in the CAVE virtual reality environment. In *Proceedings of the Symposium on Research Frontiers in Virtual Reality* (San Jose, CA, USA, Oct. 1993), IEEE Computer Society Press, pp. 59–66.

[EKE01] Engel K., Kraus M., Ertl T.: High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01* (2001), Annual Conference Series, Addison-Wesley Publishing Company, Inc., pp. 9–16.

[FG98] Fuhrmann A., Gröller E.: Real-time techniques for 3D flow visualization. In *IEEE Visualization '98* (1998), Ebert D., Hagen H.,, Rushmeier H., (Eds.), IEEE, pp. 305–312.

[FLS97] Fuhrmann A., Löffelmann H., Schmalstieg D.: Collaborative augmented reality: Exploring dynamical systems. In *Proceedings of the 8th Annual IEEE Conference on Visualization (VISU-97)* (Los Alamitos, Oct. 19–24 1997), Yagel R., Hagen H., (Eds.), IEEE Computer Society Press, pp. 459–462.

[HMBG00] Hauser H., Mroz L., Bischi G.-I., Gröller M.: Two-level volume rendering - fusing MIP and DVR. In *Proceedings Visualization 2000* (2000), Ertl T., Hamann B.,, Varshney A., (Eds.), IEEE Computer Society Technical Committee on Computer Graphics, pp. 211–218.

[KKC01] Kniss J., Kindlmann G., , C.Hansen: Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings IEEE Visualization 2001* (2001), IEEE, pp. 255–262.

[MH01] Mroz L., Hauser H.: Rtvr - a flexible java library for interactive volume rendering. In *Proceedings IEEE Visualization 2001* (2001), IEEE, pp. 279–286.

[PHK*99] Pfister H., Hardenbergh J., Knittel J., Lauer H., Seiler L.: The volumepro real-time ray-casting system. In *Siggraph 1999, Computer Graphics Proceedings,* (Los Angeles, 1999), Rockwood A., (Ed.), Annual Conference Series, ACM Siggraph, Addison Wesley Longman, pp. 251–260.

[RE01] Rheingans P., Ebert D.: Volume illustration: Nonphotorealistic rendering of volume models. In *IEEE Transactions on Visualization and Computer Graphics* (2001), vol. 7(3), IEEE Computer Society, pp. 253–264.

[Sch97] Schaufler G.: Nailboards: A rendering primitive for image caching in dynamic scenes. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97* (London, UK, 1997), Springer-Verlag, pp. 151–162.

[SFH*02] Schmalstieg D., Fuhrmann A. L., Hesina G., Szalavári Z., Encarnação L. M., Gervautz M., Purgathofer W.: The studierstube augmented reality project. *PRESENCE: Teleoperators and Virtual Environments 11*, 1 (February 2002).

[SG97] Szalavari Z., Gervautz M.: The personal interaction panel — A two-handed interface for augmented reality. *Computer Graphics Forum 16*, 3 (Sep 1997), C335–346.

[Sil] Silicon Graphics Inc.: SGI OpenGL Volumizer homepage. `http://www.sgi.com/software/volumizer/`.

[SNL01] Schulze J. P., Niemeier R., Lang U.: The perspective shear-warp algorithm in a virtual environment. In *Proceedings IEEE Visualization 2001* (2001), IEEE, pp. 207–213.

[Ste98] Stevens W.: *UNIX Network Programming, Interprocess Communications*, second ed., vol. 2. Prentice-Hall, Upper Saddle River, NJ 07458, USA, 1998.

[Swa00] Swan J. E.: A computational steering system for studying microwave interactions with missile bodies. In *Proceedings Visualization 2000* (2000), IEEE Computer Society Technical Committee on Computer Graphics, pp. 441–444.

[vDFL*00] van Dam A., Forsberg A. S., Laidlaw D. H., LaViola, Jr. J. J., Simpson R. M.: Immersive VR for scientific visualization: A progress report. *IEEE Computer Graphics and Applications 20*, 6 (Nov./Dec. 2000), 26–52.

[WSE00] Wohlfahrter W., Schmalstieg D., Encarnação M.: Interactive volume exploration on the studydesk, June 2000. `http://www.cg.tuwien.ac.at/research/vr/studierstube/medidesk/`.