Interactive Volume Rendering in Virtual Environments

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Jürgen Peter Schulze-Döbold

aus Dortmund

Hauptberichter:	Prof. Dr. T. Ertl
1. Mitberichter:	Prof. Dr. R. Rühle
2. Mitberichter:	Prof. Dr. C. Hansen
Tag der mündlichen Prüfung:	8. August 2003

Institut für Visualisierung und Interaktive Systeme der Universität Stuttgart

2003

Abstract

This dissertation is about the interactive visualization of volume data in virtual environments. Only data on regular grids will be discussed. Research was conducted on three major topics: visualization algorithms, user interfaces, and parallelization of the visualization algorithms.

Because the shear-warp algorithm is a very fast CPU-based volume rendering algorithm, it was investigated how it could be adapted to the characteristics of virtual environments. This required the support of perspective projection, as well as specific developments for interactive work, for instance a variable frame rate or the application of clipping planes. Another issue was the improvement of image quality by the utilization of pre-integration for the compositing.

Concerning the user interface, a transfer function editor was created, which was tailored to the conditions of virtual environments. It should be usable as intuitively as possible, even with imprecise input devices or low display resolutions. Further research was done in the field of direct interaction, for instance a detail probe was developed which is useful to look inside of a dataset. In order to run the user interface on a variety of output devices, a device independent menu and widget system was developed.

The shear-warp algorithm was accelerated by a parallelization which is based on MPI. For the actual volume rendering, a remote parallel computer can be employed, which needs to be linked to the display computer via a network connection. Because the image transfer turned out to be the bottleneck of this solution, it is compressed before being transferred.

Furthermore, it will be described how all the above developments were combined to a volume rendering system, and how they were integrated into an existing visualization toolkit.

Kurzfassung

Diese Dissertation befasst sich mit der interaktiven Visualisierung von Volumendaten in virtuellen Umgebungen. Es werden ausschließlich Daten auf regulären Gittern behandelt. Forschungsarbeiten wurden in drei Themengebieten durchgeführt: Visualisierungsalgorithmen, Benutzerschnittstellen und Beschleunigung der Darstellung durch Parallelisierung.

Da der Shear-Warp-Algorithmus einer der schnellsten CPU-basierten Algorithmen zur Volumenvisualisierung ist, wurde untersucht, wie er an die Gegebenheiten von virtuellen Umgebungen angepasst werden kann. Dazu wurde zum einen die perspektivische Projektion ermöglicht, zum anderen wurden spezielle Entwicklungen für interaktives Arbeiten durchgeführt, wie beispielsweise eine wählbare Bildberechnungsdauer oder die Unterstützung von Schnittflächen. Ein weiterer Punkt war die Erhöhung der Darstellungsqualität durch die Einbindung von Vorintegration bei der Bildberechnung.

Im Bereich der Benutzerschnittstellen wurde an einem für virtuelle Umgebungen geeigneten Transferfunktionseditor gearbeitet. Er sollte möglichst intuitiv zu benutzen und auch mit ungenauen Eingabegeräten oder niedrigen Bildschirmauflösungen noch bedienbar sein. Weitere Aktivitäten fanden im Bereich der direkten Interaktion statt, beispielsweise die Unterstützung einer Lupenfunktion, um das Innere eines Datensatzes zu betrachten. Um diese Benutzerschnittstelle auf verschiedenen Ausgabegeräten einsetzen zu können, wurde eine geräteunabhängige Bibliothek von Menüelementen erstellt.

Eine Beschleunigung des perspektivischen Shear-Warp-Algorithmus konnte erreicht werden, indem die Bilderstellung mit MPI parallelisiert wurde. Dabei kann ein Parallelrechner eingesetzt werden, der örtlich vom Visualisierungsrechner getrennt und durch eine Netzwerkverbindung angebunden ist. Da sich die Übertragung des berechneten Bildes als Flaschenhals herausstellte, wird es vor der Übertragung komprimiert.

Im Anschluss wird erläutert, wie die Einzelergebnisse zu einem Gesamtsystem zur Volumenvisualisierung kombiniert und in ein bestehendes Visualisierungssystem eingebunden wurden.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Volumetric datasets are generally characterized by a set of values at certain positions in three-space without connectivity information. This dissertation will deal exclusively with volumetric datasets consisting of scalar data values on regular grids. The goal of this dissertation is to improve both the quality of the visualization and the usability of the interaction methods. The greatest realism of any type of visualization can be achieved in immersive virtual environments with interactive graphics. Therefore, only interactive rendering methods suitable for immersive visualization with stereo imaging will be considered. This dissertation will address corresponding volume rendering algorithms, appropriate user interfaces, and parallelization issues. Finally, all the developed volume rendering methods will be integrated into an existing visualization framework.

1.1 Motivation

Only after the appearance of multi-pipe graphics computers was it possible to drive virtual environments consisting of multiple synchronized screens delivering stereo images. When the research for this dissertation began in 1999, there was no volume rendering library available which would run in a virtual environment driven by a multi-pipe SGI Onyx2 and which could be used in conjunction with SGI's OpenGL Performer. A straightforward implementation using 3D texturing hardware acceleration was the first step towards the goal of real-time volume rendering in virtual environments. Soon it showed that volume rendering requires a transfer function editor and several options for interaction so that users can explore volumetric datasets.

The computing center of the University of Stuttgart did not only have the Cube, which is a 4-sided CAVE-like virtual reality device, but also high-speed network connections to a variety of parallel computers like a Cray T3E, a NEC SX-5, and a SUN Fire cluster. This was a great opportunity to find out if CPU-based volume rendering on a remote parallel computer, displaying the visualization result on a local computer, can catch on with the latest graphics hardware based approaches. It initiated the research in remote volume rendering using the shear-warp algorithm.

1.2 Contributions

Parts of the research presented in this dissertation were carried out in collaboration with other researchers. In these cases, the collaborators will be mentioned in the respective sections. However, it will be reported only on those parts of the joint work that were decisively carried out by the author of this thesis.

The major scientific contributions of this dissertation are summarized in the following list:

- **The perspective shear-warp algorithm:** Since Lacroute's shear-warp algorithm [53] is a very fast CPU-based volume rendering algorithm, and due to its good scalability on parallel computers, it was expected to be well suited for interactive virtual environments. However, due to stereo projection and non-flat projection screens, it could not be used because existing implementations could only do orthogonal projection. Therefore, the algorithm was extended to perspective projection and adapted to some specific requirements of virtual environments. This work was published in [84] and can be found in Section 3.2 of this dissertation.

- **Remote rendering in virtual environments:** In order to take advantage of the scalability of the perspective shear-warp algorithm on parallel computers, it was parallelized. The parallelized version of the algorithm runs on all types of parallel computers which support MPI. The parallel computer composites the intermediate image and transfers it to the display machine in a compressed form. The warp is then done on the display machine, supported by texturing hardware if present. These achievements were published in [82] and [83], and in this dissertation they will be discussed in Chapter 5.

- **Device independent user interface for virtual environments:** There are a number of different types of output devices for virtual environments, each of which having specific requirements for its user interface. Furthermore, there are many different input devices that can be used for interaction with the environments. In order to enable the development of interactive applications running independently of both input and output devices, a new device independent widget library was designed and implemented. It was published in [30], and it will be described in Section 4.1.

- **User interface for volume rendering in virtual environments:** Virtual environments have different characteristics than desktop computers with respect to the user interface. Therefore, no existing desktop interface for volume rendering in virtual reality could be used. In particular, the transfer function editor had to be redesigned. But also a number of interaction elements that would traditionally be activated from a menu can be invoked by direct interaction methods when three dimensional input devices are available. A volume rendering user interface was developed, which exploits the specific characteristics of virtual environments, and it was evaluated in two user studies. This work was published in [85] and [108], and it can be found in Section 4.2.

- **Shear-warp with pre-integration:** Pre-integration is a technique which has previously been applied successfully to texture hardware supported volume rendering [26]. It allows more exact rendering results than the traditional linear interpolation between voxels. Pre-integration can also be applied to the shear-warp algorithm, which is described in Section 3.3. The results were published in [81].

1.3 Master's Theses and Semester Projects

In the process of the research activities for this dissertation, the author advised on the following master's theses and semester projects:

- Pablo Gußmann [33], *Erstellung eines verteilten, synchronisierten und kooperativen Renderers* (Creation of a distributed, synchronized, and cooperative renderer), Diplomarbeit (Master's Thesis), 2000
- Marc Schreier [79], *An Audio Server for Virtual Reality Applications*, Master's Thesis, 2002
- Frank Föhl [30], *Geräte- und szenengraphunabhängige grafische Benutzungselemente für Virtual Realtiy-Anwendungen* (Device and scene graph independent graphical user elements for virtual reality applications), Diplomarbeit (Master's Thesis), 2002
- Sven Wergandt [99], *Selektion, Extraktion und Aufbearbeitung von Volumendaten auf Multiprozessor-Systemen* (Selection, extraction, and processing of volume data on multi-processor systems), Studienarbeit (Semester Project), 2002

1.4 Structure

The structure of this dissertation is depicted in Figure 1.1. After an introductory section, the first chapter gives a motivation for the work that was carried out, and it summarizes its scientific contributions.

Chapter 2 reports on the current state of technology in the fields that are addressed in the subsequent chapters. This chapter addresses four major topics: volume rendering methods, interaction methods in virtual reality, parallelization and distribution methods, and visualization systems. The four main chapters of the dissertation, in which new developments are described, will represent these four topics.

Chapters 3 to 5 report on the three main areas in which research was done. In Figure 1.1, these chapters are depicted in one row to indicate that they do not depend on each other.

The results from research in the field of volume rendering techniques are presented in Chapter 3. Two volume rendering algorithms will be addressed: graphics hardware supported rendering with texture mapping and CPU based rendering using the shear-warp

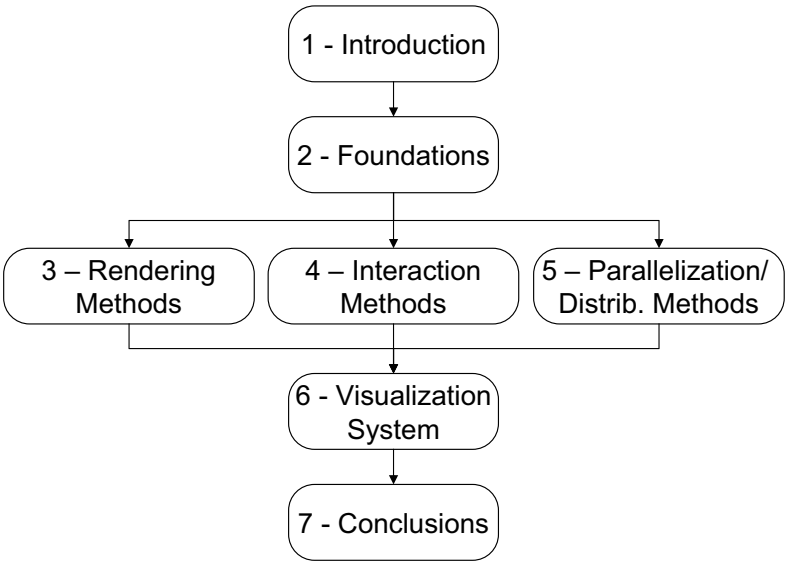


Figure 1.1: Structure of the dissertation with chapter numbers.

algorithm. Two major extensions of the shear-warp algorithm will be presented: the support of perspective projection with optimizations for virtual environments, and the enhancement of rendering quality with pre-integration.

Chapter 4 describes the interaction elements which had to be developed to allow volume rendering in interactive virtual environments. As a basis for all interaction elements, a flexible and device independent 3D widget library was created. It was used to create a volume rendering user interface, allowing a variety of functionalities like transfer function editing or clipping planes.

Chapter 5 addresses topics related to the integration of remote parallel computers into the volume rendering system. These are mainly the parallelization of the shear-warp algorithm, and the issues that arise with the transfer of rendered images from a remote parallel computer to the display computer.

All the developments from the previous three chapters were integrated into an existing visualization framework. The problems that had to be solved to achieve this and the resulting system will be described in Chapter 6.

Chapter 7 concludes the dissertation, discusses the scientific relevance of its achievements, and suggests topics for future work.

Chapter 2

Foundations

This chapter gives the information which is required for an understanding of the following chapters. Four topics will be addressed, which correspond with the four main chapters of this dissertation: volume rendering techniques, virtual reality, parallel computing, and visualization software.

2.1 Volume Rendering

Volume rendering is the technique of visualizing three dimensional arrays of data. These arrays can either be acquired by scanning real objects, or they can result from simulations or other computational methods. In the medical field, volume data is typically acquired by magnetic resonance imaging (MRI) or computed tomography (CT) scanners. MRI scanners generate data fields of specific characteristics resulting from polarizing the spin of the hydrogen electrons. CT scanners measure the x-ray absorbation of parts of the human body. CT scanners are not only used in the medical field, but also by the metal manufacturing industry, for instance to quality check parts used for the construction of machinery. Another field in which real world data is scanned is the oil and gas industry. In order to find oil reservoirs underneath the ocean, special purpose ships emit ultrasonic waves and record their reflectance patterns. This way, several terabytes of data can be acquired of a submarine region. With the latest microscope technologies, biologists can take slice images of tiny objects, or even structures inside of a cell. Some of these technologies use non-destructive techniques, so that sequences of three-dimensional datasets can be generated.

There is a large number of fields in which volume datasets are generated artificially. For instance, the pressure inside of the cylinder of a combustion engine can be simulated, which results in a volumetric array of data. Another example is a simulation of the temperature distribution in a car cabin, which allows engineers to find out how fast a car heats up on cold winter mornings, or how well the air conditioning distributes the cool air.

The visualization of volumetric data can be approached by two intrinsically different ways: either the data is converted to traditional visual elements, i.e., polygons, or the

data is displayed directly with a direct volume rendering approach. The traditional approaches of indirect volume rendering are either to extract iso-surfaces from the dataset, or to compute a cutting plane, both of which can be displayed with traditional surface rendering techniques. Cutting planes are a simple way to get an impression of a dataset, but because they are two-dimensional, it is difficult to perceive complex 3D structures with them. Iso-surfaces are typically computed with the marching cubes algorithm by Lorensen and Cline [58]. This algorithm is based on a list of all possible intersections of a plane with a volume element (voxel), which depends on its own and its neighbors' data values with respect to the iso-surface value. This approach is useful if the dataset consists of homogeneous or gradually changing data regions, for instance of temperature or pressure. However, particularly with scanned data, it is often difficult to create meaningful iso-surfaces because statistical noise and insufficient sampling rates result in iso-surfaces that contain millions of polygons. In these cases, sophisticated mesh reduction techniques are required to reduce the number of polygons, so that they can be displayed by the graphics hardware.

Direct volume rendering can create a spatial display of the dataset even if the internal structure is unknown to the user. The rendering technique is based on the assignment of color and opacity to each data value in the volume. This assignment is done with transfer functions. The design of meaningful transfer functions is crucial for the success of this method. In Figure 2.1, both types of transfer functions are depicted: the opacity function is drawn as a line, the color function is represented as a bar above the opacity function.

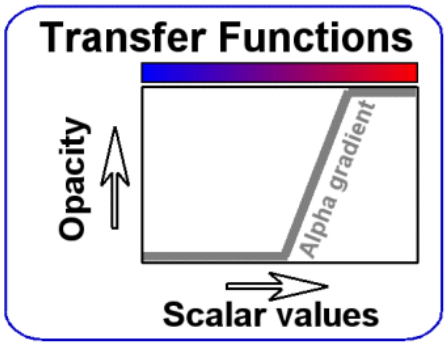


Figure 2.1: Opacity and color transfer functions.

Transfer functions can be classified according to the number of values they depend on. Typically, only the data value is used to define the transfer function [7, 35, 60], although it has long been proposed to additionally use the gradient magnitude [25, 55]. More sophisticated methods for transfer function assignments also use the second directional derivative along the gradient direction [43, 44]. A comparison of four of these methods can be found in [68].

A number of algorithms have been developed for direct volume rendering in the past. All of them share the idea of approximating the evaluation of the volume rendering integral [48]. For each viewing ray, it specifies the intensity that results from the integration of

all colors c and opacities α that the ray traverses, at all locations x on the ray up to the maximum distance D. s is the data value at the respective location:

$$I = \int_0^D \alpha(s(x))c(s(x)) \exp\left(-\int_0^x \alpha(s(x'))\mathrm{d}x'\right)\mathrm{d}x.$$

Direct volume rendering algorithms differ in the way they apply the transfer functions when evaluating the above equation. The transfer functions can either be applied before the interpolation of s from the surrounding scalar values (pre-classification) and interpolating the resulting RGBA values, or after the interpolation of s (post-classification). Only for linear transfer functions both approaches yield the same results, in the general case the results differ. Depending on the dataset, one or the other technique should be employed. If the scalar values in the dataset represent samples of a continuous scalar field, post-classification should be used. For pre-segmented datasets, in which adjacent scalar values may describe entirely separate materials, only pre-classification yields the desired results.

In the following sections, the most commonly used algorithms will be introduced: ray casting, shear-warp, splatting, texture mapping, and dedicated volume rendering hardware. A comprehensive comparison of the first four approaches can be found in [61].

2.1.1 Ray Casting

General ray casting is based on the idea of shooting rays, which originate in the user's eye, through an object, thus computing the colors of the pixels passed by the rays. For ray casting through volumetric data, each ray is traversed from the location of the eye until it leaves the dataset, evaluating the discretized volume rendering integral on the way [25, 55]. Figure 2.2 illustrates how the rays are cast from the eye through the screen and the object: dark blue voxels in the volume object are the voxels that are traversed by the algorithm. The blue pixels on the screen represent the pixels that are involved. Typically, ray casters require the volume data to be located on regular grids, because this allows a number of algorithmic optimizations. Ray casting is classified as an image order algorithm, because on the top level loop of the algorithm, it traverses the pixels of the output image.

In the past, a large number of publications have addressed optimizations of the general ray casting algorithm for volume rendering. Some of the more significant approaches will be mentioned here. Levoy [56] presented a method for empty space skipping based on a pyramid of binary volumes, also known as a complete octree. In the same publication he presented a method for early ray termination: the incremental compositing of the volume data along a ray is terminated when an empirical opacity threshold is reached. Danskin and Hanrahan [21] pre-compute multiple volume pyramids, which are basically pointerless octrees [104]. Each pyramid describes the volume dataset with respect to an attribute like average value, maximum value, or range of homogeneity.

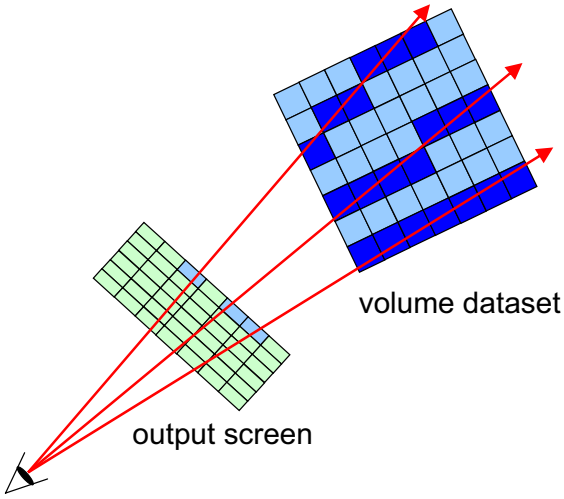


Figure 2.2: Casting rays through an object.

Yagel and Kaufman [110] presented a template-based optimization approach which takes advantage of the coherence of the rays, but it only works with orthogonal projection. Another approach was presented by Yagel and Shi [111]. They create a buffer which stores the coordinates of the first non-empty voxel visible from a pixel. This allows fast changes in material properties, and also when the viewing parameters change, these values serve as an approximation for the new rays. Freund and Sloan [31] implemented a space leaping approach which is not based on octrees, but on a distance map. For each voxel, it contains the distance to the first non-empty voxel in its vicinity.

2.1.2 Splatting

Instead of traversing the output image pixels, the splatting algorithm traverses the volume elements, making it an object order algorithm. First, the color and opacity values of a voxel are looked up in the transfer functions. Then, the location of the projection of the voxel's center on the output image is computed. After that, this pixel's color value is blended with the voxel's color value, weighted by its opacity. Finally, in order to omit holes in the image due to differences in object and image resolution, the surrounding voxels are also blended according to a previously computed footprint. The approach of pre-computing footprints in order to significantly improve rendering speed works only for the case of orthogonal projection. This algorithm was first described by Westover [103].

2.1.3 Shear-Warp

The shear-warp algorithm can only process volume data located on regular grids. It is based on the idea of factorizing the viewing matrix V into a shear S and a 2D warp component W, and doing the projection P after the shear:

$$V = W \times P \times S$$

Figure 2.3 illustrates this approach. After applying the shear matrix, the volume slices are projected and composited to a 2D sheared image. The shear step enables the algorithm to perform a ray casting in object space with high memory locality, which optimizes the usage of RAM caching mechanisms. The warp being performed in 2D space by generating the final image from the intermediate image, decreases the computational complexity considerably, compared to a 3D operation.

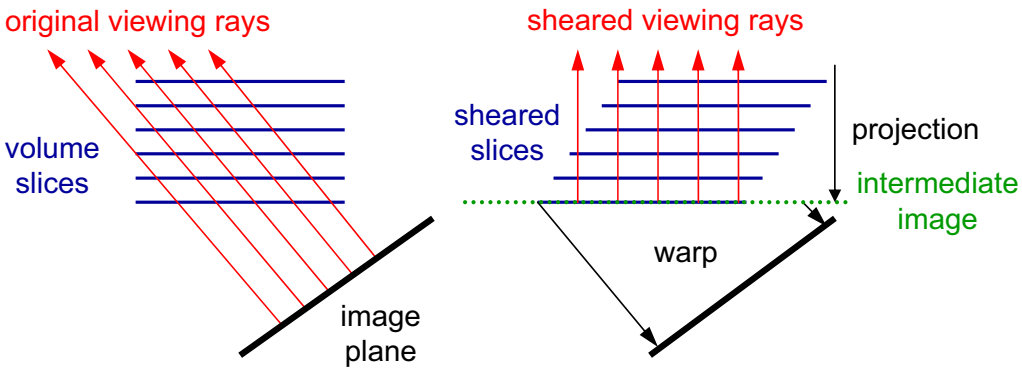


Figure 2.3: Shear-warp with orthogonal projection.

Lacroute's dissertation about the shear-warp algorithm [51] adds some ideas to further increase rendering speed. Both the volume data and the intermediate image are run length encoded (RLE) to minimize the number of memory accesses, to save storage space, and to further increase memory locality. The run length encoded volume data is stored in memory three times, once for each principal coordinate axis. Shading is performed by precomputing a normal vector for each volume element and assigning colors using a lookup table. A fast classification can be done by using an octree based algorithm instead of run length encoding, because the RLE volume has to be recomputed on every change of the transfer functions. Lacroute implemented pre-classification: the transfer functions are applied to the original scalar values, the classified values are interpolated (see Section 2.1).

Lacroute also worked on the perspective shear-warp algorithm. In his dissertation, he presented the main ideas for the transition from orthogonal to perspective projection: the volume slices do not only have to be sheared, but also scaled, before they are composited to the intermediate image. This process is illustrated in Figure 2.4.

2.1.4 Texture Mapping

Providing that the graphics hardware has built-in acceleration for 2D texture mapping, or even 3D texturing, it can be used for direct volume rendering. In both cases, the volume data is first transferred to texture memory. Then, for every frame, the volume is reconstructed with textured slices, rasterized from back to front with blending enabled.

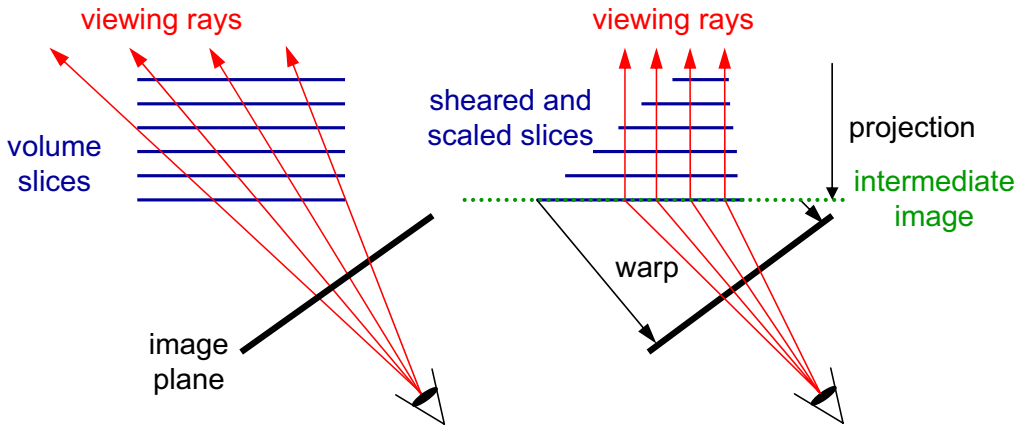
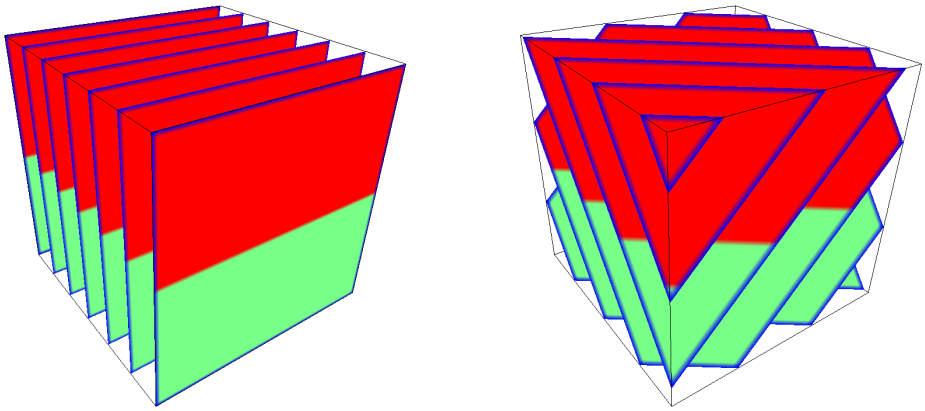


Figure 2.4: Shear-warp with perspective projection.

If only 2D texturing acceleration is available, three sets of textures have to be stored in texture memory, one for each principal axis. When a frame is rendered, it is determined which of the three sets is to be used. The slices have to be aligned with the sides of the volume dataset.



(a) (b)

Figure 2.5: Texturing modes: (a) object aligned and (b) viewport aligned slices.

When 3D texturing is available, the volume data needs to be stored in texture memory only once. Furthermore, the slices can be aligned with the viewport, which minimizes the artifacts. Figure 2.5 shows the two different slicing techniques.

The number of textured slices that can be drawn at interactive frame rates depends mainly on the number of pixels the volume occupies on screen. This is due to the pixel fill rate being the limiting factor for this technique.

Another issue with this approach is that it generates artifacts with perspective projection, because different viewing rays cover different distances between the slices. This issue can be solved by rendering concentric shells, as proposed by LaMar et al. [54] (see Figure 2.6). In their publication, they introduce a multiresolution technique, which is based on

changing the distances between the shells according to their distance to the viewer. This notion was refined and extended by Weiler et al. [97], who worked with planar slices again.

A significant image quality improvement was achieved with pre-integration, which was first published in the context of texture based volume rendering by Engel et al. [26]. This approach is based on the computation of the color of ray segments instead of single point samples on the viewing rays.

As depicted in Figure 2.7, the scalar function $s(x)$ is approximated by a piecewise linear function. The volume rendering integral for this function can be efficiently computed by one table lookup for each ray segment. The three arguments of this table lookup for the i-th ray segment from id to $(i+1)d$ are the data value at the start of the segment $s_f := s(id)$, the data value at the end of the segment $s_b := s((i+1)d)$, and the length of the segment d.

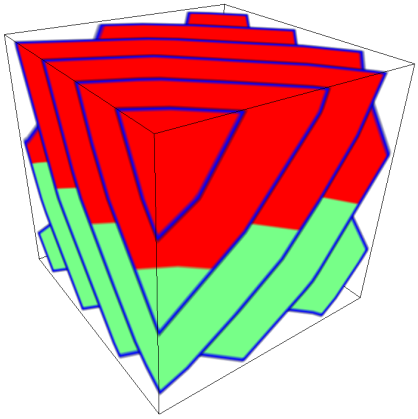


Figure 2.6: 3D texturing with concentric shells.

Pre-integrated rendering allows a better approximation of the volume rendering integral without increasing the sampling rate for nonlinear transfer functions. It improves accuracy because there is less undersampling, and it improves performance because fewer sampling operations have to be processed.

2.1.5 Special Purpose Hardware

Although a number of volume rendering hardware designs have been proposed, only a few of them have actually been implemented. Günther et al. [32] created VIRIM, which consists of four VME boards and applies a ray casting algorithm, but it supports only orthogonal projection. The VIZARD system also implements ray casting, but it is based on a PCI card with an FPGA. It was developed by Knittel and Straßer [47], and it can do perspective projection. Both of these systems were developed as prototypes.

The VolumePro card by Pfister et al. [66] is a PCI card which is commercially available. It is based on the earlier developments for Cube-4 [67]. VolumePro applies the shear-warp algorithm, and it can only do orthogonal projection. In more recent versions of

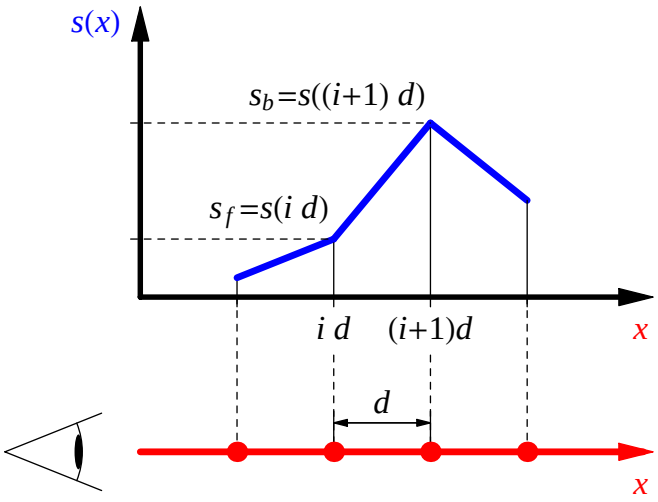


Figure 2.7: Piecewise linear interpolation of samples of $s(x)$ for pre-integrated volume rendering.

this hardware, a perspective projection mode was added. VolumePro creates only the intermediate image, the warp is done by a general purpose graphics card, which also needs to be present in the system. Due to the usage of hardware accelerated texture mapping, the warp is very fast.

None of the above hardware solutions allow the concurrent display of volume data with overlapping polygonal representations.

2.1.6 Ray Casting on Graphics Hardware

The fast development of commodity graphics hardware towards complex and fully programmable graphics processing units (GPU) allows the implementation of more and more sophisticated rendering algorithms directly on the GPU. One of the most important works in this field was presented by Purcell et al. [69]. They describe how ray tracing can be mapped to new GPU designs by taking advantage of parallel fragment units and high bandwidth to texture memory. The data stream generated during rasterization is fed to the programmable fragment processors, which process the data in parallel.

Krüger and Westermann [49] extend this idea to volume ray casting. Their implementation includes early ray termination and empty-space skipping. For datasets with either large empty or large opaque regions, which is when their optimizations can be applied efficiently, their approach is more than three times faster than traditional 3D texture mapping.

Another important and recent approach for volume rendering taking advantage of programmable graphics hardware has been presented by Röttger et al. [76]. They merge several previously published GPU-based techniques, which are based on the 3D texture

mapping approach, into one implementation: pre-integration, volumetric clipping, and advanced lighting. Furthermore, they increase the rendering quality by using the pixel shader for efficient oversampling.

The observation that the computing power of GPUs currently increases much faster than that of CPUs will most likely have a significant impact on future volume rendering algorithms. However, today most of the users of volume rendering software do not have the type of graphics hardware installed which is required for GPU-based ray casting.

2.1.7 Fourier Volume Rendering

Traditionally, volume rendering is done in Euclidean space. However, the approach introduced by Malzbender [59] and extended by Totsuka and Levoy [92] works in the frequency domain by using the Fourier projection slice theorem. This theorem is also used by Computed Tomography, where it converts the raw data from the scanning device to a volumetric data structure. Malzbender's approach inverts this process.

In a preprocessing step the volume dataset is transformed into the frequency domain. In the rendering stage this representation is sampled on a 2D slice, which is inversely Fourier transformed to result in the spatial projection of the original dataset. An advantage of this approach is that the actual rendering works on a 2D slice in frequency space and is thus much faster, as opposed to rendering a 3D data space in the traditional approach. A major disadvantage of Fourier volume rendering is that it is limited to maximum intensity projection.

2.1.8 Compression Domain Volume Rendering

Compression domain volume rendering algorithms work on volume data stored in a compressed format. Ideally the data remains compressed even in the integration process. The shear-warp algorithm with run length encoding is an example for a compression domain volume rendering approach, albeit a very simple one. Its advantage is that the compression is lossless.

A more sophisticated compression domain volume rendering approach is based on the wavelet transform of the volume data, which was first introduced by Muraki [62]. Westermann [100] extended this approach to maintain the sparse wavelet representation in the integration process. The disadvantage of the wavelet approach is that the compression is lossy, which is unacceptable for instance in the medical field.

Guthe and Straßer [34] presented an algorithm to render animated volume datasets using 3D wavelet transforms. Their approach allows to render large animated datasets on commodity PCs in real-time, which could not be displayed otherwise. However, the wavelet compression is lossy again.

Recently Schneider and Westermann [78] published a novel compression domain approach, which is based on vector quantization. The algorithm runs on current graphics chips using programmable hardware. The dataset remains compressed, even during integration. However, rendering is restricted to nearest neighbor interpolation.

2.2 Virtual Reality

In the media, the term “virtual reality” is used in many different contexts. It is applied to anything that has to do with computer generated worlds. In the context of this dissertation, the meaning of virtual reality will be limited to the abstract concept of three dimensional computer generated worlds, which are perceived by the user in true 3D with separate images for both eyes. The term “virtual environment” will be used when talking about actual implementations of 3D worlds, comprising both the virtual world generated by a computer, and the actual hardware that delivers the images. “Immersive” virtual environments make the users feel as if they were part of the virtual world. This requires screens providing a large field of view, back projection to avoid shadows, and head tracking to generate correct stereo images.

In this section, the most commonly used virtual reality input and display devices will be discussed, which are used in today’s virtual environments.

2.2.1 Input Devices

Input devices for virtual reality enable the user to navigate in a virtual world and give commands to the software. The major difference between the devices is the number of degrees of freedom, which influences how flexible the user can work with them.

2.2.1.1 Desktop Mouse

The desktop mouse consists of two or three buttons and a sensor to determine relative movement in two dimensions. Thus, it provides basically two degrees of freedom, which can be enhanced by interpreting mouse movements differently, depending on which buttons were pressed. For instance, the user could translate an object horizontally and vertically by moving the mouse while one button is pressed, and the translation could be in depth when the mouse is moved while another button is pressed. However, intuitive 3D movements are not possible with this device. In virtual reality systems, the mouse is typically used only as a backup device or for testing in the software development process.

2.2.1.2 Data Glove

Being one of the first virtual reality devices, the data glove (Figure 2.8a) soon turned out to be quite cumbersome to use, compared to its competitor, the wand. Although it theoretically allows more intuitive and precise action than many other devices, it would require force feedback to fully take advantage of its potential. Force feedback data gloves (see Figure 2.8b) have been developed, but they are even more cumbersome to use because of the additional structures that are involved.

(a)

(b)

Figure 2.8: (a) 5DT data glove (b) Immersion CyberForce.

2.2.1.3 Mechanical Tracking Systems

The SensAble Phantom (see Figure 2.9a) is both a mechanical tracking and a force feedback device. It consists of a robot-like arm structure with up to six degrees of freedom, depending on the model, and a button. While moving the Phantom's arm, the software computes the precise location and orientation of the user's hand. When used with the appropriate software, the Phantom can issue a force on the user's hand, which allows to actually touch virtual objects.

The Phantom is well suited for work at the desktop, because it provides easy access—the device only needs to be grabbed—and precise tracking. Although there is a large and expensive version of the Phantom which can be fixed to the ceiling, it is usually not used for tracking in CAVEs or in front of large projection screens which require great freedom of movement.

(a)

(b)

Figure 2.9: Mechanical tracking devices: (a) Phantom (b) Space Mouse.

The Space Mouse was originally conceived by the German Aerospace Research Establishment (DLR), and is today distributed by Logitech (see Figure 2.9b). An array of buttons attached to the device can be programmed by the software to trigger events. In contrast to the Phantom, which processes movements in absolute coordinates, the Space Mouse can only distinguish relative movements. Its advantage is that six degrees of freedom are manageable in a compact device, which only requires little space on the desk.

2.2.1.4 Electromagnetical Tracking Systems

An electromagnetical tracking system consists of movable devices which the users hold in their hands (wand) or wear on the head, and a fixed device which emits pulsating electromagnetic waves in three orthogonal directions. Sensors in the wand and in the other tracked devices receive the waves and send the measured signal strength to the tracking computer, from which it can compute location and orientation of the sensors. The most popular systems are the Ascension Flock of Birds (see Figure 2.10a) and Motionstar, and the Polhemus FASTRAK.

A disadvantage of electromagnetical tracking systems is that they are sensitive to any kind of metal that is located in the range of the emitter, because it distorts the magnetic field. Metal walls, tables, or stabilizing structures for displays may cause problems. Furthermore, there are currently no entirely wireless tracked devices for these systems. There is the wireless Motionstar, but it requires a bulky transmitter which has to be carried by the user and which has cable connections to all sensors (see Figure 2.10b). It is typically put in a backpack which the users have to put on before they can use it.

(a) (b) (c)

Figure 2.10: (a) Flock of Birds, (b) Wireless Motionstar, (c) ART hand tracker.

2.2.1.5 Optical Tracking Systems

Optical tracking systems consist of wireless tracking devices with optical markers attached and at least two cameras filming the area of interaction. The tracker by ART has a complex structure of sticks and balls attached to the tracked devices (see Figure 2.10c). The balls are made of infrared reflecting material, and they are filmed by infrared sensitive cameras with infrared lights. A computer compares the reflection patterns from all available cameras and computes the location of the tracked devices.

The great advantage of optical tracking systems is their extremely high accuracy, which is in the sub-millimeter range. Furthermore, the tracked devices are wireless. A disadvantage is that for virtual environments, a large number of cameras is required so that the sensors can always be seen from at least two cameras.

2.2.1.6 Hybrid Tracking Systems

As an example for hybrid tracking systems, the Intersense IS-600 combines gyroscopic with ultrasonic methods to track six degrees of freedom. The orientation is measured with

a gyroscope in the sensor, while the translation is measured with ultrasound, which is emitted by the tracked device and received by four receivers. By combining the information of both measurement types, a resolution up to 1.5 millimeters can be reached. The advantage of this tracking system, compared to electromagnetical tracking is that it is not influenced by metal objects. Compared to optical tracking, this system is not as exact, but more reliable when the user moves around frequently, because at least the orientation sensor does not require intervisibility.

2.2.2 Display Devices

In order to display data so that the user can perceive it three dimensionally, several techniques have been developed. Since many of them are well suited for one goal but less useful for another, a variety of devices is used in virtual reality centers. They differ mostly in the field of view occupied by the screen, the number of people that can use the system at a time, and the stereo mechanism. Two stereo mechanisms have caught on until now: active stereo by alternately displaying the frames for each eye, and passive stereo by using polarization filters to distinguish the images. There are two types of polarization: linear and circular. Linear filters need to be installed in a specific orientation, while circular filters are orientation independent. Active stereo requires high tech liquid crystal shutter glasses for every user (see Figure 2.11a), while with passive stereo the users can wear lightweight and inexpensive glasses with polarization filters (see Figure 2.11b).

(a)

(b)

Figure 2.11: Stereo glasses: (a) active, (b) passive.

2.2.2.1 Monitor

Most cathode ray tube (CRT) monitors are capable of generating active stereo images if they are driven in a way that they alternately receive images for the left and for the right eye. The user has to wear shutter glasses which need to be triggered by a signal from the graphics board. Although monitors are too small for an immersive experience and there is space for only a few people, they are a practical way for developers to view their 3D structures spatially.

Regular liquid crystal displays are too slow to do active stereo, but they can display passive stereo images when a striped mask of prisms is put in front of the screen. A particularly

sophisticated solution is the Dresden 3D display (see Figure 2.12a). It is an autostereoscopic display which is based on a TFT monitor with a prism mask. An eye tracker locates the user's eyes and computes the correct position for the prism mask which is then shifted horizontally. This technique allows the user to see 3D images without glasses, but only one user is supported at a time.

(a) (b) (c)

Figure 2.12: (a) Dresden 3D display, (b) 5DT HMD, (c) Fakespace BOOM.

2.2.2.2 Head Mounted Display

A head mounted display (HMD) is a helmet which contains two displays which project images through a system of lenses, so that the users' impression is to sit in front of a screen which is at a reasonable distance from their eyes (see Figure 2.12b). An arbitrary type of tracking sensor is attached to the helmet. Early HMDs employed CRT displays, but they were heavy and large, so that it was difficult to work with them. Today's HMDs use small and lightweight TFT displays, which allow the creation of much more wearable devices. An important issue with HMDs is their limited resolution. The highest resolution HMD made was the Sony LDI-100, but its production was discontinued. It could display 800×600 pixels on each screen.

The Fakespace BOOM (Binocular Omni-Orientation Monitor, see Figure 2.12c) is similar to an HMD, but it is not worn on the head. Instead, it is connected to an arm structure providing six degrees of freedom, with position and orientation sensors at the joints. Users can quickly look through the device whenever they need to, without the cumbersome procedure of putting a helmet on.

2.2.2.3 Workbenches

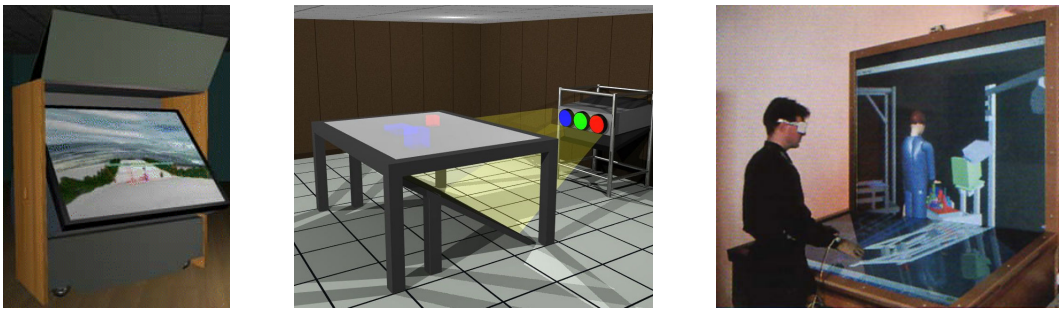
Multiple types of virtual workbenches have been developed. All of them are characterized by either CRT or TFT projectors that are mounted on the backside of a screen, so that the user cannot cast shadows.

The ImmersaDesk was developed in 1994 at the Electronic Visualization Laboratory at the University of Illinois (EVL) [20]. It is a drafting table format virtual reality display. Its

projection screen is about 1.7 by 1.3 meters wide and installed at an angle of 45 degrees. The resolution is 1024×768 pixels, which are displayed at 96 Hz.

The Responsive Workbench [50] is a device similar to the ImmersaDesk, but its screen is horizontal. Thus, it is well suited for tasks that are performed on a virtual table, for instance surgery simulations.

The TAN Holobench is an L-shaped 3D projection table with two orthogonal projection surfaces. The two screens are mounted at an angle of 90 degrees. The advantage of this system is the large field of view, which allows the users to see objects both below and in front of their eyes.



(a) (b) (c)

Figure 2.13: (a) ImmersaDesk, (b) Responsive Workbench, (c) Holobench.

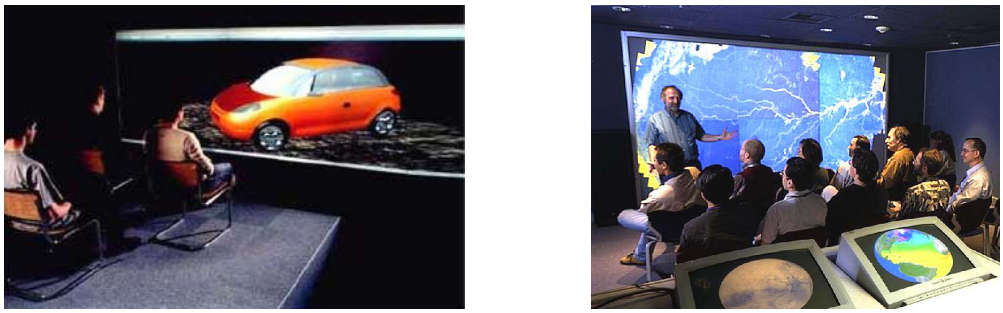
2.2.2.4 Power Wall

In general, power walls are flat screens which display images from more than one projector. The original PowerWall was developed at the University of Minnesota in 1994. Its display was a 2.4×1.8 meter flat screen, rear-projected by a 2 by 2 matrix of CRT projectors, and driven by four SGI RealityEngine2 graphics engines. Each projector provided a resolution of 1600×1200 pixels, making the entire PowerWall resolution 3200×2400 pixels.

In the automotive industry, power walls are required to display cars in their original sizes (see Figure 2.14a). A single projector would not achieve a resolution high enough for design details to be visible. In scientific visualization, the size of the screen is less important, but scientists need to see fine detail in the image. Using multiple projectors not only in a row structure, but in a 2D matrix, allows to increase the resolution while maintaining the aspect ratio. Figure 2.14b shows the 3×2 projector setup at JPL.

2.2.2.5 Curved Screen

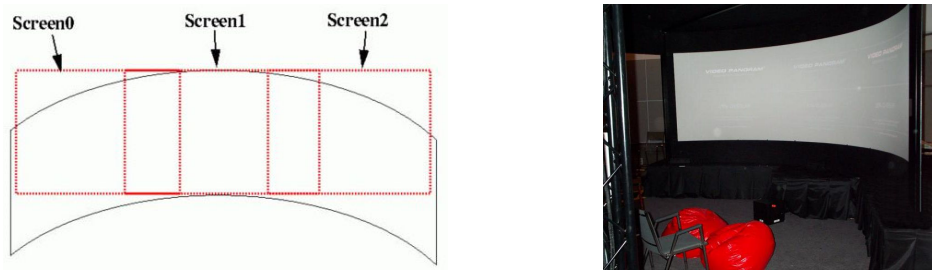
A curved screen is similar to a power wall in that the image is generated by an array of projectors, but the screen is curved instead of flat. Being curved, the field of view is greater than with a flat screen of the same size. But since projectors are made to generate



(a) (b)

Figure 2.14: Power walls: (a) wide screen, (b) high resolution.

flat images, the generation of a correct image is more difficult, and because they typically use front projection, they are less immersive. In 3D scenes, the imagery at the edges of each projected image does not fit, so the images usually overlap, and special edge blending hardware is used to produce a seamless image (see Figure 2.15a). Figure 2.15b shows the installation of the Panoram screen at EVL.



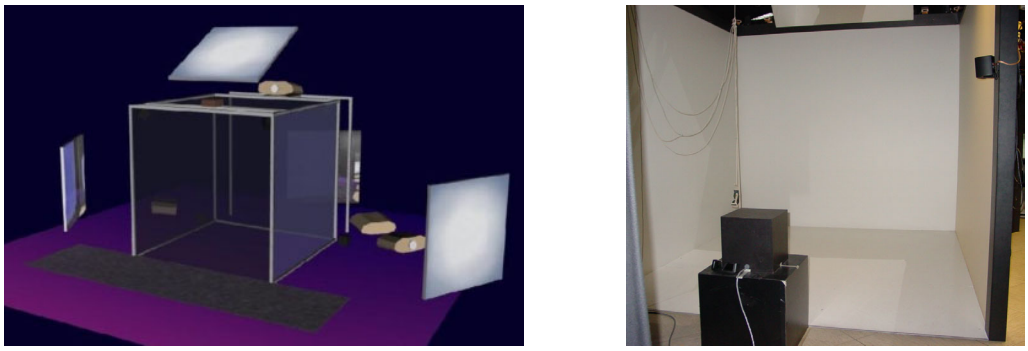
(a) (b)

Figure 2.15: Curved screen: (a) overlapping images, (b) Panoram at EVL.

2.2.2.6 CAVE

The CAVE (Cave Automatic Virtual Environment) was developed by Cruz-Neira et al. [17] at EVL. It consists of three to six projection screens which are assembled as the sides of a cube (see Figures 2.16a and 2.16b). In order to prevent black areas at the edges of the screens, the supporting structure has to be installed without obstructing the optical path of the projectors. Depending on its size, a CAVE can accommodate small groups of people, but, in contrast to power walls and curved screens, it is not suited for presentations in front of larger audiences.

The perspective of the image in the CAVE is only correct for the person with head tracking. Everybody else will see distortion at the edges of the projection screens, which also occur with flat screen devices. However, they are worse in the CAVE because straight lines deviate at the edges.



(a) (b)

Figure 2.16: (a) CAVE setup at EVL, (b) CUBE at HLRS.

2.3 Parallel Computing

Chapter 5 of this dissertation will cover parallelization issues. In the following sections, the related technical terms will be introduced and explained briefly.

2.3.1 Hardware

A classification of parallel computers based on instruction and data streams was created by Flynn [29]. He distinguishes four types of instructions-to-memory relationships: SISD, SIMD, MISD, and MIMD. SISD (single instruction single data) is the typical von Neumann PC or workstation computer with one processor. SIMD (single instruction multiple data) is implemented in massively parallel vector computers, and also within current MPPs (massively parallel processors). MISD (multiple instruction single data) systems have never been built, they would be able to execute several commands at once on the same data. MIMD (multiple instruction multiple data) is used in standard parallel computers with shared memory, distributed memory, or hybrid memory structures. All the parallel computing hardware that was used in the framework of this dissertation belongs to the MIMD category.

2.3.1.1 Distributed Memory

Distributed memory parallel computers consist of a number of processors, each with its own memory. In order to access memory of other processors, messages are sent to request it. Therefore, access to memory on other nodes takes longer than to the local memory. A typical computer of this category is the Cray T3E, but also a PC cluster of single CPU nodes. In order to increase the communication speed between the nodes, high speed network connections like Myrinet are used.

2.3.1.2 Shared Memory

Shared memory parallel computers are typically limited in the number of processors, because each processor needs to be able to efficiently reach any memory location. As the number of processors grows, the complexity of the interconnection network increases exponentially. Shared memory systems can be based on uniform memory access (UMA) or non-uniform memory access (NUMA). In UMA systems, all the CPUs and the memory share one interconnection bus. NUMA systems, like the SGI Onyx and Origin series, contain multiple interconnection buses, each serving a subset of CPUs and memory. Particularly in NUMA systems it is costly to maintain cache coherence when a processor writes to a memory location, because the respective cache line has to be invalidated on all processors working with it.

2.3.1.3 Hybrid Architectures

In a cluster of nodes with multiple CPUs each node has one common memory, which is shared by all its CPUs. However, the memory of other nodes cannot be accessed directly. This category of parallel computers uses a hybrid of shared and distributed memory. The clusters combine the advantages of both architectures, which are the ability to run efficient shared memory algorithms, and the scalability of distributed memory architectures. This type of parallel computer is becoming more and more popular. The most powerful computer today is the Earth Simulator, which has a hybrid memory architecture. It consists of 640 nodes, each of which is equipped with eight vector processors.

2.3.2 Programming Models

The two fundamental memory architectures, shared and distributed memory, require specific approaches for programming. Shared memory architectures are programmed with threading concepts, distributed memory machines require a message passing concept.

2.3.2.1 Threading

A thread is a lightweight process, which is a process that shares its memory with other threads. Threading is supported by all of today's operating systems, but it is not supported by all higher level programming interfaces. OpenMP is a set of compiler directives, which provide a simple interface for typical sections in a program that are parallelizable by threading. With OpenMP, the programmer gives hints to the compiler as to which parts of a program can be parallelized. The actual parallelization is done by the compiler.

2.3.2.2 Message Passing

Communication between processes of distributed memory machines is done by passing messages over the inter-node connection network. In order to facilitate this process, the message passing interface (MPI) was developed. It is a programming interface which offers simple commands for data exchange and for synchronization. In contrast to OpenMP, with MPI the programmer needs to parallelize the code explicitly. This can be a tedious process, because debugging parallel programs is difficult.

2.4 Visualization Software

In Chapter 6, the developments in the fields of rendering algorithms, user interaction, and parallel computing will be integrated into a visualization system. For the evaluation of the new system, the following sections will provide some background information.

2.4.1 Programming Interfaces

Most visualization systems are based on an application programming interface (API) for the graphics. The simpler the API, the more portable is the visualization system, but the more difficult it is to implement sophisticated visualization techniques and interaction.

2.4.1.1 Low Level

The most basic graphics APIs are OpenGL and DirectX. OpenGL emerged from IrisGL, which was developed by SGI, and it became a standard on both Unix and Windows systems. It is basically the implementation of a complex state machine. OpenGL extensions can be defined by graphics hardware vendors to support their new features. The ongoing release of new graphics hardware led to a large number of non-standard extensions, making it difficult to write portable programs.

DirectX, which was created by Microsoft, can only be used on Windows systems. Due to frequent releases of new versions of DirectX, newer programs do not work with older versions of DirectX. However, the advantage of this approach is that the API is much less graphics hardware dependent than OpenGL, because DirectX provides software implementations for commands which cannot be executed by the hardware.

2.4.1.2 High Level

High level graphics APIs encapsulate low level APIs, and they add high level commands to facilitate typical visualization tasks. SGI's OpenGL Performer [75] is an API which was developed for visual simulation and virtual reality applications. It offers culling, pipelining, intersection testing, and scene graphs. It is based on OpenGL, but this is transparent to the programmer. In case Performer does not provide a command for a specific

OpenGL function, for instance a new extension, OpenGL commands can be mixed with Performer commands.

OpenGL Volumizer [94] was also created by SGI. It is an API for direct volume rendering that uses a tetrahedral description of the volume data, bricking, shading, shadows, and a multiresolution approach.

OpenGL Optimizer [64] is also an API by SGI which is also based on OpenGL. Its focus is on the real-time visualization of large and detailed datasets in the CAD field. It offers the parametric representation of surfaces which allows higher quality images than with polygonal representations. These surfaces are tesselated at runtime. Optimizer also offers mesh reduction and optimization methods. Between Optimizer and OpenGL lies the scene graph API Cosmo3D, which can optionally be used independently of Optimizer.

Java 3D [1] is a scene graph based API for platform independent graphics programming. Implementations are freely available for all major operating systems. Its basic functionality resembles Cosmo3D and OpenGL Performer. It is based on OpenGL or DirectX, depending on what is installed on the host system. Wakeup criteria can be attached to several types of events, for instance based on proximity or the collision of geometry. Java 3D also supports tracking devices, provided that a driver is available by the manufacturer. In addition to graphics routines, Java 3D offers a sophisticated audio engine to generate spatialized sounds. In the past, Java 3D has not only been used in desktop applications, but also in virtual environments [88] and even CAVEs [86].

Open Inventor [40] is a strictly object oriented graphics API for interactive 3D applications. It does not only provide a scene graph, but also a rich set of geometric objects, such as cubes, polygons, text, materials, cameras, lights, trackballs, handle boxes, and 3D viewers. Furthermore, it supports some simple selection and interaction techniques.

The Visualization ToolKit (VTK) is an open source, freely available graphics API for 3D computer graphics, image processing, and visualization [96]. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods, as well as polygon reduction and mesh smoothing. VTK includes a C++ class library and interface layers for Tcl/Tk, Java, and Python.

2.4.2 Visualization Frameworks

In order to solve visualization tasks in an application field, a visualization framework is required, which is an application that can be used without programming knowledge. Instead of approaching visualization from a graphics hardware point of view, visualization frameworks focus on the data representation. They can be classified by their usage models. In modular tools, the user selects reader, filter, and renderer modules and models the data path between them. Integrated tools provide all functionality within a menu driven application. They are easier to learn, but less flexible than modular tools.

2.4.2.1 Modular Tools

The majority of the available visualization frameworks is modular. In this section, the most prominent packages COVISE, AVS, Amira, and OpenDX will be presented briefly.

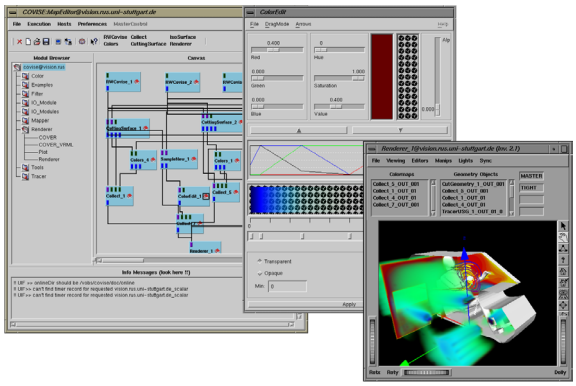
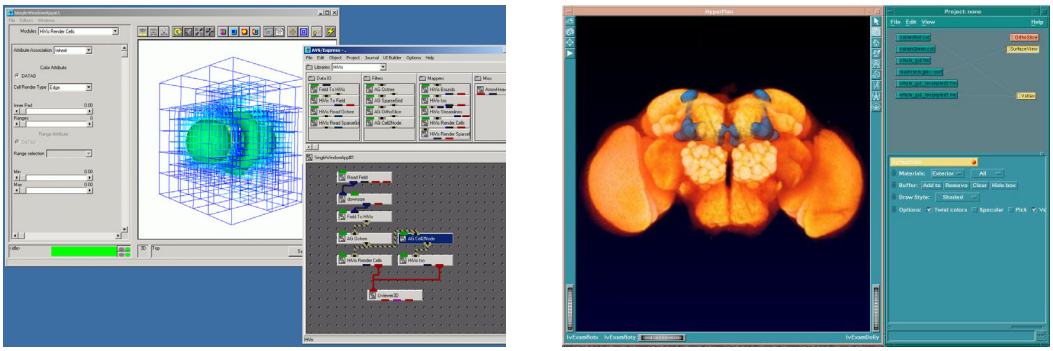


Figure 2.17: COVISE user interface with Open Inventor renderer.

COVISE [16] is a modular visualization system, which supports collaborative visualization both at the desktop and in virtual environments. It was originally developed by the visualization group of the HLRS [72, 73] and is now also commercially offered by the company VirCinity. In a visual application builder, the data flow is represented by a network of modules (see Figure 2.17). At the beginning of the data pipeline, reader modules load geometry data from disk. With this data, filter modules can compute cutting surfaces, iso surfaces, or streamlines, and sampling modules can convert grid types. At the end of the pipeline, renderer modules display the resulting data stream. Arbitrary numbers of reader, filter, and renderer modules can occur in a data flow network. In collaborative work, the modules of a network can reside on different computers to employ the computing power of remote supercomputers.



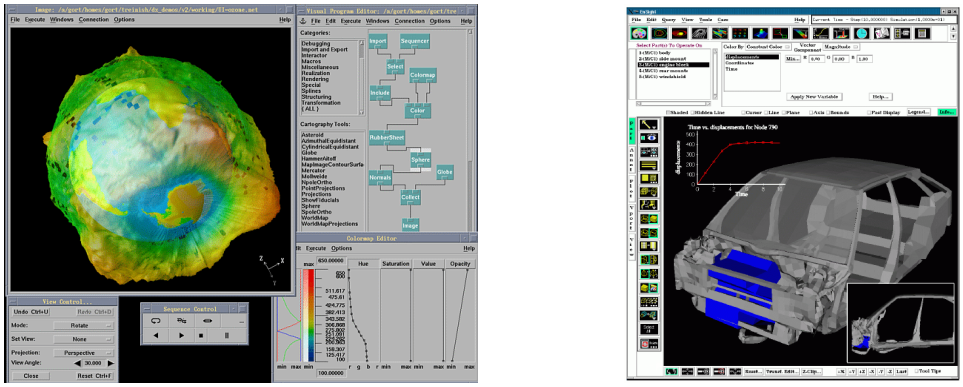
(a) (b)

Figure 2.18: (a) AVS, (b) Amira.

Two types of renderer modules are available. An Open Inventor based renderer is used for visualization at the desktop, multiple instances of it can be opened at once. The Performer

based renderer COVER was created for visualization in virtual environments. It runs from within COVISE, or it can be used as a stand-alone renderer. It supports a large number of tracking systems and screen configurations. It features a plug-in system, which is an API for creating additional functionality without recompiling the visualization framework. In COVER, interaction is based on a “laser” beam of finite length that is cast into the direction in which the input device points.

AVS [5] is a visualization system similar to COVISE. It is also based on modules and a data flow network, as displayed in the desktop layout in Figure 2.18a. However, it was not originally designed for collaborative work, and until recently, it did not support interactive work in virtual environments.



(a) (b)

Figure 2.19: (a) OpenDX, (b) EnSight.

The visualization framework Amira (see Figure 2.18b) emerged from the research activities at the Department for Scientific Visualization of the Konrad-Zuse-Zentrum for Information Technology in Berlin (ZIB). It is distributed by Template Graphics Inc. (TGS). Amira does not offer collaborative work, but it has virtual reality support, and it runs on both Unix and Windows systems. It is based on OpenGL and Open Inventor and contains algorithms for direct volume rendering.

OpenDX [63] is an open source software project, based on the Visualization Data Explorer, which was originally created by IBM. Similar to the above modular systems, it is based on a data flow network paradigm (see Figure 2.19a).

2.4.2.2 Integrated Tools

EnSight [27] is an integrated visualization system, which does not work with modules or data flow networks. Instead, it offers its functionality through menus and icons, as can be seen in Figure 2.19b. EnSight does not support collaborative work or direct volume rendering, but it can display data in virtual environments.

Chapter 3

Rendering Methods

This chapter will present the author's research in the field of rendering algorithms for virtual environments. Three major topics will be addressed: the well-known algorithm based on texturing hardware acceleration, the extension of the shear-warp algorithm to support perspective projection, and the enhancement of image quality by pre-integration within the compositing stage of the shear-warp algorithm.

3.1 Texture Hardware

The most commonly used approach for volume rendering in virtual environments is to rasterize screen aligned slices with data from the volume dataset, as outlined in Figure 3.1. The rendering of the slices can be done efficiently by taking advantage of hardware accelerated 3D texturing, as suggested by Cullip and Neumann [19]. This capability became possible with the release of SGI's RealityEngine graphics hardware, which is described in [2]. An application of this technique to medical data is presented in [12].

Today, 3D texturing acceleration is not only provided by high-end graphics computers, but also by more recent PC graphics boards. Even if only 2D texturing hardware is available, Rezk-Salama et al. [74] developed an approach to generate high quality volume images. Westermann and Ertl [101] describe improvements for texture based volume rendering.

Using 3D texturing, the programmer only needs to compute polygons in the plane of the slices, which are large enough to cover the worst case volume size at this location. The intersection of these polygons with the volumetric dataset is then done by the graphics hardware using trilinear interpolation between the volume samples.

The major drawback of this approach results from the graphics board's limited pixel fill rate, which directly affects how many textured polygons can be drawn: the larger the volume object on the screen, the fewer textured slices can be rasterized for its reconstruction in the same time. Another disadvantage of this approach is the limitation of the volume size to the size of the texture memory on the graphics hardware, which is necessary to prevent slow swapping to main memory.

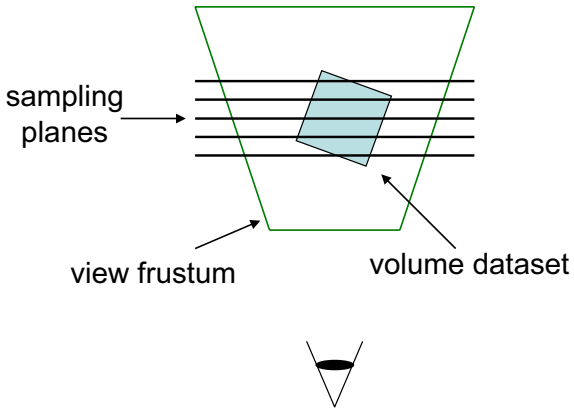


Figure 3.1: Sampling planes in texture hardware supported algorithm.

3.1.1 Optimizations for Virtual Environments

The approach to texture hardware accelerated volume rendering in virtual environments, which is used in this dissertation, is a hybrid of a slice based algorithm and a simple multiresolution technique. In virtual environments, the datasets are typically displayed in stereo at full screen resolution and at interactive frame rates. Because transferring data in and out of texture memory is expensive, the selected approach requires the dataset to fit entirely in the memory of the graphics board. With these requirements, the pixel fill rate remains the limiting factor on the frame rate.

None of the publicly available volume rendering packages could be used for the research described in this chapter, because either they did not provide multi-pipe support, or they lacked an API. For VolPack, 3DDataMaster, Amira, TeleInVivo, VoxelView and VTK this was investigated in [106]. SGI's OpenGL Volumizer could not be used because it did not provide multi-pipe support when used with Performer.

For interactive classification, the mapping from data values to RGBA (red, green, blue, alpha) values has to be done by lookup tables, or otherwise the volume data would have to be reloaded into texture memory for every change. Depending on the volume size, reloading would last up to a few seconds, due to the limited data throughput from main to texture memory, as measurements showed [95]. Two different OpenGL extensions have been created for rendering textures with indices into lookup tables. They differ in the order of interpolation and lookup (see Section 2.1): `GL_EXT_paletted_texture` uses pre-classification, while `GL_SGI_texture_color_table` implements post-classification. The application described below finds out which extension the graphics hardware supports and uses it, thus giving instant feedback to the user on transfer function modifications.

In an immersive virtual environment, the frame rate should not drop below 10 frames per second in order to retain the effect of immersion. On the other hand, more than about 25 frames per second are not required for an optimal perception of motion. Without swapping texture data and keeping only one instance of the volume dataset in memory, the simplest multiresolution technique is to draw a variable number of equidistant slices. Because the

optimal frame rate depends on both the user and the application, it can be adjusted by the user from within the virtual environment, and it will be maintained automatically by the system: after each frame, the drawing time is measured and compared to the selected value. Depending on the difference, the number of slices to rasterize in the next frame is computed.

In order to retain the correct overall volume opacity when changing the number of slices, the algorithm adapts the opacity values in all slices accordingly by using lookup tables for the opacity values and by re-computing the tables whenever the number of slices changes. The required opacity correction formula is derived in [97]:

$$\alpha_k = 1 - (1 - \alpha_0)^{\frac{\triangle_k}{\triangle_0}}$$

α_0 is the opacity as computed by the original transfer function, $\triangle_0$ is the distance between the texture elements, $\triangle_k$ is the distance between the slices, and α_k is the new opacity, which is to be used instead of α_0. This approach requires the graphics hardware to support either the paletted textures or the color index mode extension.

Another issue in virtual environments arises when the projection surface is non-planar. This is the case in CAVE-like environments, on the holobench, and on curved screens. For instance, in a CAVE the approach of screen aligned slices applies only to one screen, while they are not aligned with the other screens. In these cases, two different methods will be used to determine the normals of the textured slices, depending on the location of the user's eyes. Whenever the center of the eyes is outside of the volume boundaries, the slice normals are made parallel to a line from the volume center to the center of the eyes. If the user's eyes are inside of the volume, the normals are made parallel to the viewing direction. The experiments carried out in the framework of this research showed that this approach yields the least amount of artifacts. The theoretically optimal approach of always setting the normals parallel to the viewing direction cannot be applied. This is because the hardware usually cannot generate enough slices for an image without artifacts, and when the user's head moves about, the artifacts change for every frame. This showed to be irritating for the viewer.

3.2 The Perspective Shear-Warp Algorithm

The volume rendering approach based on hardware accelerated 3D texturing is capable of delivering high-quality images, but its limitations are particularly apparent in virtual environments. For interactive frame rates and full screen images, the artifacts become severe enough to prevent high end applications. Therefore, alternatives are required.

Lacroute [53] presented an optimized algorithm based on the shear-warp factorization, which is a very fast CPU-based volume rendering algorithm. This algorithm is described in Section 2.1.3. Although on single processor machines the shear-warp algorithm is usually slower than hardware supported solutions, the shear-warp's good scalability allows it to be competitive on multiprocessor machines [52].

As discussed in Section 3.1, the image quality of algorithms based on texture hardware acceleration is limited by the pixel fill rate. In contrast, the speed of the shear-warp algorithm does not depend on the output image size. At comparable output image quality in a 1000^2 window, the PC system used for the tests in Section 3.2.3 renders a 64^3 dataset at 2.2 frames per second using the shear-warp algorithm, compared to 1.3 frames per second with texture hardware acceleration.

Many extensions, like stereo rendering [36], parallel algorithms [52], clipping planes [112], and performance improvements [18] have been added to the shear-warp algorithm for orthogonal projection. Optimizations for the rendering of time dependent datasets by the exploitation of time-coherence were published in [4]. Several approaches to enhance the image quality have been presented in [91]. However, only a few implementations or enhancements of the shear-warp algorithm for perspective projection were reported, for instance an improvement of the warp [13]. None of them have addressed the compositing.

Perspective projection is a requirement for many applications. For instance, in radiation therapy planning, depth information is crucial, and it can only be achieved correctly by perspective projection. While images generated with "fake" stereo, i.e., drawing an object twice with a horizontal offset and a slight rotation about the vertical axis, can give some kind of 3D effect, virtual environments displaying multiple objects in different depths require perspective projection.

The following subsections will describe the application of the perspective shear-warp algorithm in a virtual environment. The mathematical foundation of the factorization will be proved and an implementation will be presented. The implementation presented below is not based on Lacroute's source code. It was done from scratch because Lacroute's code was not flexible enough to integrate the perspective extensions while sharing routines with the previous algorithm. A number of developments will be presented which were necessary for virtual environments, such as a clipping plane, adaptive rendering speed, usage of texturing hardware for the warp, and concurrent display with polygonal menu systems.

In Section 3.2.1, the mathematical background of the shear-warp algorithm will be described. Section 3.2.2 addresses specific implementation requirements for the algorithm when used in virtual environments. Section 3.2.3 provides performance numbers and a comparison of the shear-warp algorithm to texture hardware based approaches.

The research presented in this section was carried out in collaboration with Roland Niemeier and Ulrich Lang. It has been published in [84]. Roland Niemeier introduced the author to Lacroute's orthogonal and perspective projection shear-warp approaches and discussed some issues that he expected to appear in the implementation. The enhanced derivation and the implementation of the perspective algorithm, as well as all its extensions for virtual environments were carried out solely by the author.

3.2.1 The Perspective Algorithm

In this section, the factorization of the perspective viewing transformation will be reviewed in brief. It basically follows Lacroute's derivation [51]. Furthermore, the permu-

tation of projection and warp will be proved. Finally, warp performance of the orthogonal and the perspective algorithm will be compared.

3.2.1.1 Conventions

Lacroute uses four different coordinate systems in his derivation. For an easier understanding of the algorithm, six coordinate systems will be distinguished in the following derivation. They are listed in Table 3.1, which also assigns a unique single character identifier to each of them. Additionally, the coordinate systems are illustrated in Figure 3.2.

Table 3.1: Coordinate systems definition.

o	object space	actual coordinate system of the volume dataset
s	standard object space	coordinate system after permutation of object space coordinate axes
d	deformed space	3D coordinates after shear
i	intermediate image space	2D coordinates within intermediate image
w	world coordinate space	3D world coordinates
v	viewport space	2D output window coordinates

In the following subsections, transition matrices between coordinate systems carry the names of the corresponding source and destination coordinate systems. For instance, the transition from coordinate system o to w would be named M^{ow}. The inverse matrix $(M^{ow})^{-1}$ would be named M^{wo}. Vector elements are named x, y, z, w.

3.2.1.2 Prerequisites

The goal of the factorization is to obtain a shear matrix M^{oi} and a warp matrix M^{iv} so that the viewing matrix is:

$$M^{ov} = M^{iv} * M^{oi}$$

The camera parameters define the projection from world coordinates to viewport space M^{wv}, so the transformation from object space to world coordinates is:

$$M^{ow} = M^{vw} * M^{ov}$$

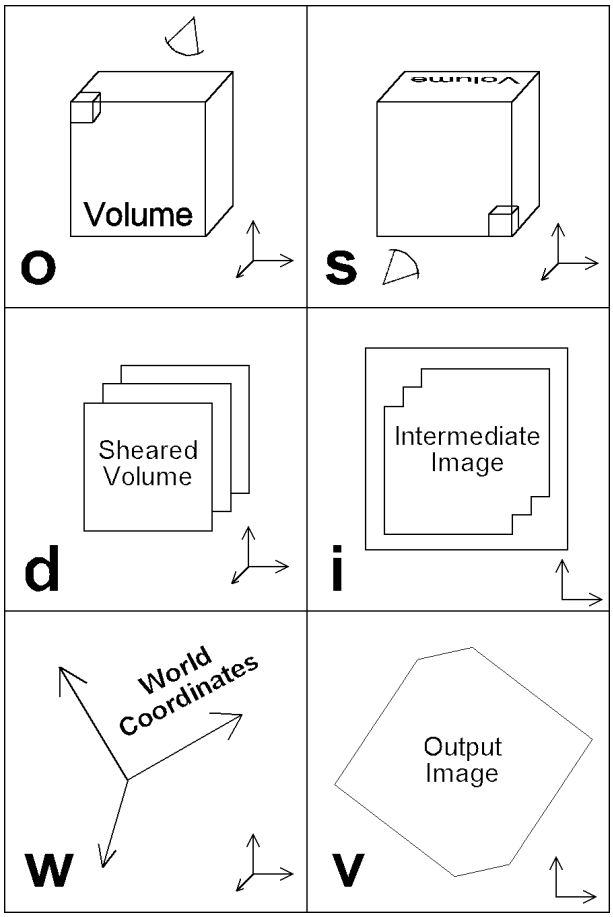


Figure 3.2: Coordinate systems illustrated.

3.2.1.3 Factorization

This section briefly explains the required computation steps for the factorization of the perspective viewing matrix.

First, the object space eye position e^o has to be found:

$$e^o = M^{wo} * \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix} \tag{3.1}$$

Then the slice order and the principal viewing axis can be determined. The principal viewing axis determines the permutation matrix M^{os}, which is needed for the adaptation of the coordinate system to the three viewing axis aligned datasets. Slice order and permutation matrix allow the compositing step to always process the slices front-to-back with memory aligned voxel data.

The eye position in standard object space is:

$$e^s = M^{os} * e^o$$

Now the shear to deformed space can be computed:

$$M^{sd} = \begin{pmatrix} 1 & 0 & -\frac{e_x^s}{e_z^s} & 0 \\ 0 & 1 & -\frac{e_y^s}{e_z^s} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{e_w^s}{e_z^s} & 1 \end{pmatrix}$$

The sheared object is scaled to the size of the intermediate image by the scaling matrix M^{scale}. The scaling factor depends on the object space and voxel space volume dimensions, and on the slice order. Section 3.2.2.1 will show how to modify this matrix to control compositing speed.

The deformed and scaled object is projected to the intermediate image by:

$$M^{di} = \begin{pmatrix} 1 & 0 & 0 & \frac{width}{2} \\ 0 & 1 & 0 & \frac{height}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Width and *height* are the dimensions of the intermediate image in pixels. The object is always projected to the middle of the intermediate image. The size of the image is made large enough to suit a reasonable viewing range. If this range is exceeded, the scaling matrix is adjusted so that the object fits.

The above leads to the overall shear matrix:

$$M^{oi} = M^{di} * M^{scale} * M^{sd} * M^{os}$$

The warp matrix follows from the goal of $M^{ov} = M^{iv} * M^{oi}$, incorporating the above equations and their inverses, respectively:

$$M^{iv} = M^{wv} * M^{ow} * M^{so} * M^{ds} * (M^{scale})^{-1} * M^{id}$$

3.2.1.4 Permutation of Projection and Warp

Although the permutation of the projection and the warp is a basic premise for the perspective projection shear-warp algorithm, it has not been proved before. The new proof, which will be derived below, computes the two viewing matrices and then compares their components.

Let P be the projection matrix:

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Let us assume that W is a general warp matrix:

$$W = \begin{pmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{pmatrix}$$

S is the shear matrix:

$$S = \begin{pmatrix} 1 & 0 & e_{x,z} & 0 \\ 0 & 1 & e_{y,z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & e_{w,z} & 1 \end{pmatrix}$$

where

$$e_{x,z} = -\frac{e_x^o}{e_z^o}; e_{y,z} = -\frac{e_y^o}{e_z^o}; e_{w,z} = -\frac{e_w^o}{e_z^o}$$

The viewing matrix is:

$$M^{ov} = P * W * S \tag{3.2}$$

For the shear-warp algorithm, the following matrix is used as the viewing matrix, applying the projection before the warp:

$$V = W * P * S$$

Comparing the substantial elements of the matrices M^{ov} and V results in potential differences only in the first and second row of the third column:

$$M_{02}^{ov} = w_{00} * e_{x,z} + w_{01} * e_{y,z} + w_{02} + w_{03} * e_{w,z}$$

$$M_{12}^{ov} = w_{10} * e_{x,z} + w_{11} * e_{y,z} + w_{12} + w_{13} * e_{w,z}$$

and

$$V_{02} = w_{00} * e_{x,z} + w_{01} * e_{y,z} + w_{03} * e_{w,z}$$

$$V_{12} = w_{10} * e_{x,z} + w_{11} * e_{y,z} + w_{13} * e_{w,z}$$

In order to lead to identical results, it is sufficient that

$$w_{02} = 0 \tag{3.3}$$

$$w_{12} = 0 \tag{3.4}$$

where from (3.2):

$$w_{02} = M_{00}^{ov} * e_{x,z} + M_{01}^{ov} * e_{y,z} + M_{02}^{ov} + M_{03}^{ov} * e_{w,z} \tag{3.5}$$

$$w_{12} = M_{10}^{ov} * e_{x,z} + M_{11}^{ov} * e_{y,z} + M_{12}^{ov} + M_{13}^{ov} * e_{w,z} \tag{3.6}$$

Multiplying (3.5) and (3.6) by e_z^o gives:

$$w_{02} * e_z^o = M_{00}^{ov} * e_x^o + M_{01}^{ov} * e_y^o + M_{02}^{ov} * e_z^o + M_{03}^{ov} * e_w^o$$

$$w_{12} * e_z^o = M_{10}^{ov} * e_x^o + M_{11}^{ov} * e_y^o + M_{12}^{ov} * e_z^o + M_{13}^{ov} * e_w^o$$

Because of (3.1), multiplied by M^{ow} from the left side, it follows that:

$$M_{00}^{ov} * e_x^o + M_{01}^{ov} * e_y^o + M_{02}^{ov} * e_z^o + M_{03}^{ov} * e_w^o = 0$$

$$M_{10}^{ov} * e_x^o + M_{11}^{ov} * e_y^o + M_{12}^{ov} * e_z^o + M_{13}^{ov} * e_w^o = 0$$

independently of e_z^o, which proves (3.3) and (3.4).

Therefore, the projection and the warp matrices can be permuted.

3.2.1.5 Warp Complexity Comparison

Both the perspective and the orthogonal projection shear-warp algorithms spend most of their time with compositing and warp. The slightly greater number of matrix computations for the factorization in the perspective algorithm can be neglected.

In the case of orthogonal projection, the warp is an affine operation compared to the perspective projection warp, in which it is non-affine.

Let W_{par} describe the general orthogonal projection warp matrix. Constant elements are listed as their values, a_{rc} represents variable elements:

$$W_{par} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{pmatrix}$$

Multiplying W_{par} by a vector $(x,y,1)^T$ requires 4 multiplications and 4 additions, accumulating to 8 floating point operations.

W_{per} describes the general perspective projection warp matrix:

$$W_{per} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix}$$

In this case a multiplication with a vector $(x,y,1)^T$ requires 6 multiplications, 6 additions, and 2 divisions, which accumulate to 14 floating point operations.

From this it follows that the perspective warp takes almost twice as long as the orthogonal warp on a system which has equal execution times for the above mentioned operations.

3.2.2 Algorithmic Issues

The application of the perspective shear-warp in virtual environments raises several issues which had to be solved. They will be addressed in the following subsections.

Just as the algorithm for orthogonal projection, the new algorithm for perspective projection uses pre-classification. Three stacks of slices are stored in memory, permuted according to the coordinate axes. The new algorithm does not use run length encoding because the typical use of virtual environments is to explore previously unknown datasets, which involves frequent changes of the transfer functions. Changing a transfer function of a run length encoded dataset requires the recomputation of all three representations of the volume in memory, which would interrupt the workflow and thus break the effect of immersion. With non-RLE datasets the transfer functions can be changed without an additional performance hit.

3.2.2.1 Compositing

Keeping the frame rate close to constant is one of the requirements to establish and to sustain immersion in a virtual environment. For the same reasons, it is crucial for the frame rate not to drop below a certain value, which is usually about 10 frames per second, depending on the application. For the shear-warp algorithm, one way to increase rendering speed is to reduce the sampling rate.

When using texture hardware accelerated volume rendering techniques, a constant frame rate can be accomplished by reducing the number of textures drawn (see Section 3.1), which leads to a reduction of the sampling rate in one dimension. The opacity has to be corrected to take the different numbers of volume slices into account.

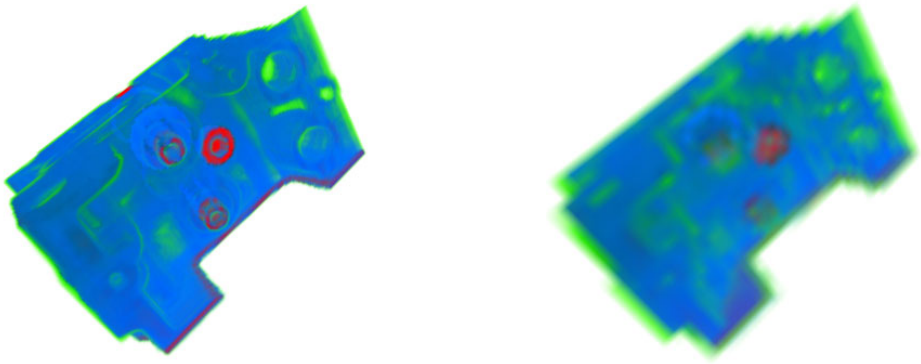
Using the shear-warp algorithm, the following approaches can be applied to increase rendering speed by a reduced sampling rate:

- *Reduction of the number of slices drawn.* In the compositing step, a certain number of slices are skipped, just as in the above described texture based approach. Also, the opacity values need to be corrected, which does not even slow down the rendering process, since the shear-warp algorithm already uses a lookup table for the mapping of RGBA to scalar data values. The disadvantage is that the voxels of the skipped slices do not contribute to the final image. Furthermore, step-by-step changes in the number of slices drawn are irritating to the user.

- *Reduction of the intermediate image size.* Drawing a smaller intermediate image than the optimal 1:1 pixel to voxel ratio for the first slice requires less voxels to be composited, so that rendering speed increases. The resulting image looks blurred due to the reduced intermediate image size, but because of a footprint-based interpolation no original data values are ignored.

Due to its smooth variability in the domain of image pixels, the second solution for runtime frame rate adaption was implemented. Using this technique, there are no sudden changes in image quality which could affect immersion. Furthermore, if permitted by the rendering hardware, rendering quality can arbitrarily be increased by enlarging the intermediate image. Figure 3.3 shows the effect of different intermediate image sizes on the $128 \times 128 \times 55$ UNC Engine dataset [24].

Algorithmically, the adaption was implemented by modifying the parameters of the matrix M^{scale} (see section 3.2.1.3), thus directly affecting the size of the intermediate image. The fact that also a magnification of the intermediate image is allowed requires the compositing to provide not only footprint based resampling, but also bilinear resampling for the case that there are multiple pixels to be drawn for each voxel in a slice.



(a) (b)

Figure 3.3: Intermediate image size: (a) 2048^2 and (b) 256^2. (See also Color Plate 1 on page 123.)

3.2.2.2 Warp

Section 3.2.1.5 derived the higher complexity of the perspective warp compared to the orthogonal warp. Since the warp matrix is not affine in the case of perspective projection, the warp accounts for a more substantial part of the total rendering time, compared to the orthogonal warp.

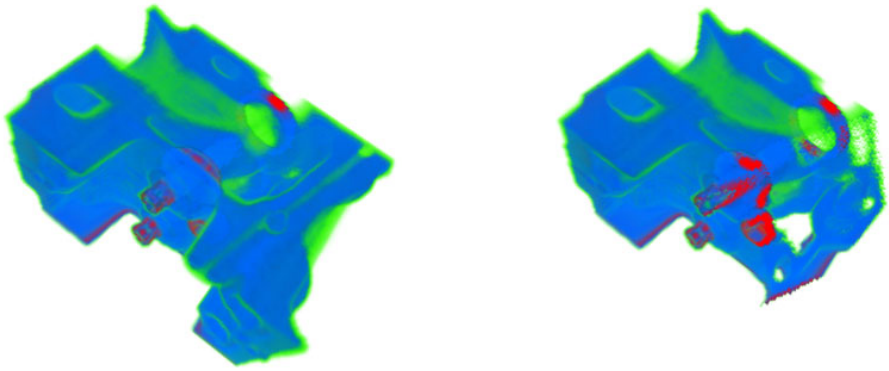
Considering that the warp applies a transformation matrix to 2D data, it can be done by 2D texturing hardware, just as the orthogonal projection warp is carried out by Pfister's VolumePro board [66]: the OpenGL model/view matrix is set to the warp matrix, and the OpenGL projection matrix is set to the identity matrix. In this case the warp matrix is not inverted, while the software warp uses its inverse to get pixel values from the intermediate image. The texturing hardware can perform the warp very fast, bilinear interpolation is added at no cost, and the final image size practically does not affect rendering speed. Furthermore, only the intermediate image has to be transferred to the graphics hardware, instead of the final image, which usually is the larger one for applications in virtual environments. On a typical SGI Onyx2 system the rendering time share of the warp is less than 2% using this method (see Section 3.2.3), so that the warp time can be neglected when determining overall rendering speed.

Using texture mapping hardware for the warp does not break with the idea of a software based rendering algorithm. The advantages of the shear-warp algorithm, like simple parallelization and limitation of the volume size to fit to main memory instead of graphics hardware memory, still persist.

3.2.2.3 Clipping Plane

For users working with volume rendering, it is useful to have one or more arbitrarily located clipping planes to look inside of objects, if adjusting the opacity transfer function does not suffice. Texturing hardware based volume rendering makes use of hardware accelerated clipping planes provided by OpenGL.

Shear-warp based algorithms cannot make use of the OpenGL clipping planes because they create a 2D image with no depth information. Thus, the clipping planes have to be introduced in the compositing step. Yen et al. [112] extract thin slabs out of the volume, but the core of their approach can be applied to arbitrarily oriented clipping planes similarly: the compositing loops have to be limited to the intersections with the clipping plane. This technique can be applied similarly to both the orthogonal and the perspective projection algorithm. For an example, see Figure 3.4.



(a) (b)

Figure 3.4: Engine dataset: (a) complete, (b) clipped. (See also Color Plate 2 on page 123.)

3.2.2.4 Viewing Angle

Lacroute found that if there is more than one principal viewing axis, the volume has to be subdivided into up to six separately rendered pyramidal sub-volumes. This is the greatest disadvantage of the perspective projection shear-warp algorithm. It would impose a significant performance degradation on the implementation described above, because several sub-volumes would have to be rendered and assembled seamlessly.

This issue was examined for the special case of a CAVE-like environment. Due to the specific geometry of the setup, all viewing rays deviate less than 90 degrees from the corresponding projection axis (see Figure 3.5). The case of approaching 90 degrees, when image distortion would become a problem, is the case of being very close to a wall. This case does not occur in typical CAVE situations with a tracked presenter surrounded by a non-tracked audience. Coming close to a wall is typically correlated with a viewing direction nearly perpendicular to the wall. As also discussed in the region of interest related literature [102], the edges of the field of view outside of the region of interest can be displayed rather coarsely.

3.2.2.5 Viewpoint

In a virtual environment, the users need to be free in choosing their position to look at the scene, and they can always only see things which are located within the viewing frustum.

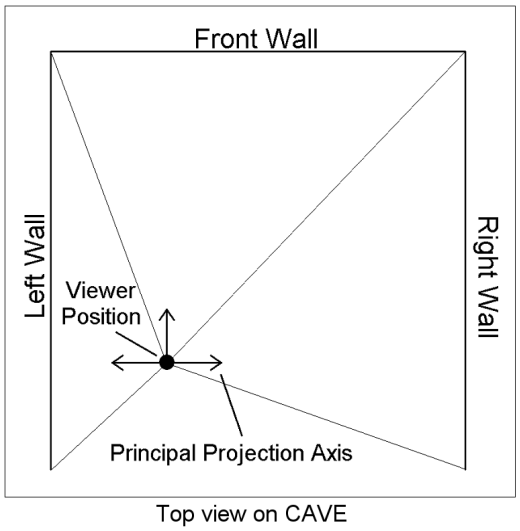


Figure 3.5: Multiple principal viewing axes.

For the volume location, three cases can occur, which are depicted in Figure 3.6:

1. The volume is located entirely in front of the viewer.
2. The volume is located entirely behind the viewer.
3. A part of the volume is located in front of and another part is behind the viewer.

In order to find the appropriate case, a bounding box check needs to be done on the volume boundaries. In the first case, no further action is necessary, because the viewing frustum clipping is done by the 2D warp. In the second case, the volume is simply not drawn at all. In order to deal with the third case, a clipping plane was set directly in front of the user's eye point, at the location of the viewing frustum's near plane, with its normal facing to the user. Thus, the user can see the dataset from the inside.

3.2.2.6 Concurrent Display of Polygons

The current implementation of the perspective projection algorithm allows two techniques to warp the intermediate image to the screen:

- Computation of the final 2D image using a CPU-based algorithm and pasting it to the viewport.
- Usage of 2D texturing hardware.

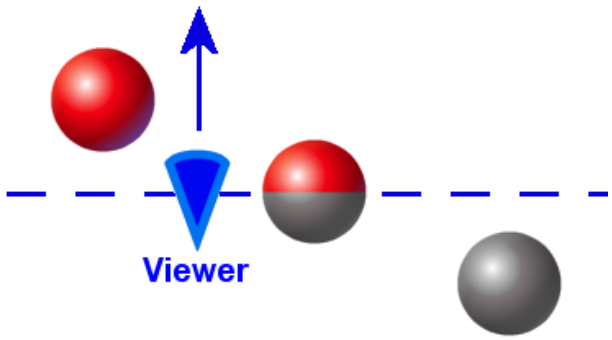


Figure 3.6: Viewpoint-object relations.

The first alternative does not allow for automatic object ordering; the programmer can only choose to draw the final image at a certain point of time during rendering of the scene. Since depth sorting the polygons is usually prohibitive due to their large number, the programmer's only choice is to draw the volume before or after the polygons.

The second alternative provides a reasonable solution automatically because the warp matrix transforms the intermediate image into 3D space, which corresponds roughly with the correct volume position. Thus, due to the Z-buffer, the hardware draws the scene's polygons at the correct depth, relative to the volume. The result is incorrect only for polygons that intersect the volume.

3.2.3 Results

For the measurements of computation times, the following two systems were used:

- PC: a single processor PC with a 1.4 GHz Pentium 4 and a 3Dlabs Wildcat II 5110 graphics board.
- Onyx2: an SGI Onyx2 with four IR2 pipes and 16 R10000/195 MHz processors.

The tests used only one processor and were done on the monitor in mono mode. In all tests, bilinear interpolation was used. The dataset was rendered in RGB colors, the alpha values were defined by a linearly increasing opacity ramp.

3.2.3.1 Rendering Time

Figure 3.7 shows the rendering times (compositing and warp) for different intermediate image sizes using the adaptive perspective projection algorithm as described in Section 3.2.2.1. This test was done on the PC with an output image size of 300^2 pixels and the $64 \times 64 \times 27$ voxels Engine dataset [24]. The measurement curve skips at intermediate image edge lengths of 512 and 1024 pixels because the texturing hardware, which is used for the warp, only works with image sizes that are powers of two.

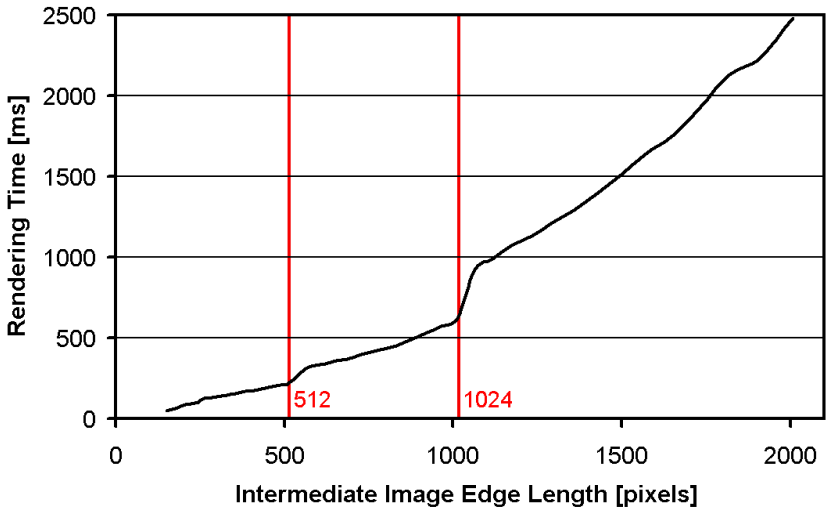


Figure 3.7: Rendering speed relative to intermediate image size.

3.2.3.2 Perspective vs. Orthogonal Projection

Table 3.2 shows the ratio between the computation times of the perspective and the orthogonal projection algorithm. The output window size was 300^2 pixels, and the texture based warp was used. Again, the Engine dataset was rendered. It can be seen that, in this example, perspective projection is about 45% slower than orthogonal projection.

Table 3.2: Perspective vs. orthogonal projection.

	PC	Onyx2
Perspective : orthogonal projection	1.43	1.46

3.2.3.3 Software vs. Texture Based Warp

In Table 3.3 the computation times of the two warp implementations for the perspective algorithm are given for different output image sizes of 256^2 and 512^2 pixels. The intermediate image was always 1024^2 pixels. On each architecture, the texture based warp speed almost does not depend on the output image size. However, because the 512^2 pixels image contains four times the amount of pixels compared to the smaller image, the software based warp takes four times as much time.

3.2.3.4 Compositing vs. Warp

In Table 3.4 the fractions of compositing and warp, in relation to the total rendering time, are listed for both the software based and the texture based warp algorithm, with differently sized output images. The Stanford version of UNC's Brain dataset [23] was used

Table 3.3: Software vs. texture based warp.

Warp type	Output image size	PC [ms]	Onyx2 [ms]
Software	256^2	19.1	42.5
Software	512^2	80.1	163.9
Texture	256^2	10.5	19.0
Texture	512^2	10.6	19.4

($128 \times 128 \times 84$ voxels), and the intermediate image size was 1024^2. The table demonstrates that by using texturing hardware, the warp only accounts for about 1% of the total rendering time on both architectures, independent from the window size. In contrast, the computation time of the software based warp accounts for about 7% of the total rendering time with both architectures. The times for compositing and warp do not always sum up to 100% because rendering a frame also involves the computation of the viewing parameters, which is not listed in the table.

It is interesting to see that the fraction of the software warp with a 256^2 output image is slightly smaller for the PC than for the Onyx, but the same ratio is inverted for the texture based warp. This is because the PC's CPU is much faster than the Onyx2's, but the performance difference is smaller for texture processing.

Table 3.4: Cost of compositing and warp.

		PC		Onyx2	
Warp type	Output image size	Comp.	Warp	Comp.	Warp
Software	256^2	98.3%	1.7%	98.1%	1.9%
Software	512^2	93.2%	6.7%	93.0%	7.0%
Texture	256^2	98.9%	1.1%	99.1%	0.9%
Texture	512^2	99.0%	0.9%	99.1%	0.9%

3.3 The Pre-Integrated Shear-Warp Algorithm

Although the traditional shear-warp volume rendering algorithm achieves a high rendering performance, its image quality is usually inferior to the results of texture hardware based volume rendering. This is because in the shear-warp algorithm, the resampling in the compositing is only bilinear, and the intermediate image is usually smaller than the final image, so the warp has to enlarge it, again using bilinear interpolation.

For texture hardware accelerated volume rendering, Engel et al. [26] have presented the pre-integration approach, which considerably improves image quality. This approach is very efficient as long as the graphics hardware provides the required functionality. But the major disadvantages of the texture-based approach, as mentioned in Section 3.1, remain.

Since the shear-warp algorithm is purely CPU-based, it allows the integration of many improvements that have originally been developed for other volume rendering algorithms. Pre-integrated volume rendering provides an efficient way to interpolate in-between slices of the volume data with only little loss in rendering performance. It is based on the pre-computation of a lookup table, which supplies RGBA values (i.e., color and opacity) for every pair of scalar data values. With the help of this table, pre-integrated volume rendering can interpolate linearly between the slices, instead of assuming a constant data value, as in the original shear-warp algorithm. Thus, pre-integration achieves significantly improved results, in particular for nonlinear transfer functions.

The research presented in this section was carried out in collaboration with Martin Kraus, Ulrich Lang, and Thomas Ertl. It was published in [81].

3.3.1 Shear-Warp With Pre-Integration

The following subsections will discuss the extension of the shear-warp algorithm with pre-integration and the implementation issues that were encountered. The following topics will be addressed: slab rendering, buffer slices to avoid redundant computations, the pre-integration table lookup, opacity and color correction, and rasterization differences between the new and the standard shear-warp algorithm.

3.3.1.1 Slab Rendering

As described in Section 2.1.4, pre-integrated volume rendering computes the color of ray segments instead of single point samples on viewing rays. Thus, the variant of the shear-warp algorithm described below renders slabs between adjacent slices instead of individual slices (see Figure 3.8). The volume data is still traversed slice by slice in front-to-back order, and the slab in front of each slice is rendered. Note that there is one less slabs than there are slices in the volume.

As each slab between two slices is rendered with the help of the data values s_f and s_b on these slices, the bilinearly interpolated data values are used twice, once for each adjacent slab. Instead of computing the same bilinear interpolation for each slab, a buffer slice can be used, which will be presented next.

3.3.1.2 Buffer Slice

The buffer slice stores interpolated scalar data values of the back slice as floating point numbers. These values can be reused for the front slice of the next slab. For the implementation of the buffer slice, two approaches were implemented and compared. The first option is to store two buffer slices in memory, each with the size of the volume slices that are rendered to the intermediate image (slice-aligned buffer slices, see Figure 3.9a). Two buffer slices are required in order not to overwrite buffered values before they are needed for the pre-integration table lookup. Thus, two blocks of memory have to be allocated,

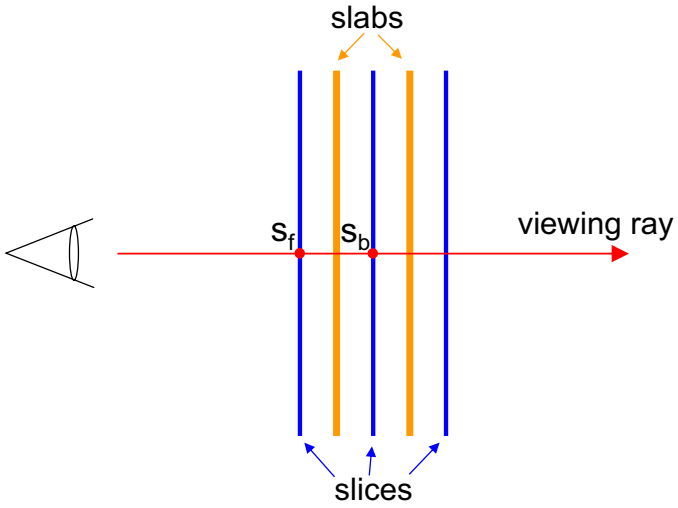


Figure 3.8: A viewing ray through the volume, traversing slices and slabs. The scalar data values of the volume dataset on the front slice and the back slice are denoted by s_f and s_b, respectively.

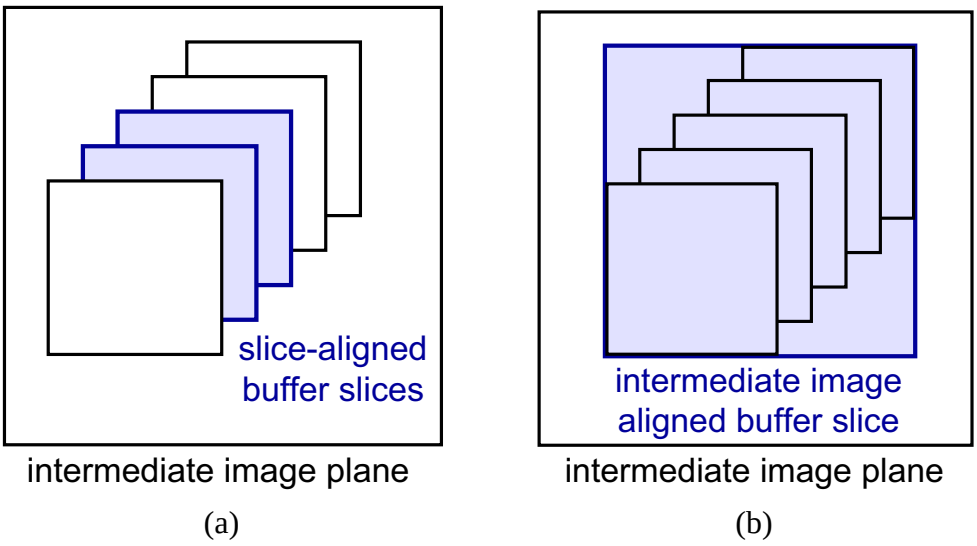
and the size of the two buffer slices has to be adapted whenever the size of the displayed slices changes. Depending on the volume size, this may happen whenever the principal viewing axis changes. Only in the case of cubic volumes the size of the buffer slices remains constant because the slices rendered to the intermediate image have the same size for each principal axis. In order to prevent allocation and de-allocation of memory whenever the principal axis changes, the memory for the buffer slices can be allocated only once and the size of the largest slice can be used.

The second approach is to create a single buffer slice, which has the same size as the intermediate image (intermediate image aligned buffer slice, see Figure 3.9b). In this case only one slice is needed because a data value is always buffered right after the data value buffered previously at the same position has been read. This approach requires to change the size of the buffer slice whenever the size of the intermediate image changes, i.e., for every change of the viewpoint. This size is easily computed because the implementation of the shear-warp algorithm presented here already uses the same idea for the allocation of the memory for the intermediate image. In order to prevent frequent memory allocation, the same approach as for the slice-aligned buffer slices can be followed by allocating memory once for the largest intermediate image size.

With the approaches described above, there is no difference in the frequency of memory allocation, but there is a difference in the size of the allocated memory. Let v_x and v_y be the width and height of the slices in voxels, respectively. Then, in the case of the orthogonal projection shear-warp algorithm, the intermediate image consists of $(2 \times v_x) \times (2 \times v_y)$ pixels in the worst case, i.e., when the viewer looks along the diagonal of the object. The intermediate image aligned buffer slice requires almost as much floating point (float) elements as there are intermediate image pixels, i.e., $(2 \times v_x) \times (2 \times v_y) = 4 \times v_x \times v_y$ floats. (Strictly speaking, it requires one row and one column less.)

The two slice-aligned buffer slices require $2 \times v_x \times v_y$ floats (again, the correct value is one row and one column less). Thus, the slice-aligned buffer slices require almost exactly half the amount of memory compared to the intermediate image aligned slice buffer.

In the case of the perspective projection shear-warp algorithm, similar considerations apply. The intermediate image aligned buffer slice can be used analogously to the case of the orthogonal projection algorithm. However, the slice-aligned buffer slices vary with the size of the volume slices that are composited to the intermediate image. By allocating buffer slice memory only for the front slice and changing the size of the buffer slice by changing its size variables, there is no memory allocation penalty to the slice-aligned buffer slice approach.



(a) (b)

Figure 3.9: (a) Slice-aligned and (b) intermediate image aligned buffer slices.

3.3.1.3 Pre-Integration Table Lookup

The bilinear interpolation that is carried out to determine the scalar data values s_f and s_b for the lookup in the pre-integration table generates floating point numbers. Thus, the lookup in the pre-integration table should bilinearly interpolate the tabulated colors and opacities. This is rather expensive, since it adds another bilinear interpolation for the composition of each voxel. Therefore, using the nearest-neighbor value in the pre-integration lookup table was tried. It turned out that for typical transfer functions, no difference is visible in the resulting images. Thus, it is generally sufficient to use nearest-neighbor interpolation for the lookup and gain a few percent of rendering speed (see Section 3.3.2).

3.3.1.4 Opacity and Color Correction

In the shear-warp algorithm, the distance between the volume samples on the viewing ray depends on the viewing direction, as shown in Figure 3.10. Larger distances between volume samples should result in higher opacities when compositing the slices: the sampled

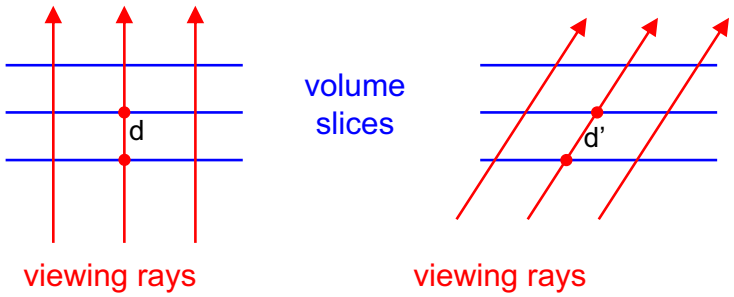


Figure 3.10: Opacity and color correction due to different viewing directions.

opacity α needs to be corrected to α'. The derivation of the new opacity can be found in [51], it results in:

$$\alpha' = 1 - (1 - \alpha)^{\frac{d'}{d}}$$

Similarly, the color components c have to be corrected. Since they are proportional to α', the correct values c' result in:

$$c' = c\frac{\alpha'}{\alpha}$$

3.3.1.5 Rasterization

A fundamental difference in rendering between the traditional approach with bilinear interpolation compared to pre-integration is the number of slices that are actually rendered: traditionally, each slice that is present in the volume dataset in the principal viewing direction is rendered. Since the pre-integration approach requires two volume slices and renders the slab in-between them, one slice less has to be rasterized with this approach. However, for typical volume sizes with dozens of slices, this effect can be neglected.

3.3.2 Results

After all the discussed improvements were integrated in the author's own implementation of the shear-warp algorithm for orthogonal projection, speed tests of the algorithm were carried out with different combinations of extensions and compared the resulting image quality.

3.3.2.1 Rendering Performance

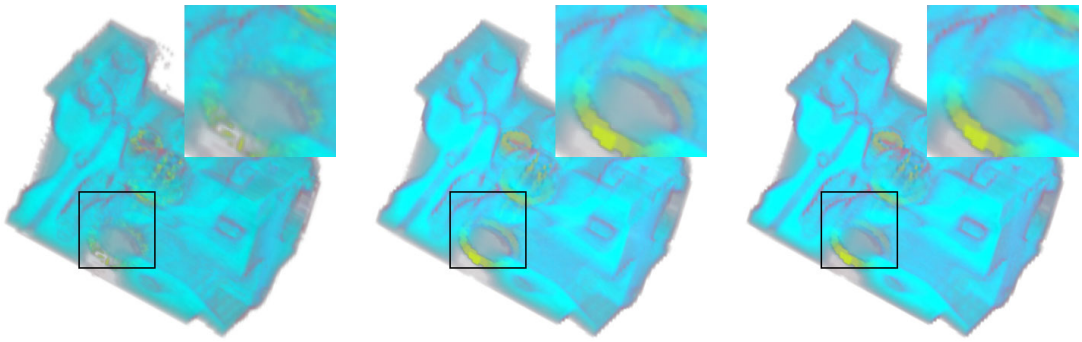
The rendering performance tests were done on a PC with a 1.7 GHz Pentium 4 processor, 256 MB RAM, and an ATI Radeon 7500 graphics card. The output image size was 512^2. The following datasets were used for the performance tests: the UNC Engine [24], Stanford's version of the UNC Brain [23], and Stefan Röttger's Bonsai tree [22]. The opacity transfer function was set to a linear ramp from zero to full opacity, which extended over the entire data range. An automatic performance measurement procedure was applied, which rotated the volume by 180 degrees in steps of 2 degrees about its vertical axis. The average rendering times per displayed frame are listed in Table 3.5.

Table 3.5: Rendering performance in seconds per frame. The abbreviated rendering parameters are: NL: nearest neighbor lookup, BL: bilinear lookup, NC: no opacity correction, OC: opacity correction.

				Pre-Integrated			
				NL		BL	
Dataset	Size [voxels]	Transparent	Standard	NC	OC	NC	OC
Engine	$128^2 \times 55$	28.0 %	0.26	0.30	0.34	0.36	0.40
Brain	$128^2 \times 84$	13.3 %	0.43	0.49	0.56	0.58	0.66
Bonsai	128^3	79.5 %	0.23	0.56	0.60	0.65	0.69

In the table, the first three columns specify the dataset, its size, and the percentage of transparent voxels it contains. The fourth column shows the performance of the standard shear-warp algorithm without pre-integration and without opacity correction. The remaining columns list the times that were achieved with different combinations of extensions. Two types of extensions are distinguished: lookup in the pre-integration table, and opacity correction (including color correction). The first two columns of the pre-integrated rendering tests are results from rendering with nearest-neighbor lookup in the pre-integration table, for the last two columns this lookup is improved by bilinear interpolation between the table values. The abbreviations used for the further classification of the table are: NL (nearest neighbor lookup), BL (bilinear lookup), OC (opacity correction enabled), and NC (no opacity correction). In all the performance tests, the intermediate image was warped by 2D texturing hardware, as mentioned in Section 3.2.2.2. Slice-aligned buffer slices are slightly faster than intermediate image aligned buffer slices because the computation of the location within the buffer slices is simpler, but the performance difference is less that 1% and the resulting images are identical. Therefore, only the numbers for slice-aligned buffer slices are shown in the table.

The times indicate that the pre-integrated shear-warp algorithm achieves a performance which is between 34% and 88% of the speed of the standard shear-warp, depending on the dataset. Pre-integration is fastest with nearest-neighbor interpolation in the pre-integration lookup table, no opacity correction, and slice-aligned buffer slices. In the performance tests, opacity correction accounts for 6-13% of the rendering time if enabled, bilinear interpolation in the pre-integration lookup table results in a 14-20% performance penalty.



(a) (b) (c)

Figure 3.11: The Engine dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction. (See also Color Plate 3 on page 124.)

3.3.2.2 Image Quality

A number of images were rendered, which result from different combinations of rendering parameters. The images were created with the same datasets and output image resolution as in the performance tests, but different transfer functions were used to emphasize the differences of the applied algorithms. The inset in the top right corner of every image shows a magnification of the region highlighted by a black square.

In Figure 3.11, the Engine dataset is depicted using three different settings. Figure 3.11a was created by the standard shear-warp algorithm without any of the extensions presented in this chapter. For Figure 3.11b, the pre-integrated rendering algorithm with nearest-neighbor interpolation in the pre-integration table and no opacity correction was used. Figure 3.11c was computed using the same settings, except that opacity correction was enabled. The difference between the standard algorithm and pre-integration is clearly visible: the engine's features are depicted much smoother and show more detail with pre-integration. The impact of opacity correction can clearly be seen by comparing Figures 3.11b and 3.11c: the semi-transparent engine block is more opaque in Figure 3.11c.

For the creation of the images of the Brain dataset in Figure 3.12, the same pre-integration settings were applied as for the Engine. Here, the subtle details on the cheek, which are enlarged in the inset, can only be seen with pre-integration. Again, opacity correction makes a difference, but due to the nature of the selected transfer function, it can not be seen as clearly as in the previous example.

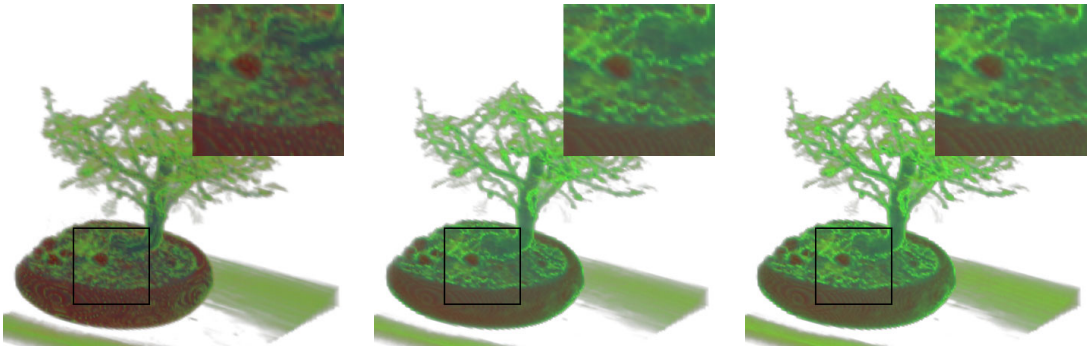
The images of Figure 3.13 depict the Bonsai dataset. They were rendered using the same pre-integration settings as before. Pre-integration accounts for significantly less staircasing artifacts on the flower pot than the standard algorithm, as can be seen very well in the inset.

In Figure 3.14, texturing hardware was employed for rendering the Bonsai dataset with the same transfer functions, the same viewpoint, and the same volume resolution as for the shear-warp. In Figure 3.14a, 128 image plane aligned textured polygons were rendered,

which is the same amount of slices as were composited for the shear-warp algorithm. The texturing hardware's capability of performing trilinear interpolation while compositing and sampling at image resolution result in a clearer image than the shear-warp can achieve even with pre-integration. However, staircasing artifacts are obvious in the resulting image. For Figure 3.14b, 256 textures were rendered. This reduces the staircasing artifacts significantly, but they are still noticeable, even more clearly than in the images rendered by the pre-integrated shear-warp algorithm. Figure 3.14c demonstrates that 1024 textured polygons result in an image of high quality.

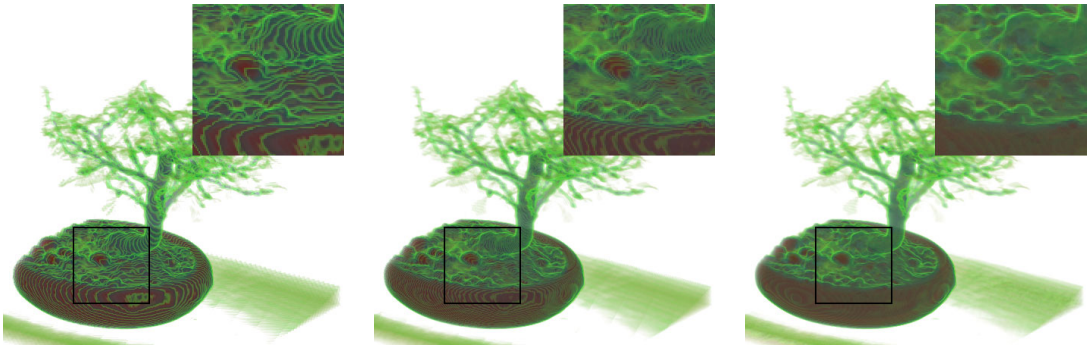
(a) (b) (c)

Figure 3.12: The Brain dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction. (See also Color Plate 4 on page 124.)



(a) (b) (c)

Figure 3.13: The Bonsai dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction. (See also Color Plate 5 on page 124.)



(a) (b) (c)

Figure 3.14: The Bonsai dataset rendered with 3D texturing hardware support using different numbers of textured polygons: (a) 128 polygons, (b) 256 polygons, (c) 1024 polygons. (See also Color Plate 6 on page 125.)

Chapter 4

Interaction Methods

This chapter describes the developments for user interaction in virtual environments. First, a new widget library for device independent virtual reality user interfaces will be presented. It allows to write applications which can be used with a variety of display devices from desktop to CAVE without the necessity of writing specific code for every device.

The second part of this chapter describes the development of new interaction elements for direct volume rendering in virtual environments, using the widget library from the first part of this chapter. It will be reported on two evaluation steps, in which user studies were carried out to find usability issues with the developed volume rendering user interface.

4.1 Device Independent VR User Interface

A user interface for virtual reality applications should have an API which allows the program developer to describe the parameters that can be changed by the user, but it should hide how the interaction elements are presented to the user, and how the user interacts with them. Furthermore, the user interface should support different types of input devices like the ones described in Section 2.2.1. They differ basically in the number of degrees of freedom and in the number of push buttons. Also, the interface should be usable with different types of display devices, which were described in Section 2.2.2, and it should take advantage of the additional space that immersive environments provide compared to single screen devices. Finally, the interface should be easy to use, preferably it should be based on what users know from their experience with other graphical user interfaces (GUIs).

The widget libraries for user interfaces which were available at the beginning of the developments either do not work in conjunction with Performer because they are designed for window systems on desktop computers (e.g., Qt [70], Java AWT [6], FLTK [28], wxWindows [109]), or they were device dependent (Virtual Tricorder [105], Studierstube [77], Virtual Palette [14], G3Menu [8], Palmtop interface [37]). For these reasons, a new library had to be developed.

4.1.1 Basis

The widget library that was created in the first step was designed for CAVE-like environments and input devices with six degrees of freedom and three buttons. At this stage, device independence was not considered, in order to be able to evaluate the general usefulness of the idea of the new GUI. The basic idea was that traditional 2D menu elements float in 3D space. The widgets were derived from desktop user interfaces as depicted in Figure 4.1: labels, push buttons, check boxes, radio buttons, and sliders. As an alternative to the slider, a rotary knob widget was created. The selection of menu entries was done with a virtual beam of limited length, which is drawn as an extension of the input device. The graphics API used was OpenGL Performer. The widget library was integrated into COVISE's virtual reality renderer COVER, which was introduced in Section 2.4.2.1. This basic virtual reality user interface (VRUI) was developed in collaboration with a colleague, Uwe Wössner.

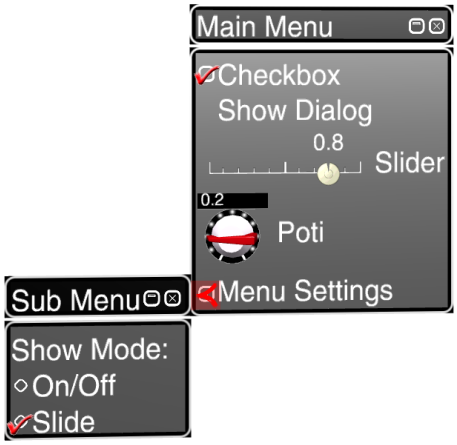


Figure 4.1: Basic menu widgets: label, push button, check box, radio button, slider, rotary knob, sub-menu.

A menu has a title bar, which allow the user to move it, to close it, or to shrink it to the title bar. Moving a menu in 3D is done similar to moving a window in a typical desktop application, which is by clicking the title bar and dragging the pointer. Depending on which button is used for clicking, the menu either always hangs down, or it is rotated with the pointer movement. Clicking it with another button and turning the input device changes the size of the menu. By default, the left button moves with gravity, the middle button scales, and the right button moves with free orientation.

Activating a menu button is done by pointing at it and clicking a button on the input device. Check boxes and radio buttons are used in the same way. Sliders are moved by clicking on the position marker and dragging the mouse from there up to the desired value.

Rotary knobs are manipulated by clicking on them and then turning the hand, similar to real knobs. There are three advantages to rotary knobs, as opposed to sliders. First, the human hand can turn much more precisely than translate. Second, rotary knobs do not have a limited value range, they can be rotated multiple times for manipulation of

unconstrained values. Third, the accuracy of the adjustment is independent of the user's distance from the knob.

4.1.2 Extensions

After the basic widget library had been created and successfully tested in VR applications, device and scenegraph independence, as well as new widgets and layout options were integrated. The resulting widget library VRUI+ was published in [30].

4.1.2.1 Scenegraph Independence

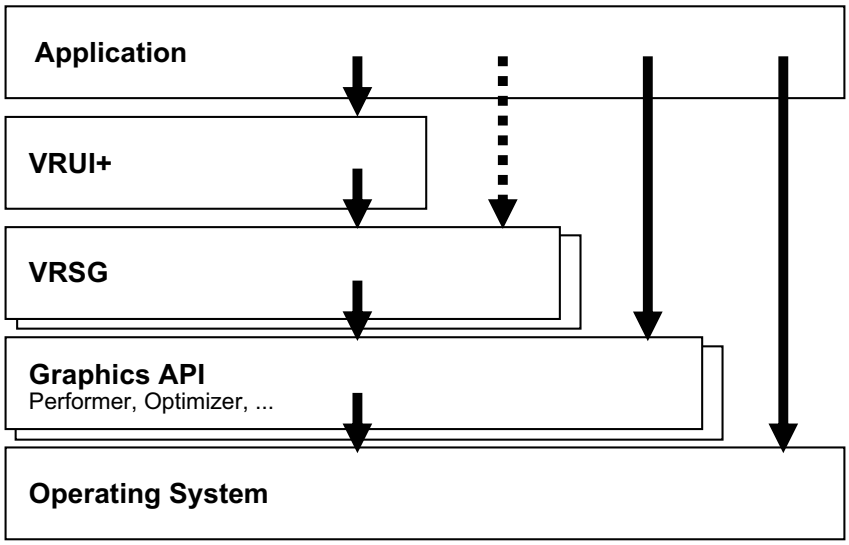


Figure 4.2: Layers of VRUI+.

The original widget library required OpenGL Performer. In order to achieve scenegraph independence, an abstraction layer had to be inserted between the widget library and the graphics API. The abstraction layer, called virtual reality scene graph (VRSG), had to provide all the functionality of the graphics API that the widget library required. Figure 4.2 shows the location of the VRSG between the widget library VRUI+ and the graphics API.

The VRSG was modeled similar to OpenGL Performer and Cosmo 3D, which is the scenegraph API of OpenGL Optimizer. This facilitates its use for Performer and Optimizer developers, and it simplifies the interface to the actual graphics API in these two cases, so that in many cases the VRSG commands can be mapped directly to the corresponding graphics API commands. The VRSG class hierarchy is depicted in Figure 4.3.

While all the other images in this dissertation show the OpenGL Performer version of the user interface, Figure 4.4 shows that the widgets look the same when Cosmo 3D is linked instead.

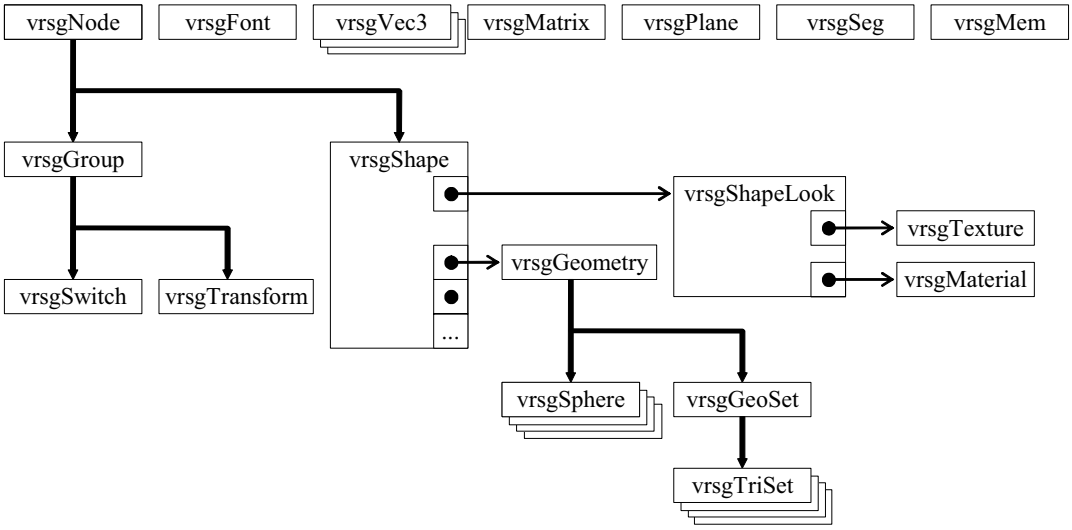


Figure 4.3: VRSG class diagram.

4.1.2.2 Device Independence

The menus of the basic widget library float in space and can be moved freely by the user. On multi-screen projection devices like CAVEs or power walls there is enough space for several menus and a dataset. However, on single screen devices the dataset and the menus often overlap because space is limited. If the dataset is located in front of a menu, the menu is hidden by it, so in order to select a menu entry the user would have to move away the dataset.

To overcome this usability issue VRUI+ makes use of the stencil buffer. The user interface and the pointing device are displayed first and they set the stencil buffer. After that the dataset is displayed with stencil buffer testing enabled. Thus, the user interface remains on top.

However, in places where the menus are visible although they are actually located behind or within the dataset, the stereo effect is significantly reduced. Therefore, a snapping mode was created in which all windows snap to the nearest edge of the screen. The stereo effect of the dataset is much less affected at this location, because it appears as if the menus were not part of the actual 3D display area.

Another difference between output devices is that on single screen displays, menus and dialog windows cannot be moved along the depth axis, but they always lie in the plane of the display screen. This restriction of the degrees of freedom was necessary to make windows appear as if they are part of the display screen border instead of the 3D scene, so that the stereo effect can be maintained. In order to ensure that they occlude each other entirely even if they contain elements that have depth like knobs or sliders (see Figure 4.6b), they are displayed at slightly different depths. The window that was clicked on last is always displayed closest to the user.

The user interface can be used with a variety of input devices. The minimum requirement is two degrees of freedom for the position and one push button, which includes the desktop

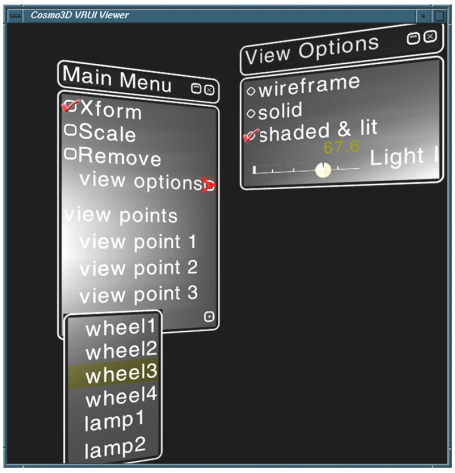
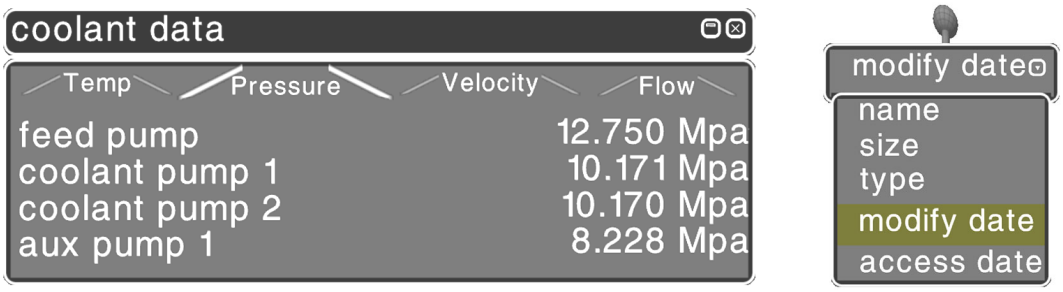


Figure 4.4: VRUI+ with Cosmo 3D.

mouse. Rotary knobs can be turned with a mouse by clicking on them and dragging the mouse up or down while keeping the button pressed. All the other widgets can be used similar to their 2D counterparts in traditional GUIs.

4.1.2.3 New Widgets and Layouts

Two additional widget types were created: a tab widget which allows to change between multiple dialog windows that are located at the same place, and a choice widget which allows the selection of one element of a set of options (see Figure 4.5).

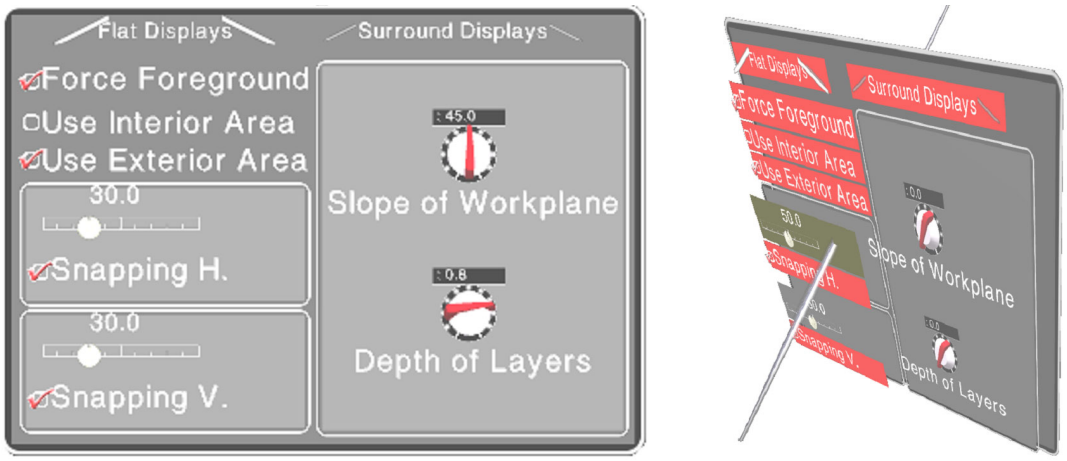


(a) (b)

Figure 4.5: New widgets: (a) tab, (b) choice.

Figure 4.6a shows a typical VRUI+ dialog window. In order to demonstrate that some window elements are true 3D, the same window is shown in Figure 4.6b after its elements have been extruded.

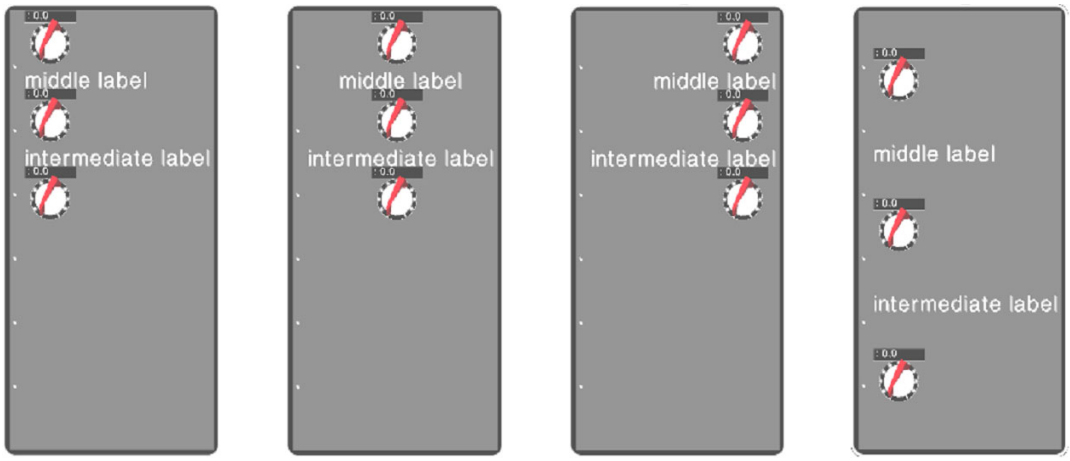
Within dialog windows, widgets can be placed in several ways. VRUI+ has only one layout manager, which arranges its elements in a row or column, similar to flow layout in Java's AWT. The layout managers can be nested to achieve the desired widget arrangement. Furthermore, the layout manager provides options for the alignment of its



(a) (b)

Figure 4.6: Dialog window: (a) front view, (b) side view, extruded.

elements, which can be left, middle, or right, and also the spacing between the elements can be changed. Figure 4.7 shows several layout options.



(a) (b) (c) (d)

Figure 4.7: Widget layout: (a) left, (b) centered, (c) right, (d) larger spacing.

4.2 Interaction Elements for Volume Rendering

The most significant previous developments of user interfaces for volume rendering in virtual environments are the Crumbs tool [11] and the StudyDesk interface [106, 107]. However, none of them allows an easy modification and display of both color and opacity transfer function, and they cannot be used in conjunction with a traditional visualization system. Therefore, a new volume rendering user interface had to be created.

A number of interaction elements are required to work with volumetric data in virtual environments. The author's most important contribution is a transfer function editor, which allows to change color and transparency of the data elements from within the virtual environment. A 3D menu offers various datasets to the user, provides control over time dependent datasets, and has options to enable or disable various rendering parameters. Furthermore, three ways of cutting off locally constrained parts of the dataset have been implemented.

After the first implementation of the user interface, a user study with external participants was carried out. After the study, the interface was improved according to the results. Then, another study was conducted to evaluate the improved system. This section will describe all of these stages in chronological order. The work presented in the remaining part of this chapter was done in collaboration with the author's colleagues Uwe Wössner and Ulrich Lang, who participated in the preparation and realization of the user studies. Uwe Wössner also helped with the implementation of interactive elements of the user interface.

4.2.1 First Approach

In the first approach, a general purpose volume rendering user interface was developed. It was created as a plug-in for the virtual reality renderer COVER. It contains a transfer function editor and a virtual reality menu to access the volume rendering options.

4.2.1.1 Transfer Function Editor

The new transfer function editor was specifically designed for virtual environments, which usually suffer from imprecise input device tracking and limited display screen resolution. The editor and all input elements are placed on a floating menu, as displayed in Figure 4.8. The right part of the image is an assembly of all possible knob layouts for different editing modes.

The new transfer function editor allows to edit one dimensional transfer functions, i.e., its color and opacity assignment depends only on the data value. The color and opacity transfer functions are modified in the function window. The data values are located horizontally, the smallest value is at the left. The transfer functions assign both a color and an opacity to each data value. The resulting RGBA values, as they are applied to the volume data, are displayed in the color bar above the function window. The alpha value can be perceived by the amount of opacity of the upper bar in comparison to the window background texture.

Changing the color value (R, G, and B components) requires setting a control point (pin) in the function window, which is done by clicking on the palette icon in the icon bar and then clicking on the desired data value position in the function window. The colored disk, which then appears at the right of the function window, reflects the HSV color model. The hue and saturation components are selected by clicking the left button on the desired

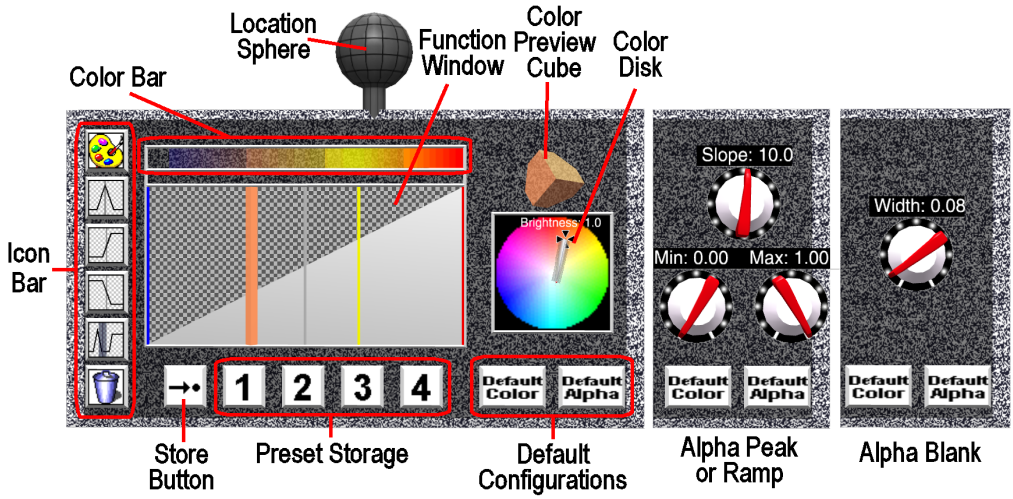


Figure 4.8: Transfer function editor.

location in the colored disk, the brightness is modified by turning the input device while pressing the middle button.

Alpha pins, which describe the opacity function, can be set independently of color pins. Setting an alpha pin requires selecting a pin of the desired type in the icon bar and dragging it to a scalar position in the function window. There are three basic types of alpha pins: peaks, ramps, and blanks. Peaks and ramps have a slope, a minimum, and a maximum value. They are used to gradually fade in or out ranges of data values. If peaks or ramps overlap, the maximum value determines the resulting alpha value. Alpha blanks have only a width, they force a scalar range to be transparent and dominate peaks and ramps.

The properties of alpha pins can either be adjusted by the rotary knobs to the right of the function window, or they can be changed by pressing the middle mouse button and turning the mouse when the pointer is in one of three areas around the pin position. The location of these areas is displayed in a horizontal bar, located between the function window and the color bar.

Both color and alpha pins can be selected by clicking next to them in the function window with the left mouse button. They can be moved horizontally when the right mouse button is pressed. If multiple pins are located closely together, repeated left mouse button clicks cycle through them. Removing a pin is done by first selecting it and then clicking the garbage can icon.

Some commonly used default color and default alpha pin configurations can be loaded by clicking the respective buttons. Multiple clicks on these buttons cycle through several default configurations: the alpha button offers ascending and descending ramps, and total opacity. The color button cycles through a set of colors from cold (blue) to warm (red), a set of all primary colors from the HSV color scheme, and a gray scale from black to white.

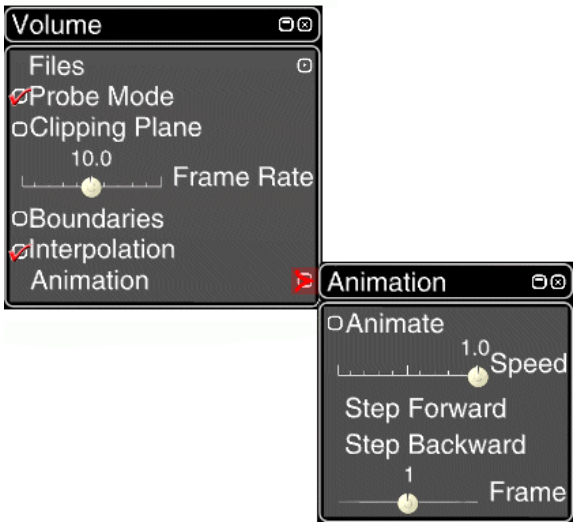


Figure 4.9: First version of the volume menu.

4.2.1.2 Volume Menu

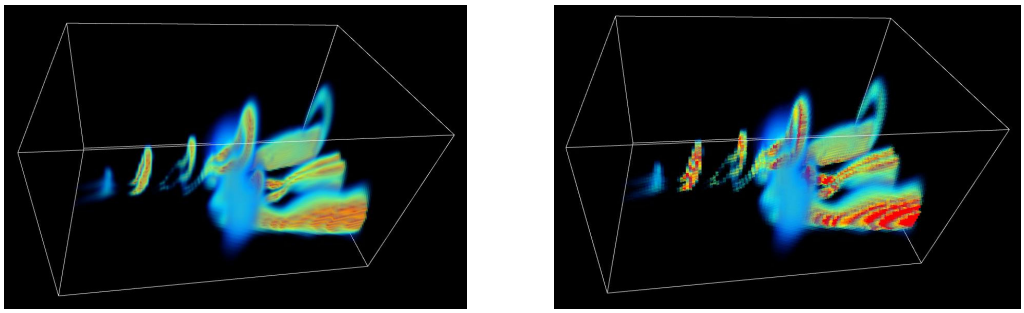
The volume menu is depicted in Figure 4.9. It can be accessed by a menu entry in a global menu, which provides mainly navigational options. The volume menu consists of the following menu entries:

- a file button which opens a list of volume files available to load.
- a toggle button to enter the probe mode.
- a toggle button to display a clipping plane.
- a slider to set the desired frame rate, which affects image quality.
- a toggle button to draw a wireframe bounding box around the dataset.
- a toggle button to turn off density interpolation (see Figure 4.10).
- an animation button which opens a sub-menu with settings for the display of time dependent datasets.

In the following sections, the more complex menu functions will be described in detail.

4.2.1.3 Image Quality

All interactive volume rendering algorithms are based on a trade-off between image quality and rendering speed. In Chapter 3, it was described how frame rate adaption can be achieved with the texture hardware based and the shear-warp algorithm. The volume rendering user interface, which is described here, offers a slider to select the frame rate, as displayed in Figure 4.9. For every frame, the system measures the rendering time and compares it to the slider value. Before the next frame is drawn, the image quality is

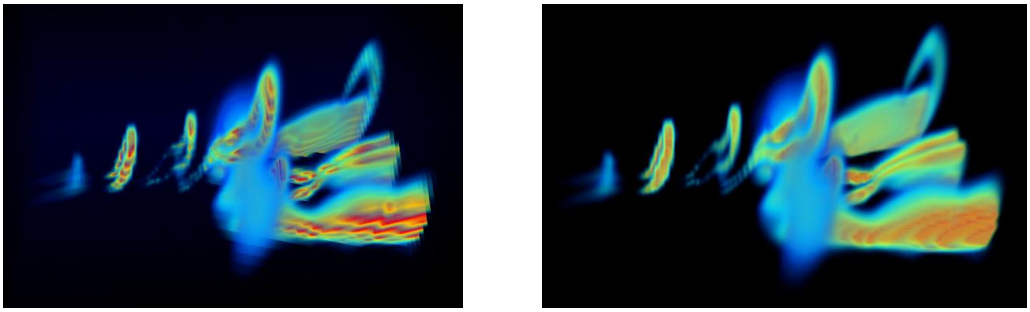


(a) (b)

Figure 4.10: Lambda dataset: (a) with tri-linear density interpolation, (b) with nearest-neighbor interpolation.

adapted so that rendering takes the desired time. Compared to a slider which influences image quality directly, the advantage of this approach is that the users will not be interrupted in their work by low frame rates without explicitly allowing this.

Whenever the users want to see a higher quality image, they can click a mouse button while the mouse is above their head to trigger the high quality mode. In Figure 4.11, the difference between the high quality mode and the adaptive work mode is depicted. The dataset used in this and some of the following images is the result of an airflow simulation generating Lambda waves, which is courtesy of Ulrich Rist from the IAG of the University of Stuttgart. In high quality mode the volume is drawn at its native resolution, usually resulting in low frame rates. Due to these low frame rates, it becomes hard to select items in the menu. Thus, in order to leave this mode and return to the adaptive mode, it suffices to press a mouse button again.



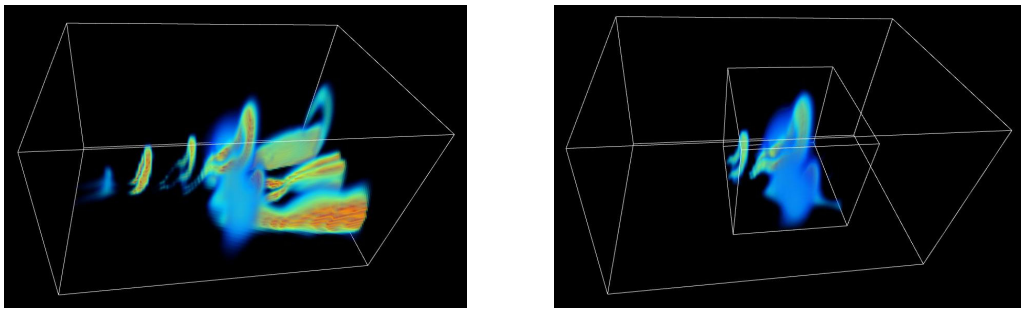
(a) (b)

Figure 4.11: Lambda dataset: (a) adaptive mode, (b) high quality mode.

4.2.1.4 Detail Probe

Motivated by the clear boxes in [11], the detail probe mode was developed (see Figure 4.12). When it is selected, the boundaries of the volume dataset are displayed and only a cubic subset of the volume is rendered. The user can drag the cube with the left mouse button, its size can be changed by turning the mouse while the middle button is pressed. The

probe can be moved around at interactive speed because the volume data is not modified, it is merely clipped at six sides.



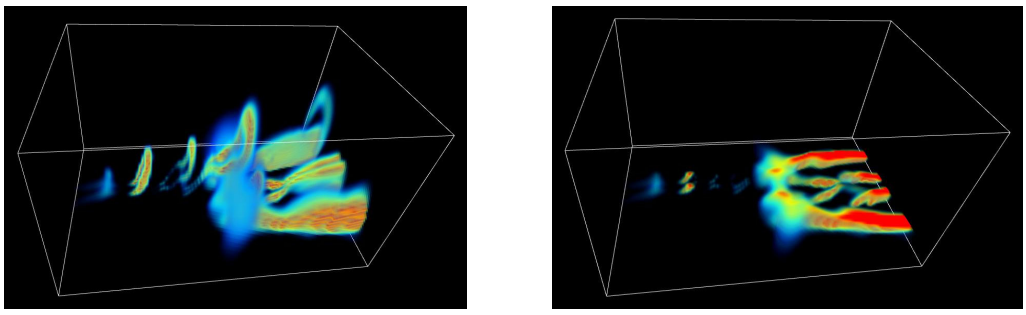
(a) (b)

Figure 4.12: Lambda dataset: (a) regular display, (b) probe mode.

4.2.1.5 Clipping Plane

In the volume menu, a clipping plane can be selected. As displayed in Figure 4.13, its effect is to cut off all volume data at one side of the plane. Clipping planes are supported by both the texture based and the shear-warp rendering algorithm. For the shear-warp algorithm, the clipping plane functionality was described in Section 3.2.2.3.

For the algorithm based on textured slices, the straightforward approach would be to use an OpenGL clipping plane. But when it is applied to volume data displayed with the slicing technique, sub-sampling artifacts occur on the clipping plane. Therefore, a new approach was developed for the clipping plane: the orientation of the volume slices is matched with the orientation of the clipping plane, and the volume is clipped by not drawing slices on one side of the plane. Furthermore, the volume slicing position is adapted so that there is always a slice drawn at the clipping plane location. This approach yields the clipping plane behavior expected by typical users of the system.



(a) (b)

Figure 4.13: Lambda dataset: (a) no clipping, (b) clipping.

4.2.1.6 Time Dependent Datasets

Time dependent datasets can either be displayed in single frame mode or as an animation with adjustable speed. The animation parameters can be set in the animation menu, as displayed in Figure 4.9. In the shear-warp algorithm, all volume data is loaded into main memory, and for every time step the pointer to the beginning of the respective dataset is adjusted accordingly.

When the texture hardware based algorithm is used, all time steps are loaded entirely into texture memory. If the texture memory is insufficient to accommodate all time steps, OpenGL automatically caches the data in main memory, slowing down animation speed considerably. Figure 4.14 shows an example animation, which is courtesy of J. Marczyk, EASi, and SGI.

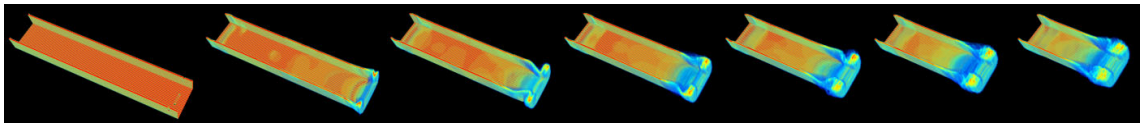


Figure 4.14: Time steps of a statistical finite elements simulation.

4.2.2 First Evaluation

In order to find out about the usability of the new interaction elements for volume rendering in virtual environments, a usability study was carried out in the CAVE at the HLRS. Derived from [90], the evaluation process consisted of two phases: a usability inspection and a scenario-based evaluation. Both phases were slightly modified to suit the requirements of the research for this dissertation.

4.2.2.1 Phase 1: Usability Inspection

In the usability inspection, a specialist from the HLRS, who was not involved in the system's development process, reported obvious flaws in the user interface design. He also made redesign suggestions to the developers, who decided on their implementation. The volume rendering system described in Section 4.2.1 already includes these changes.

4.2.2.2 Phase 2: Scenario-Based Evaluation

Before the external participants were invited, a pilot study of the system was conducted by an in-house expert. The pilot study revealed several issues, for instance with the chronology of the evaluation procedure, the number of tasks, and the understanding of the questionnaire. All of these issues were solved before the external participants were invited.

The following two scenarios were created in which the users could explore the volume rendering system:

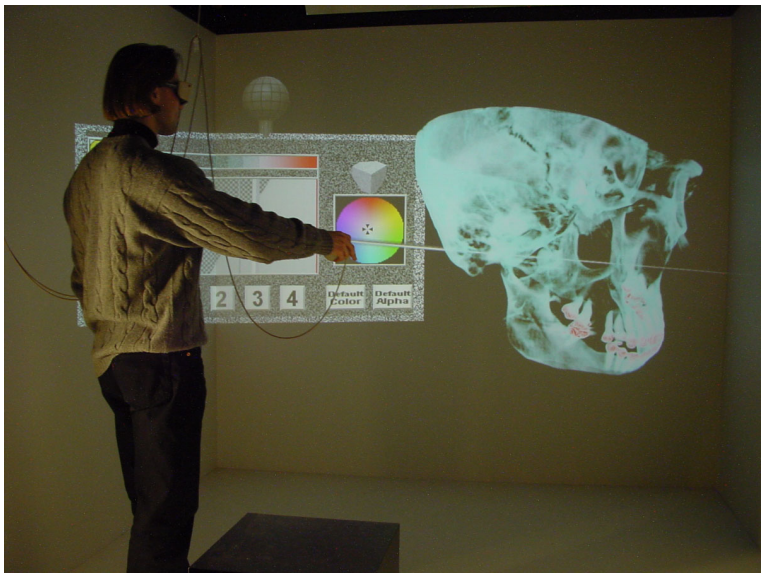


Figure 4.15: The skull of the Visible Human in the CAVE. (See also Color Plate 7 on page 125.)

- The first scenario was a downsampled $256 \times 256 \times 120$ voxels part of the skull of the male Visible Human CT dataset, which is courtesy of the National Library of Medicine [89] (see Figure 4.15). It was suggested to the participants to segment the jaw bone, the auditory canal, or the spine.
- The second scenario was a volume dataset representing the temperature distribution in the interior of a passenger car (see Figure 4.16). The car body, the seats, and the driver were rendered with polygons. It was suggested to visualize the air at a vent output or particularly hot areas.

The evaluation was carried out according to the following procedure:

1. *Self-Rating Questionnaire.* In a conference room, the users filled out a questionnaire asking them to self-rate personal skills and knowledge. The questions were based on a Likert scale [57] with a value range from 1 to 5. The users' gender, age, and physical constraints such as glasses were recorded.
2. *Introductory Presentation (Briefing).* After the questionnaire, a brief introduction to volume rendering and the CAVE was given with a Powerpoint presentation. It described the functionality of the transfer function editor in detail and finally presented two application scenarios in which the participants should work.
3. *Passive Introduction.* The users were taken to the CAVE, where three evaluators were present: M, V, and P. V and P sat outside of the virtual environment. The presenter M explained the functionalities of the CAVE and the volume rendering system. The passive introduction averaged around 5 minutes. Evaluator V videotaped steps 3 to 5, and P conducted participant observation during the same time.

4. *Active Introduction.* The users were handed the 3D input device (a mouse with three buttons) and started exploring the first scenario with M still explaining. This step seamlessly faded into Step 5, and it averaged to 5 minutes.

5. *Exploration of the Scenarios.* M left the CAVE and served as an expert mediator, but he only helped users when asked to. About ten minutes before the end, it was switched to scenario two. This step was aborted when the total elapsed time of steps 3 to 5 reached 30 minutes.

6. *Follow-up Questionnaire.* Back in the conference room, the users filled out a questionnaire. Ten questions asked for the overall usability of the user interface, 24 dealt with single components of the volume rendering system. All these questions were based on Likert scales. Five additional questions were posed as multiple-choice questions.

7. *Focused Expert Interview (Debriefing).* Finally, a focused expert interview was conducted by P. It mainly addressed the system's functionalities and direct interaction. This interview took between 20 minutes and 1 1/2 hours. The users' comments were hand-noted.

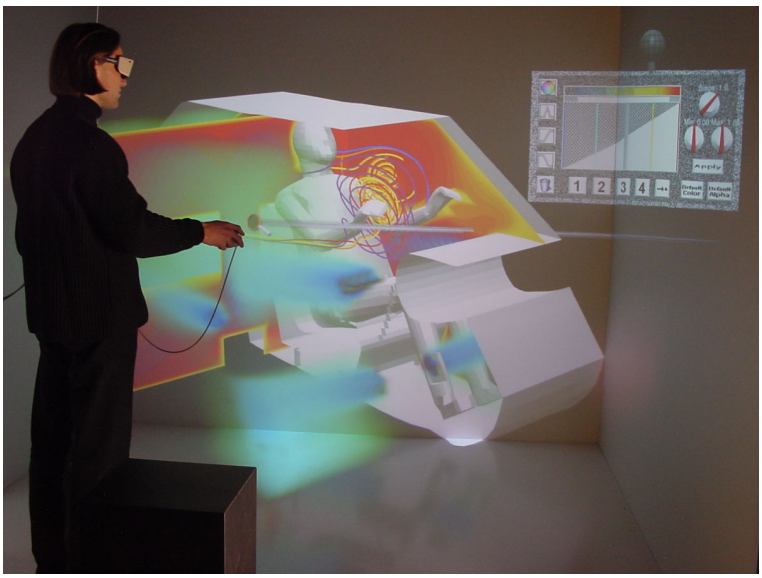


Figure 4.16: Volume rendered temperature distribution in a polygonal car cabin. (See also Color Plate 8 on page 126.)

4.2.2.3 Evaluation Results

Twelve volunteer users participated in the evaluation. They either came from related departments at the University of Stuttgart, or they worked with visualization software in the industry. Table 4.1 shows that there was the desired range of users' skills.

Table 4.1: Self-Rated experience on a Likert-Scale from 1 (low) to 5 (high).

Type of experience	Average score
CAVE or CUBE	2.83
Volume Rendering	1.90
Scenarios	1.13
Scientific Visualization	2.33

The evaluation of the follow-up questionnaire resulted in the following significant findings:

- For 2 users the image quality was not detailed enough, 7 users did not mind the poor image quality with large volumes.
- 11 users found the high quality mode useful, one found it awkward to use.
- 11 users thought the automatic alignment of the editor menu was useful.
- 10 users liked the layout of the user interface components.
- One user had trouble interacting with the rotary knobs.
- For 9 users, defining the alpha function with pins made sense.
- All users found the probe mode to be helpful, 10 users thought it was easy to use.
- 9 users found the clipping plane beneficial, 8 users thought it was easy to use.

The overall usability satisfaction with the system scored from 55% to 90%. The numbers are based on a composite measure derived from a System Usability Scale [10].

After having consulted both quantitative and qualitative data, the key findings that occurred across all investigative methods were worked out:

- Alpha and color pins are hard to differentiate when both are gray.
- The mouse button assignment is confusing when working with pins.
- Pin positioning is imprecise in the transfer function field.
- When entering the probe mode, some users were unaware of the activated clipping plane mode.
- Users requested a rectangular or trapezoidal alpha pin.
- Invisible colors in color bar resulting from transparent regions in the alpha function made it difficult to set the color of color pins.

- After deletion of all color pins the volume turns black and thus becomes invisible on a black background, even if the alpha function contains opaque regions.

The evaluation of the new volume rendering system showed its general usefulness. It was surprising how little the users were disturbed by the relatively poor image quality of large volumes, but this may in part be due to the existence of the high quality mode. Nevertheless, the user study found a number of usability issues, which were solved in a second evaluation approach.

4.2.3 Solving the Usability Issues

The first user study found several usability issues of the volume rendering system, most of them related to the transfer function editor. After the evaluation was finished, most of the major usability issues were addressed, and some functionality was added that was only requested by a few users. Here is a list of the changes:

- The mouse button assignment was confusing because separate buttons were used for selection and manipulation of pins. Now both can be done with the same mouse button. However, it can now happen that the user moves a pin accidentally.
- Imprecise pin positioning: users weren't aware of the exact pin locations. Now the respective data value is displayed below the selected pin.
- Users asked for a trapezoidal pin type. Therefore, a width parameter was added to the peak pin, so that it can be widened to the shape of a trapezoid. Additionally, the minimum value parameter was removed because none of the participants had used it and it could be substituted by a wide trapezoid.
- Previously, the color bar displayed the transfer function colors and opacities in one bar. Thus, users couldn't see the colors of transparent regions which confused them. Now the color bar is divided into two parts: one part displays only the color, the other additionally represents the opacity.
- After deletion of all color pins, the volume turned black even if opaque regions existed in the transfer function. In the new version, the default color is white, which makes the volume visible on black projection walls.
- The rotary knobs in the transfer function editor were hard to use for a precise setting of values, because they transformed hand rotations 1:1 into knob rotations. Therefore, a fine-tuning mode was added, used by turning the knob when the right mouse button is pressed. Now, angular changes of the hand are converted into knob rotations at a ratio of 1:10. The left mouse button can still be used to do 1:1 rotations.
- Some users had requested an undo function, so it was implemented. It stores the last 20 actions that affected the transfer function. Simple clicks on the undo button go back one step at a time.

- Since some users had asked for a histogram of the data values in the volume, a button was added that displays a 2D histogram as the background texture of the function window.
- Some users were irritated by the transfer function editor's background color, which was a pattern of black and white dots. It made some text difficult to read. In the new version, the background color is solid black.
- Because nobody used them, and because an undo function and an option to save the volume with the transfer function had been added (see below), the preset buttons were removed from the transfer function editor.

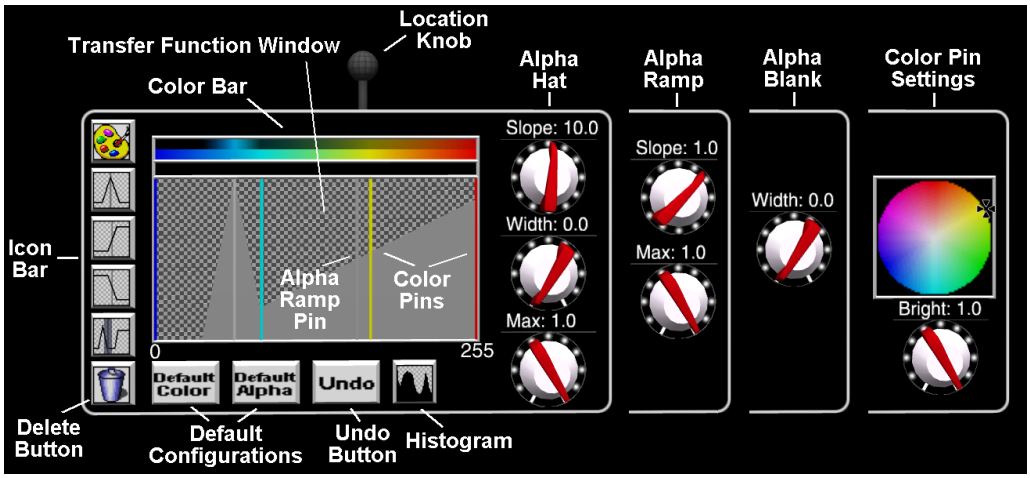


Figure 4.17: The improved transfer function editor.

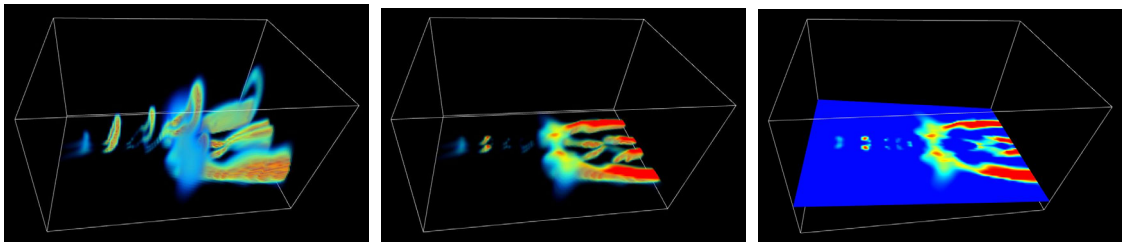
Figure 4.17 shows the improved transfer function editor with several alternatives for the knob layout, which are displayed depending on the selected pin type.

Figure 4.18: The improved volume menu.

Additionally, three new options were added to the volume menu (see Figure 4.18). As an alternative to the existing clipping mode, opaque clipping was implemented, which clips both halfs of the object away, leaving a single plane. In this plane, the opacity transfer function is set to the maximum, making the plane opaque. Figure 4.19 shows the difference between the new clipping mode and the previous one.

Another new option in the volume menu is the discrete colors knob. If it is set to zero, discrete colors are not used, which means the color transfer function consists of continuous color gradients between the color pins, just as in the earlier implementation. If the knob is set to a value $n > 0$, the color transfer function is split into n evenly sized regions of constant color.

Finally, a save button was added to store both volume and transfer function in a single file.



(a) (b) (c)

Figure 4.19: Lambda dataset: (a) no clipping, (b) regular clipping, (c) opaque clipping.

4.2.4 Second Evaluation

Another user study was carried out to support that the new volume rendering system benefits from the improvements. This time, the collaborative aspects of the system were evaluated additionally, but these will not be discussed in this dissertation. Basically, the same evaluation procedure was applied as in the first user study, but due to the collaborative aspects two users worked on the same datasets at the same time. They were located in two networked CAVEs with audio link.

4.2.4.1 Scenario-Based Evaluation

Similar to the first user study, the participants should work in two scenarios, but with additional collaborative aspects. In the first scenario, the users were given the same Visible Human skull dataset as in the first study (see Section 4.2.2.2). One of the users should present features of the dataset to the other participant, who was supposed to ask questions to the presenter. The presenter was advised to make use of the transfer function editor to demonstrate anatomic details of the skull.

For the second scenario, a knee joint, was loaded into the volume rendering system, which was also extracted from the Visible Human dataset (see Figure 4.20). Beforehand, a needle was hidden in the dataset. The needle was represented by a simple straight line of

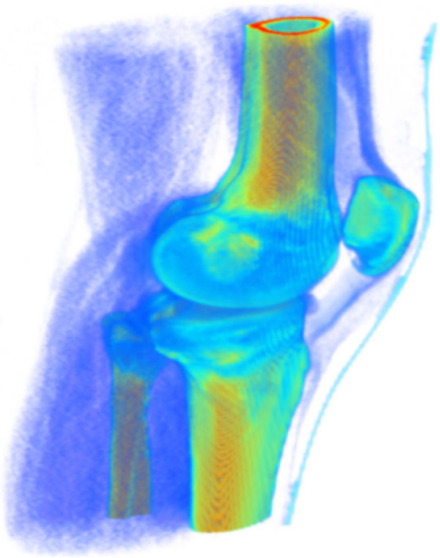


Figure 4.20: The visible human knee.

voxels with the same data values as bone, which could only be seen after precise modifications of the transfer functions. The participants had to collaborate to find the needle as fast as possible and tag it with a cone shaped marker (see Figure 4.21). This task required careful adjustments of the transfer functions.

Five pairs of people from different professional fields were invited, some of them had already participated in the first study. Most of the ten participants were members of neighboring research centers. The user study took place on three consecutive days and had a mean evaluation time of three hours per pair.

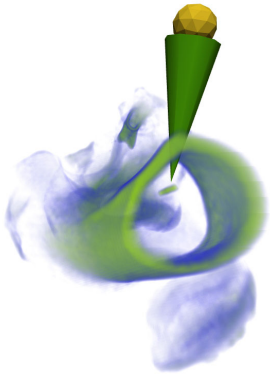


Figure 4.21: Scenario #2: to find the needle.

The evaluation started with briefings for the participants. The briefings consisted of Powerpoint presentations explaining the volume rendering system and the tasks which had to be solved. After the presentations, the participants filled out a self rating questionnaire, which was similar to the one used in the first study.

Then the participants were guided to the CAVEs and they were given short introductions to the user interface. This part of the evaluation took about 15 minutes.

When both of the participants felt comfortable with the user interface, they were connected to the audio system and from then on they could communicate with each other. Also, video recording of the session was started. The participants had 15 minutes to work in each of the two scenarios. During all of this time, the participants could ask the observers questions about the software. Occasionally, if the participants ignored them, the presenter asked the participants to use one of the new features of the transfer function editor. If after about 10 minutes into the search task there was no progress, the observers began to give hints about the location of the needle. During the time in the CAVEs, the observers made notes about the participants' questions and their behavior.

After the tasks had been solved, the participants went back to the briefing rooms. They filled out questionnaires which were split in two parts: multiple choice questions and questions for free answers. The multiple choice section contained questions about collaboration, and technical issues with the user interface. This part of the evaluation lasted about 20 minutes.

After that, the participants met in one of the briefing rooms, where they were consulted about their experiences in the CAVEs. The interviews followed pre-designed guidelines and took up to one hour.

4.2.4.2 Evaluation Results

This section will only cover the results that were obtained from the questions regarding the user interface.

- Unlike in the last user study, the participants were not confused by the mouse button assignment for the pin usage. Previously, two mouse buttons had to be used for selection and manipulation, now one button can do both. Only 2 out of 10 users said they had problems with the new usage scheme, while previously 7 out of 12 were confused with the button assignment.

- Previously the pin positioning was considered to be imprecise because there was no feedback about the exact pin position. With the display of the current data value, only 3 users requested more information.

- The new trapezoidal pin was accepted quickly, 8 users found it easy to use.

- The color bar division into one with and one without opacity representation, compared to only one bar with opacity, was considered helpful by all participants.

- 8 users liked the new fine tuning mode for rotary knobs.

- 9 users were happy with the transfer function editor's new solid black background.

- The new undo button was found to be useful by all but one user, who said it did not work as expected.
- 8 users found the histogram display useful.

Particularly in the search scenario, the participants had to use many functions of the transfer function editor, which was much less required in the first evaluation. The result shows clearly that the user interface has improved since the first evaluation.

Chapter 5

Parallelization and Distribution Methods

In this chapter, a parallelized version of the perspective shear-warp algorithm will be presented. The parallelized algorithm is designed for distributed memory machines using MPI. Fast parallel computers are often installed at remote locations, where they can only be accessed via a network connection. Local display computers typically do not offer fast parallel processing. The new algorithm composites the intermediate image on a remote parallel computer and transfers it to a local display computer. Transferring the intermediate image takes advantage of the feature that the warp can be done very fast even in simple graphics hardware. With this approach, even low end PCs or laptop computers can display complex volumetric data interactively. The proposed remote volume rendering algorithm can also be used in virtual environments, provided that there is a fast network connection to a high performance parallel computer.

The research presented in this chapter was carried out in collaboration with Ulrich Lang, and it was published in [82] and [83].

5.1 The Parallelized Perspective Shear-Warp

Although today interactive volume rendering is mostly done with specialized computer graphics hardware—which is usually high end graphics equipment with fast 3D texturing and large texture memory—this technique has limitations. Only volume datasets that fit into texture memory can be rendered at interactive frame rates, larger volumes have to be swapped between texture and main memory, which is slow. For software based volume rendering approaches, single PCs are not fast enough to display large volume datasets interactively.

Another bottleneck of the texture based approach is the pixel fill rate. It is still not high enough to reach interactive frame rates on a 1024^2 pixels screen. Display screens of this resolution are commonplace in virtual environments, which are the motivation for the developments presented in this chapter. In many installations, a large visualization machine drives multiple display screens with stereoscopic images to create the effect of immersion. The two most widely used approaches to drive virtual reality environments are

high-end multi-pipe graphics machines and networked PCs. Networked PCs suffer from the same limitations for volume rendering as single PCs, and high-end graphics hardware is expensive.

In the recent past, clusters of commodity PCs have gained importance in the field of parallel computing. These clusters are usually linked with Fast Ethernet or Myrinet, both providing high bandwidth and low latency. Many PC clusters are competitive to special purpose parallel computers. Due to their lower price, they do not need to be installed in central places, but they can be located wherever they are used, for instance in a physics or engineering department at a university. This decentralization of parallel computing power increases the number of places where interactive compute time is available on a parallel architecture.

The availability of large numbers of interactive nodes on parallel computers encourages us to use them for volume rendering in conjunction with a visualization computer that is connected to multiple stereo displays. The shear-warp algorithm is a very fast CPU-based volume rendering algorithm.

Although on single processor machines the shear-warp algorithm is usually slower than hardware supported solutions, its good scalability allows it to be competitive on multi-processor machines. Lacroute [52] and Jiang and Singh [41] showed that the orthogonal projection shear-warp algorithm scales well on shared memory parallel computers. Amin et al. [3] and Troutman et al. [93] demonstrated that it can be used efficiently on distributed memory machines. This chapter describes how the shear-warp algorithm for perspective projection can be employed efficiently for remote volume rendering on parallel computers.

5.2 Previous Work

A detailed description of the shear-warp algorithm can be found in Section 2.1.3. The version for perspective projection is discussed in Section 3.2.

Related work in the field of CPU-based volume rendering on parallel computers includes VFleet [98], which uses a ray casting renderer and a compositing tree, but it does not offer shear-warp rendering and is not real-time. At the University of Utah an interactive ray tracing system for volume visualization on parallel computers was developed [65]. It allows to render volume data in real-time, using isosurfacing or maximum-intensity projection. However, volume rendering with arbitrary opacity transfer functions and alpha blending is not supported.

Knittel's UltraVis system [45, 46] renders volume data in real-time using a ray casting algorithm. Extensive use of the MMX and SSE units of Pentium processors and optimized cache memory access result in very high performance. PC cluster support allows the approach to scale with the number of available CPUs. However, the approach requires the presence of Intel CPUs, it cannot be used on other parallel computers.

At Sandia National Laboratories the Parallel Volume Rendering system (PVR) [71, 87] has been implemented. It uses a parallel computer to provide interactive rendering rates

for very large volumes using a ray casting algorithm and a sophisticated compositing scheme. The drawback of this system is that it does not render fast enough to drive a virtual environment with multiple screens.

One of the most recent developments that uses PC clusters for visualization is the WireGL [38] library, which acts as an OpenGL driver to an application but distributes the data which is to be displayed among several PCs. Chromium [39] implemented an improved handling of the large amount of data that has to be transferred for each frame before it can be displayed. For volume rendering, it allows the distribution of a volume dataset among all cluster nodes, each node rendering only its assigned partition. The drawback of this approach is that it requires a cluster of PCs with graphics cards, while for the volume rendering approach presented in this chapter a PC cluster without graphics hardware, or a special purpose parallel computer can be used, which may already be installed for science or engineering simulations.

5.3 The Rendering System

The development of the parallelized shear-warp algorithm with perspective projection is based on the work presented in Section 3.2. The parallel extensions were done in two areas: first, the perspective projection algorithm was parallelized, and second, a new remote renderer was written, which runs on a parallel machine and communicates with the local display machine via a network connection (see Figure 5.1). The network connection is established between the display computer and one node of the parallel computer.

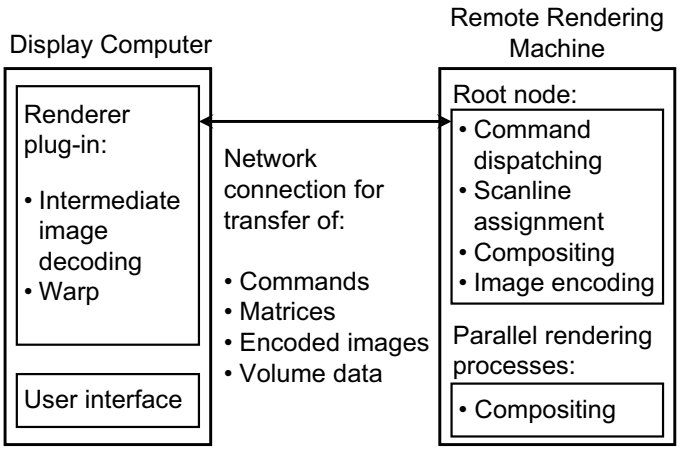


Figure 5.1: Remote rendering system components.

5.3.1 The Parallelized Shear-Warp Algorithm

In [52], Lacroute parallelizes both the compositing and the warp. The compositing is parallelized by partitioning the object space into sections of intermediate image scanlines,

which are distributed among the available processors. Additionally, dynamic task stealing is implemented to optimize load balancing. The warp is parallelized using static interleaved partitions without dynamic approaches.

The new algorithm only parallelizes the compositing, but not the warp. This is because, as shown in Section 3.2, the warp can be done efficiently by graphics hardware, even if only 2D texturing is supported. If 2D texturing is not accelerated by the display computer, the warp is still fast for small output images, but it becomes considerably slower for large output images. However, today 2D texturing acceleration is provided by most graphics hardware.

The compositing was parallelized by partitioning the intermediate image into sections of scanlines, similar to Lacroute's approach, but without task stealing. The idea is illustrated in Figure 5.2. Each process is assigned an equally sized section of the intermediate image. If the scanlines cannot be distributed evenly, the root node is the first to be assigned less lines than the other nodes, because it has to do the additional work of collecting all rendered sections and sending the result to the display machine.

For perspective projection, the compositing is more expensive than for orthogonal projection, because every intermediate image scanline does not only require data from two voxel lines, as in the case of orthogonal projection. Instead, it needs to look at multiple voxel lines, depending on the degree of the perspective. In the worst case, an entire voxel slice from the back of the volume has to be processed to compute a single intermediate image pixel. In general, the farther away the slice that is currently processed, the more voxels have to be accumulated for an intermediate image pixel. This is partly compensated because less pixels need to be drawn per slice.

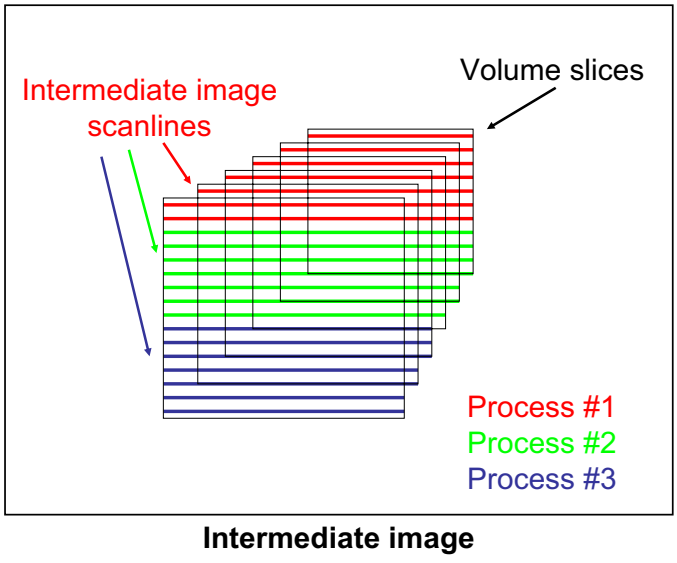


Figure 5.2: Intermediate image task distribution with sections of the same size.

This feature of the perspective projection, and the fact that shear-warp rendering requires the storage of three datasets in memory, one for each principal axis, makes it difficult to

distribute the volume data on distributed memory machines. In the approach presented here, each node has a copy of the entire volume dataset. If a large number of nodes is available, but there is not enough memory on each node to store the volume data, the available nodes could be split into three parts and get one volume dataset for each principal axis. Although the maximum usable volume size would be three times as high, this also means that only one third of the nodes can be used for rendering at a time.

5.3.2 The Renderer Plug-In

Since the intermediate image generation is decoupled from the actual drawing of the final image, the renderer plug-in for the existing volume rendering software is rather simple. It only needs to pass the current view matrix to the remote renderer, receive the intermediate image and the corresponding warp matrix, and warp the image to the screen. Additionally, all changes of image generation parameters have to be passed to the remote renderer. They include the transfer functions, interpolation mode, and image quality.

5.3.3 The Remote Renderer

At startup, the remote renderer expects a volume dataset. Depending on the volume size and the network connection, the transfer may take a few seconds. Then the three run length encoded versions of the volume data (one for each principal axis) are generated on the remote computer, and they are stored in each node. After that, the remote renderer is ready to receive commands from the renderer plug-in.

The following pseudo-code shows the flow of control for the root node and the other nodes in the parallel algorithm. The root node both distributes the commands and collects the resulting intermediate image sections. The reception is done by an `MPI_Recv()` command with the memory address for the destination of the sections, so no additional copying is necessary. When all sections have arrived at the root node, the intermediate image is run length encoded and then transferred to the renderer plug-in, along with the warp matrix.

```
procedure rootNodeRenderingLoop()
{
   Receive the view matrix from the plug-in.
   Compute the appropriate section partitioning.
   Pass the section partition parameters to the other nodes.
   Render self assigned section.
   Receive the rendered sections from the other nodes.
   Encode the intermediate image.
   Transfer the intermediate image to the plug-in.
}

procedure otherNodesRenderingLoop()
```

```
{
   Receive section parameters from the root node.
   Render the section.
   Transfer the rendered section to the root node.
}
```

The remote renderer is an MPI program without a user interface and with no user interaction after startup. This was an important design decision, because the renderer should run on as many different platforms as possible, and it should not require X Window support. In addition to the number of processes which is passed to the MPI startup tool, the remote renderer requires only two command line parameters: the port number and the display host address for the socket connection. Everything else is transferred from the display host at runtime, including the volume dataset.

5.3.4 Data Transfer

The data communication between the renderer plug-in and the remote renderer is done with one bidirectional TCP socket connection. It is established at startup and remains active until the application is closed. A TCP connection proved to be fast enough for the purposes of remove volume rendering, because the bottleneck is the compositing on the remote machine.

When the orthogonal projection shear-warp algorithm is used, the intermediate image pixels are usually mapped 1:1 to voxels. This can be done because the slices are only sheared and not scaled. In the case of perspective projection, the additional scaling makes the slices smaller when they are farther back. Thus, more than one pixel per voxel is used for the front volume slice. This ensures that the smaller slices map to enough pixels on the image, so that enough detail is retained.

For this reason, the intermediate images for perspective projection are larger than for orthogonal projection. Furthermore, the intermediate image size is constrained to edge lengths of powers of two, so the warp can be done without resizing the image—this is a 2D texturing hardware requirement. Typical 1024^2 pixel RGBA images require 4 megabytes (MB) of memory. An interactive frame rate of 10 frames per second would require a data transfer rate of 40 MB per second, which is far beyond the bandwidth of Fast Ethernet (100 Mbit/s).

The intermediate image typically contains large transparent regions, which can efficiently be run length encoded. Two run length encoding algorithms were implemented: the first algorithm encodes the entire intermediate image, the second encodes only the rectangular window which was actually worked on in the compositing step (see Figure 5.3). It turned out that for large window sizes the first algorithm is faster, but in all other cases the second algorithm is faster. Some performance numbers can be found in Section 5.4.3.

An important issue with the compression algorithm was to make sure that no memory is unnecessarily copied, allocated or deallocated in the process of encoding and decoding.

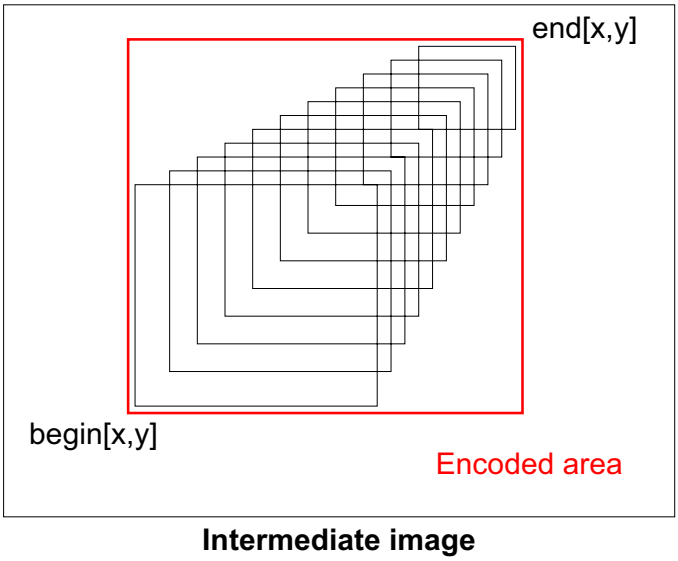


Figure 5.3: Encoding of actually used intermediate image window.

This goal was reached by not re-allocating memory when the intermediate image size remains the same or decreases. A re-allocation is done only for images larger than the allocated space. Furthermore, the intermediate image data is stored only once, so only a pointer to it is passed among the functions that work with it.

5.3.5 Overall Algorithm

The data flow for the rendering of one frame is shown in Figure 5.4. The display computer does not need to keep the volume data in memory after it was transferred to the remote renderer at startup.

5.4 Results

The parallelized perspective projection rendering algorithm was tested on the following three parallel computers, which are displayed in Figure 5.5:

- An SGI Onyx2 with 16 195 MHz R10000 processors and 16 GB of shared memory.
- A SUN Fire 6800 node with 24 UltraSparc III 750 MHz processors and 96 GB of shared memory. Up to 8 processors were available for interactive use.
- A Fujitsu-Siemens cluster of 32 Linux PCs with 64 Pentium 4 Xeon processors at 2.4 GHz and Myrinet between the nodes.

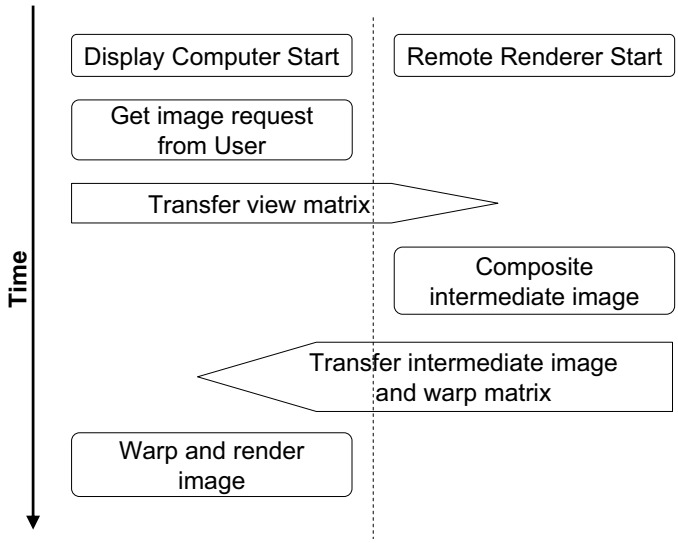


Figure 5.4: The remote rendering data flow.

(a) (b) (c)

Figure 5.5: Parallel rendering platforms: (a) SGI Onyx2, (b) SUN Fire 6800, (c) Fujitsu-Siemens PC Cluster.

Apparently, for the shared memory machines the algorithm could have been written in OpenMP or with threads. But since the program had to run on any multi-processor architecture, MPI was used.

The display machine was an SGI Onyx2 with four R10000 processors at 250 MHz, 4 GB RAM and Infinite Reality 2 graphics. It was linked to the above Onyx2 by a 1 Gbit/s Ethernet connection and to the PC cluster via 100 Mbit/s Ethernet. Both Onyx2 systems and the PC cluster were located in the same building at the HLRS. The SUN was located about 100 km away in the city of Ulm, and it was connected to the display machine via 100 Mbit/s Ethernet.

The dataset that was utilized to test the performance of the parallelized algorithm is the General Electric CT engine. It was used in two different sizes: "large" is a $256 \times 256 \times 110$ voxels version, "small" is a $128 \times 128 \times 55$ voxels downsampled version. The opacity transfer function was set to a linear ramp from zero to full opacity. The image generation was carried out in 24 bits RGB color space. Whenever the large engine was used, the intermediate image size was 1024^2, for the small engine it was 512^2 pixels. The inter-

mediate image was transferred using run length and window encoding. For all tests the volume was rotated 180 degrees about its vertical axis in 90 steps of two degrees.

5.4.1 Overall Rendering Performance

In the following three subsections, the rendering performance of the multi-processing platforms, which were used in the tests, will be displayed. For each graph the remote renderer was executed with different numbers of processes. The initialization of the MPI environment ensured that each process could run exclusively on its own processor. The length of the bars reflects the average rendering time per frame needed for the above mentioned 180 degrees rotation. The sections of the bars display how the total rendering time was distributed to specific tasks.

The idle time of the renderers is largely the time the display machine needed to decode the intermediate image, transfer it to texture memory, and display it on the screen. During this time the renderer waited for the next view matrix. There is also an implementation with pipelining to hide the idle time, but it was not used in the performance tests. It will be addressed in Section 5.4.5.2.

In all three performance tests, image decoding took about 29 milliseconds (ms) and drawing took 16 ms. Idle times that occur due to processes waiting during compositing are included in the total compositing time. In each of the three performance tests the large engine dataset was used.

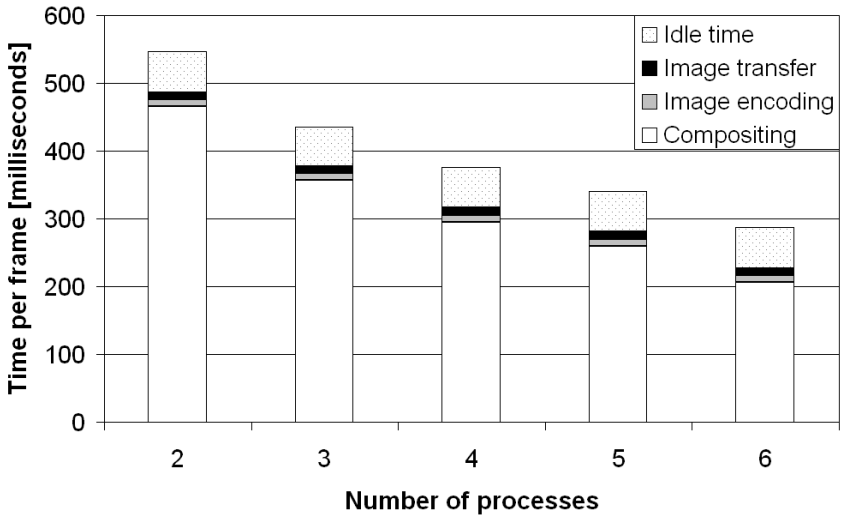


Figure 5.6: SUN Fire rendering performance.

5.4.1.1 SUN Fire

Figure 5.6 shows the rendering performance of the SUN Fire. The compositing step took most of the total time, while image encoding and image transfer both account only for very little time: encoding took 9.9 ms and the transfer took 11.1 ms.

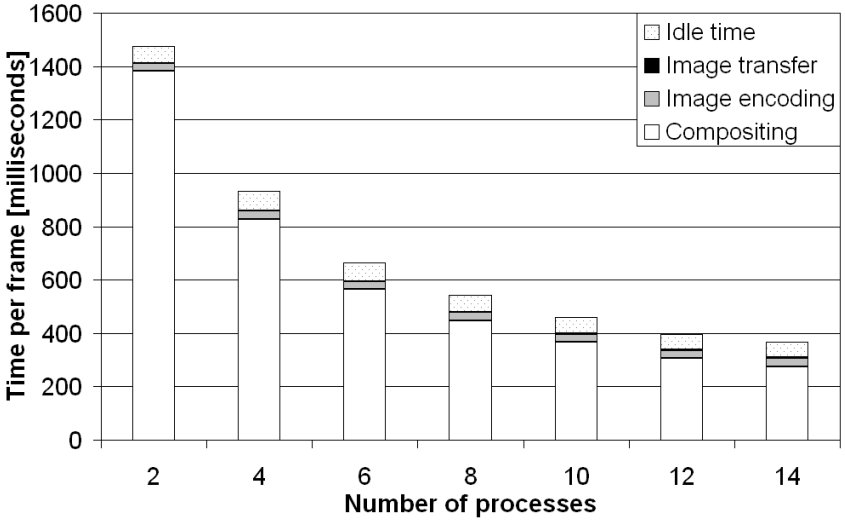


Figure 5.7: SGI Onyx2 rendering performance.

5.4.1.2 Onyx2

Figure 5.7 shows the rendering performance of the SGI Onyx2 system. Because of the fast network connection to the display machine, the image transfer took only 1.7 ms in all the tests and is hardly visible in the diagram. Image encoding took 29.2 ms.

5.4.1.3 PC Cluster

The rendering performance of the PC cluster is displayed in Figure 5.8. It differs significantly from the previous two machines. The PC cluster's computing power makes it the fastest tested machine with a minimum rendering time of 132 ms per frame. Image encoding took 3.0 ms, the image transfer accounts for 31.4 ms. The relatively slow transfer is due to the endianness adaptation that is required between the PCs and the SGI.

5.4.2 Compositing

Section 5.4.1 showed that the compositing is the most time consuming rendering step, which is why it was parallelized. Its performance can be judged by comparing the times of the total compositing, i.e., the time it takes before all processes are done with compositing, with the average compositing time of the processes. With perfect load balancing these values would be equal. Figure 5.9, which shows the performance of the Onyx2, indicates that the numbers are not equal. The solid line shows the total compositing time, while the dotted line shows the average time it actually took the processes to composite their sub-tasks. The space between the lines reflects the improvement that would be achievable by optimizing the load balancing.

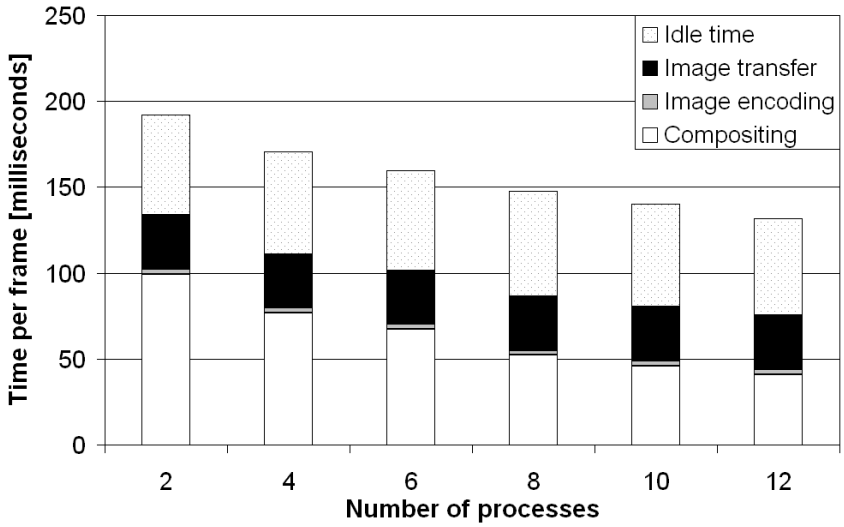


Figure 5.8: PC cluster rendering performance.

5.4.3 Intermediate Image Transfer

The comparison of the (non-parallelized) run length encoding, transfer, and decoding times for the three implemented encoding types in Figure 5.10 shows the great advantage of window encoding. Here, only the part of the image that was actually composited is run length encoded. In the test, the encoding was done on the SUN Fire, then the image was transferred to the SGI Onyx2, where it was decoded. For this test, the large engine dataset was used, and the intermediate image size was 1024^2 pixels.

5.4.4 Shear-Warp vs. 3D Texture Hardware

Section 3.2 showed that the rendering speed of the shear-warp algorithm is nearly independent of the output image size when the warp is done in texture hardware.

However, the 3D texturing hardware volume rendering approach is highly dependent on the output image size due to its pixel fill rate limitation. In Figure 5.11, the rendering times for output image sizes from 300^2 to 900^2 pixels are shown for both algorithms, using the small engine dataset. The texture hardware algorithm was used on the Onyx2, the perspective shear-warp algorithm was used for the compositing on the SUN Fire with four processors, and the Onyx2 did the warp. The graph shows that for an image size of 900^2 pixels, both algorithms are about equally fast.

5.4.5 Discussion

In this section, the performance numbers from the previous section are discussed, and ideas on how to further improve the performance are given.

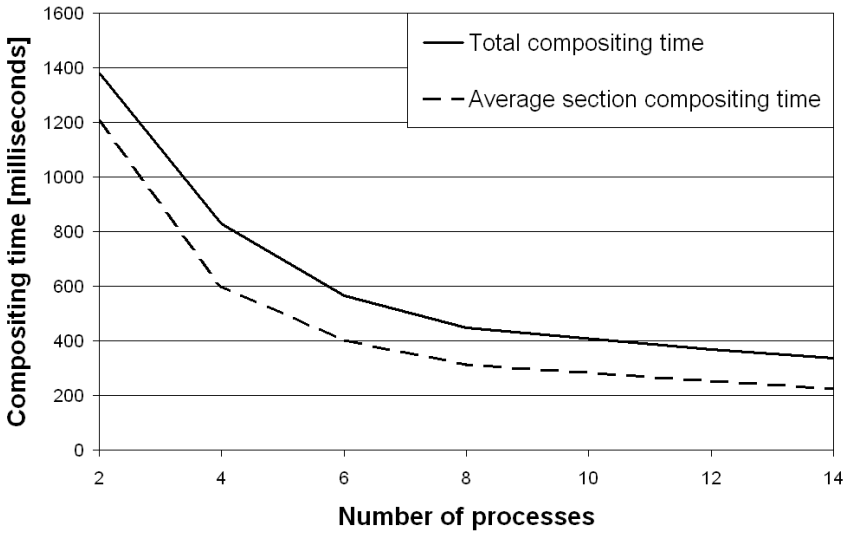


Figure 5.9: Total compositing vs. average section compositing.

Table 5.1: Maximum rendering speed of the tested machines.

Machine	# processes	images per second
SUN Fire	6	3.5
SGI Onyx2	14	2.7
PC cluster	16	8.4

5.4.5.1 Performance Comparison

The fastest rendering rates achieved by each system are listed in Table 5.1. The PC cluster is fastest with 8.4 images per second. The image transfer rates are similar for the two machines which are connected to the display computer via 100 Mbit/s connections, and with firewalls in-between. The direct gigabit connection between the two Onyx2 systems is worthwhile, it allows the shortest transfer time in the test. The PC cluster's Pentium 4 processors are so much faster than the other two architectures that the compositing ceases to be the dominant factor in the rendering process. Now image transfer and idle time, although roughly the same for the SUN Fire, are the most expensive parts.

5.4.5.2 Latency Hiding

A comparison of the performance numbers of the three tested systems shows that for the SUN and the SGI the compositing time dominates, while the PC cluster spends a large fraction of the time transferring the intermediate image to the display machine and waiting for the display machine to send a new view matrix.

While the image transfer time could be reduced by a faster network connection, the idle time can be used to begin the computation of the next image: as soon as the display

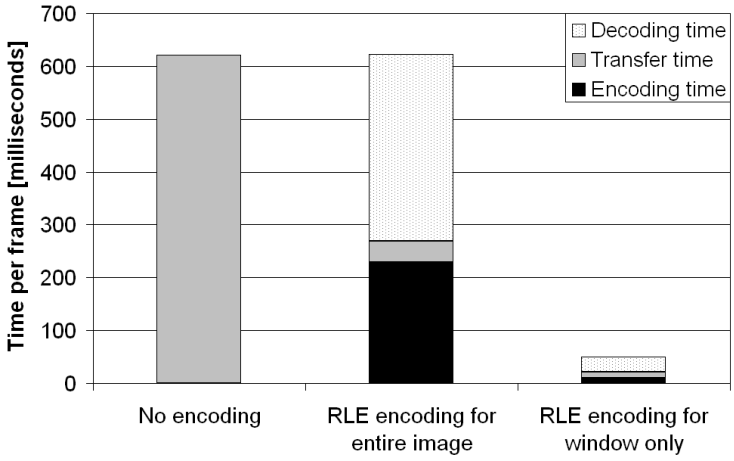


Figure 5.10: Intermediate image run length encoding graph.

computer receives the intermediate image, it sends the view matrix for the next image to the rendering system. This pipelining approach leads to effective latency hiding, and it was implemented for optional use. However, for the performance tests in this chapter, no pipelining was used in order to show the actual timings.

5.4.5.3 Image Decoding Time

A significant part of the rendering processes' idle time results in the display machine decoding the intermediate image. The decoding is not parallelized, since it usually does not run on a parallel computer. The Onyx2 that was used decodes with a 250 MHz R10000 processor, but current PCs are much faster. In another test a Windows PC was used as the display computer. It contains a Pentium 4 at 1.4 GHz, and a 3Dlabs Wildcat II 5110 graphics card.

With this PC, the intermediate image decoding time decreased from 28 ms on the Onyx2 to 6.8 ms. Looking at the overall performance, it is remarkable that the idle time increased, as seen in Figure 5.12. Obviously, the compositing and image encoding times did not change, compared to the test in Section 5.4.1.1.

Looking at the performance numbers, it can be seen that the time it takes to draw the intermediate image with texture hardware, which was 17 ms on the Onyx2, increased to 81 ms on the PC. This is due to the slower image transfer to texture memory on the Wildcat.

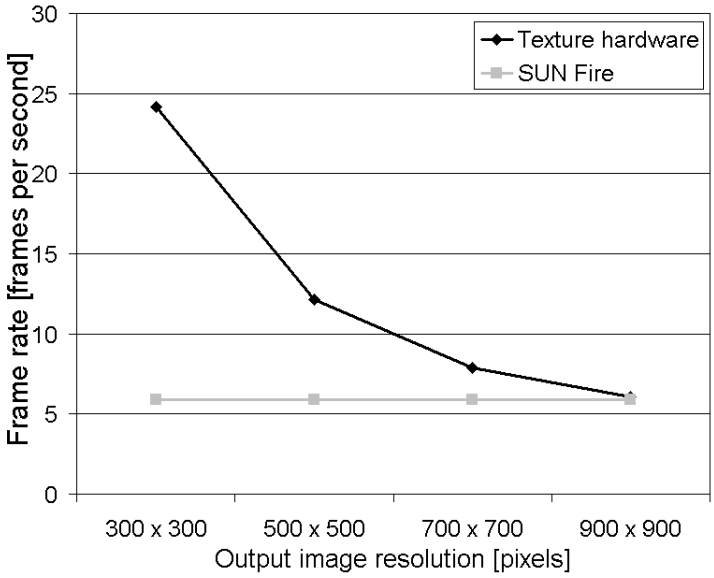


Figure 5.11: Texture hardware vs. shear-warp algorithm.

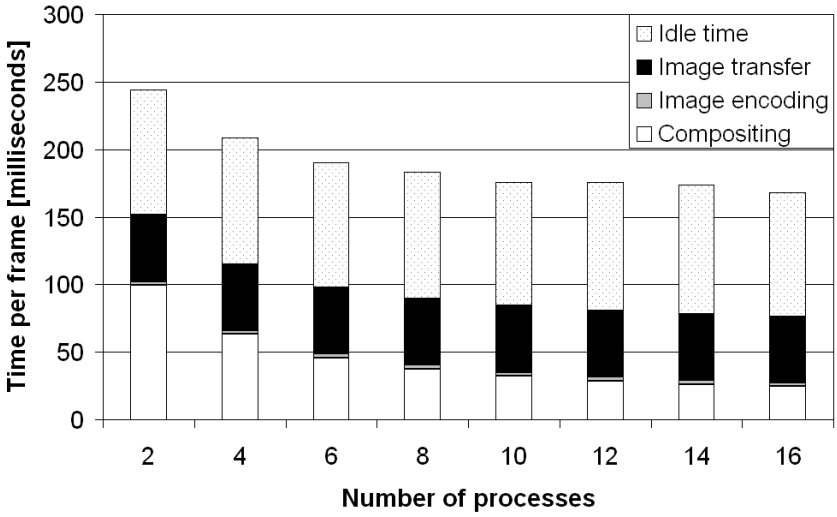


Figure 5.12: Windows PC is display machine, PC cluster renders.

Chapter 6

Volume Visualization System

The previous chapters have presented new contributions to interactive volume rendering in three fields: rendering algorithms, interaction techniques, and parallelization and distribution. In this chapter, some parts of the development environment will be described, and it will be shown how the new algorithms were assembled to extend an existing visualization framework with interactive direct volume rendering.

6.1 Development Environment

In the process of the research that was carried out for this dissertation, the author developed a test bed for the implementation of new algorithms. This environment is called "Virvo", which is short for "*vir*tual reality *vo*lume rendering" [80]. Virvo consists mainly of a volume rendering front-end with a graphical user interface (VShell, see Figure 6.1), and a remote renderer which can be used for shear-warp rendering on parallel computers (VRemote). The core parts of the development environment will be referenced in Section 6.2, which is why they will be described below.

VShell is a hybrid C++ and Java application using the Java native interface (JNI) for communication. The user interface was implemented with Java's Swing widget library. Rendering, network communication, and file handling were written in C++. The rendering window is a Java canvas of which the C++ part knows the OpenGL handle so it can draw on it. Input device handling is performed by Java routines that call the respective C++ routines for all actions that happen in the OpenGL canvas.

In Virvo, several volume rendering algorithms have been integrated: Lacroute's Volpack, orthogonal and perspective projection shear-warp versions, SGI Volumizer, and multiple algorithms based on textured slices. For image quality comparisons, the application can switch between any of these algorithms while maintaining the viewing parameters. Furthermore, several rendering options, for instance interpolation mode, image quality, or texturing style can be changed at runtime.

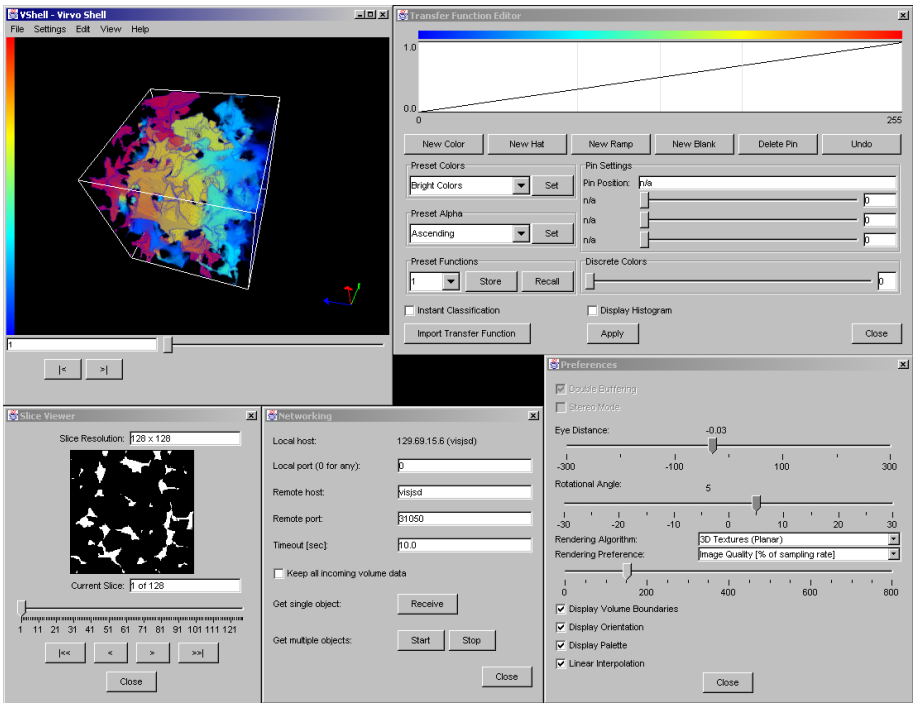


Figure 6.1: The rendering front-end VShell. (See also Color Plate 9 on page 126.)

Virvo also supports clipping planes, different viewing and projection matrices, and remote rendering. The Virvo class structure, which is depicted in Figure 6.2, allows to extract rendering and file handling classes and use them in other applications. The rendering classes, which are highlighted with a gray background in the class diagram, compile independently from the rest of the Virvo system, and they require only an active OpenGL context. It was important that changes of the OpenGL state that were required for rendering were reset before the end of the rendering routine.

VRemote is a command line C++ program that uses the MPI interface for parallelization. The command line arguments consist of the name of the visualization host, the port number for the TCP connection, and an optional parameter to set the encoding style for the intermediate image. The tasks of the remote rendering utility have been discussed in detail in Chapter 5. On the display computer, VRemote requires a counterpart, which provides it with volume data, rendering options, and viewing matrices.

6.2 Integration in Visualization Framework

For three reasons, the visualization framework COVISE was selected as the environment for the volume rendering system. First, COVISE is a versatile framework which is not restricted to a specific visualization field, and it supports virtual environments. Second, COVISE is extensible in important places: the virtual reality renderer supports plug-ins, and there is an API to complement the collection of modules influencing the data flow.

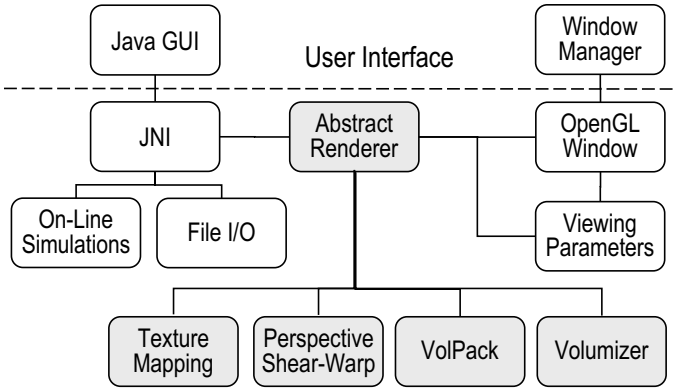


Figure 6.2: The class hierarchy of VShell.

Third, changes had to be made to the source code for extensions of the renderers—which was only possible with COVISE, because it has been developed at the HLRS.

A short introduction of COVISE can be found in Section 2.4.2.1. For the integration of the new volume rendering algorithms, mainly data format issues, file handling, and rendering had to be dealt with. Each of the following sections will discuss one of these aspects. Each section begins with the status of COVISE at the beginning of the integration efforts. Then the contributions of this dissertation will be described, and it will finally be presented how the new algorithms were integrated into COVISE.

6.3 Data Formats

This section reports on the native data formats of COVISE [15], and it will show how they could be used to represent volume data.

6.3.1 Basis

COVISE did not have any specific volume data format, but it was aware of three dimensional scalar fields. This type of data is referred to as `DO_Structured_S3D_Data`. It is simply an array of scalar floating point data values. Therefore, it has to be accompanied by information about the data grid. All of the research in this dissertation is based on volume data on regular grids. In COVISE, this data type is called `DO_UniformGrid`. It containes information about how many elements are stored in each dimension, and about size and location of the data field in object space.

Additionally, the 32 bits integer type “packed colors” was used. It contains RGBA data with 8 bits per channel, all of which stored in one 32 bit integer value.

6.3.2 Contributions

Virvo can deal with four types of volume data:

- 8 bits per voxel scalar data
- 16 bits per voxel scalar data
- 24 bits per voxel RGB color data: 8 bits are stored for each color component
- 32 bits per voxel RGB color plus scalar data: the data value is used as a lookup into the opacity transfer function

The 16 bit data type is predominantly used with DICOM data. However, although DICOM files store 16 bits per pixel, most of them only actually utilize 12 bits. The rendering algorithm based on textured slices, which is used in this dissertation, can rasterize 12 bits per texel if supported by the hardware.

6.3.3 Integration

Within most parts of the data flow network, the representation of volume data is based on COVISE's `DO_Structured_S3D_Data` and `DO_UniformGrid` data types. This means that every data value occupies one floating point value in memory, even if it was originally represented by eight bits. It also means that 16 bit values are processed at full precision. 24 and 32 bit volume data can only be used within the virtual reality renderer by loading files directly from disk.

The "packed colors" data format is used for RGBA data generated by the module `ColorEdit`. When the renderer receives this data type, the transferred RGBA data will be used directly in the compositing.

6.4 File Handling

This section will discuss the input and output (I/O) routines that were integrated for work with volume data.

6.4.1 Basis

COVISE has a large number of I/O modules for data types used in a variety of simulation tools, but it did not have read or write modules for volume data types. COVISE did offer modules that could read or write scalar data fields on uniform grids, but the scalar data was always stored as floating point numbers. There was no mechanism to read or write 8 or 16 bit values, or even to store compressed data in the files.

6.4.2 Contributions

Because the COVISE data types were inefficient with disk space and not flexible enough to store additional volume rendering related information, the standardized file type 3D TIFF was considered. However, it cannot store multiple time steps, as well as the variety of Virvo's volume data formats, or information about the data location in space. Because 3D TIFF is an extensible format, these features could have been integrated, but the resulting files would not have been readable by applications unaware of these extensions. Furthermore, due to their complex structure, 3D TIFF files are not easy to be generated by other applications. Therefore, three new volume file formats were developed: RVF, XVF, and AVF. They will be presented in the following subsections.

6.4.2.1 RVF: Raw Volume File

RVF is the simplest of the new file types. This type can easily be created from many linearly stored raw voxel datasets on disk by adding a header of 3×2 bytes (big endian) for the volume's width, height, and depth in voxels. Due to its simplicity, the header can be added manually with a hex editor. This format supports only 8 bits per voxel, the storage of only one time step per file, and no transfer functions. The data order is the same as for the letters in an English book: it starts top left in the front, continues first to the right, then down, and then back.

Table 6.1: XVF file header.

Length	Data Type	Description
9 bytes	char	file ID string: "VIRVO-XVF"
2 bytes	unsigned short	offset to beginning of data area, from top of file [bytes]
2 x 4 bytes	unsigned int	width and height of volume [voxels]
4 bytes	unsigned int	number of slices per time step
4 bytes	unsigned int	number of time steps
1 byte	unsigned char	bits per voxel (supported values: 8, 16, 24, 32)
3 x 4 bytes	float	real world voxel size (width, height, depth) [mm]
4 bytes	float	duration of one time step [seconds]
2 x 4 bytes	float	physical data range covered by the voxel data (minimum, maximum)
3 x 4 bytes	float	real world location of volume center (x,y,z) [mm]
1 byte	unsigned char	compression type (0=none, 1=RLE)
2 bytes	unsigned short	number of transfer functions
2 bytes	unsigned short	type of transfer functions: 0 = 4 x 256 Byte for RGBA channels, 1 = list of control pins

6.4.2.2 XVF: Extended Volume File

The XVF format can store more information than RVF, but it is more difficult to create its header manually. XVF files can store multiple volume time steps in one file, plus an arbitrary number of transfer functions. All volume data types presented in Section 6.3.2 can be stored. The byte order of all numbers is big endian, floating point values are stored in the 4 byte IEEE standard format. XVF files optionally support run length encoding. The header specification can be found in Table 6.1. After the header, the voxels are stored in the same order as in RVF files, all bytes of each voxel are stored consecutively.

Table 6.2: Sample ASCII volume file with $3 \times 2 \times 2$ voxels.

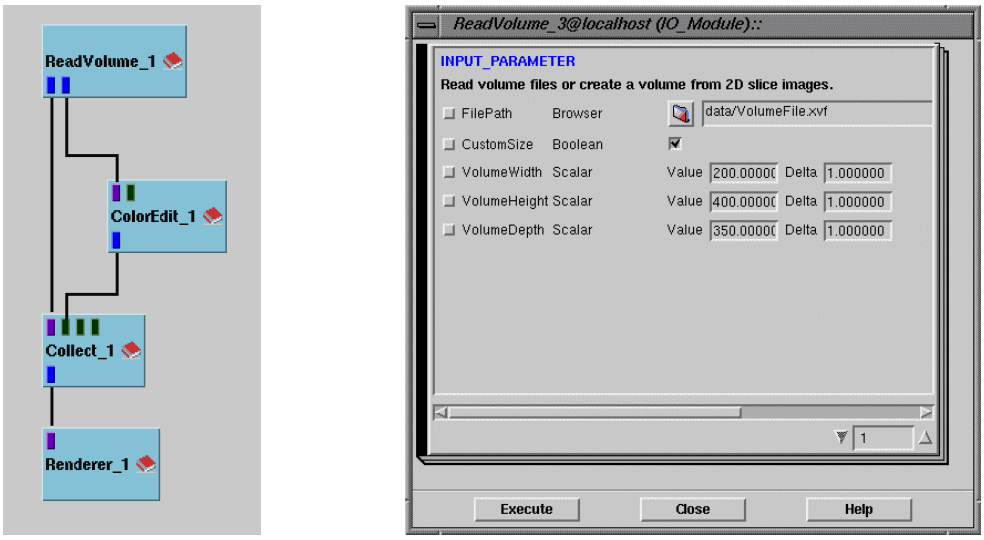
WIDTH	3
HEIGHT	2
SLICES	2
FORMAT	SCALAR8 # 8 bit data
XDIST	1.0
YDIST	1.0
ZDIST	1.0
0.7 0.9 0.1	
0.9 0.7 0.6	
0.1 0.6 0.1	
0.2 0.7 0.3	

6.4.2.3 AVF: ASCII Volume File

AVF files are ASCII representations of volume data. This format provides a simple and verifiable way for users to create volume files. AVF files consist of a header and a data section. In the header, several lines give information about the data format. Each line consists of an identifier and a value, separated by whitespace. This file format cannot store transfer functions. Comments starting with "#" may occur. The specification of this format can be found in the COVISE documentation [15]. Table 6.2 shows a sample AVI file.

6.4.3 Integration

For file handling in COVISE, a read and a write module were created. They are called `ReadVolume` and `WriteVolume`, respectively. The following two subsections will describe their functionality and the way they are used within COVISE.



(a) (b)

Figure 6.3: `ReadVolume`: (a) map, (b) preferences.

6.4.3.1 Read Volume Files

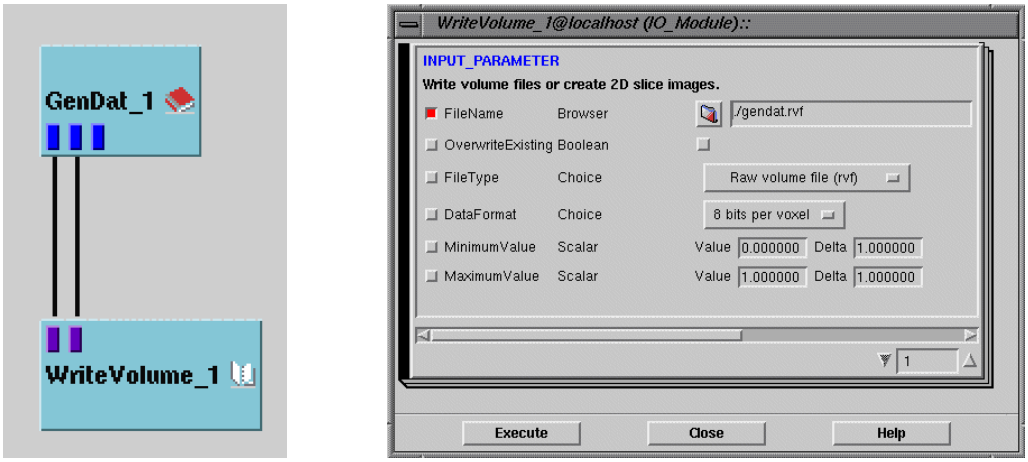
Figure 6.3a shows a typical COVISE data flow network which reads a volume file and displays it with a renderer. The module `ReadVolume` accepts several 3D volume data types and sequences of 2D slice images, which are listed in Table 6.3.

Table 6.3: File types supported by `ReadVolume`.

File Extension	Description
rvf	Raw Volume File
xvf	Extended Volume File
avf	ASCII Volume File
tif	3D TIF File
dat	Raw volume data (no header) - automatic format detection
nrd	Nrrd volume file [42]
dcm	DICOM file
rgb	RGB image file
pgm	Portable Graymap file
ppm	Portable Pixmap file

Figure 6.3b displays the preferences window of `ReadVolume`. The source file name is expected at the `FilePath` entry. If `CustomSize` is selected, the volume size will be set as entered in `VolumeWidth`, `VolumeHeight`, and `VolumeDepth`. Otherwise, the size entries are ignored and default values, or the values from the volume file if present, are used.

6.4.3.2 Write Volume Files



(a) (b)

Figure 6.4: `WriteVolume`: (a) map, (b) preferences.

The module `WriteVolume` was created to write volume data to disk. Figure 6.4a shows a sample COVISE network which writes volume data created by the module `GenDat`. Table 6.4 lists the file types which are supported by `WriteVolume`.

Table 6.4: File types supported by the module WriteVolume.

File Extension	Description
rvf	Raw Volume File
xvf	Extended Volume File
nrd	Nrrd volume file [42]
pgm, ppm	Density or RGB images (depending on volume data type)

Figure 6.4b shows the preferences window of the module `WriteVolume`. The `FileName` entry expects the name of the destination file. If `OverwriteExisting` is selected, the destination file will be overwritten if it exists. File type and data format can be selected with the respective choice menus. `MinimumValue` and `MaximumValue` allow to constrain the stored data range: all values up to `MinimumValue` will become zero, values greater than `MaximumValue` will become the maximum value of the selected data format. The remaining values will be mapped to the data range in-between.

6.5 Rendering and Interaction

COVISE contains two separate renderers. An Open Inventor based renderer is responsible for the visualization at the desktop, a Performer based renderer is used for visualization in virtual environments. In the following subsections, both renderers and their extensions

for volume rendering will be discussed. Furthermore, existing functionality of COVISE which contributes to volume rendering will be mentioned.

6.5.1 Basis

Before the extensions, the only way COVISE could display three dimensional scalar fields was with polygonal approaches. There was an iso-surface module, which created surfaces from the data with the marching cubes algorithm, and there was a cutting plane module which displayed the data values which were located on a plane cut through the dataset.

If a data field was stored as an unstructured grid, a sampling module could be used to map this data to a uniform grid. This module could be exploited to allow volume visualization of unstructured grids after a resampling step.

Because COVISE did not support volume rendering, there was no explicit transfer function editor. However, the task of assigning colors and opacities was required to visualize scalar fields with traditional visualization methods. That could be done with the color editor module `ColorEdit`. This module can be used as a transfer function editor without further changes. However, because it is not part of the renderer, it cannot be used for interactive classification. The color editor creates the data type “packed colors”, which was described in Section 6.3.1.

The desktop renderer is based on Open Inventor, wich is a versatile object oriented graphics API. All the graphics objects are embedded in a class hierarchy. The original Open Inventor API did not have a class for volume data or a volume rendering node. The object that was most similar to volume data was `SoCube`, which was derived from `SoShape`.

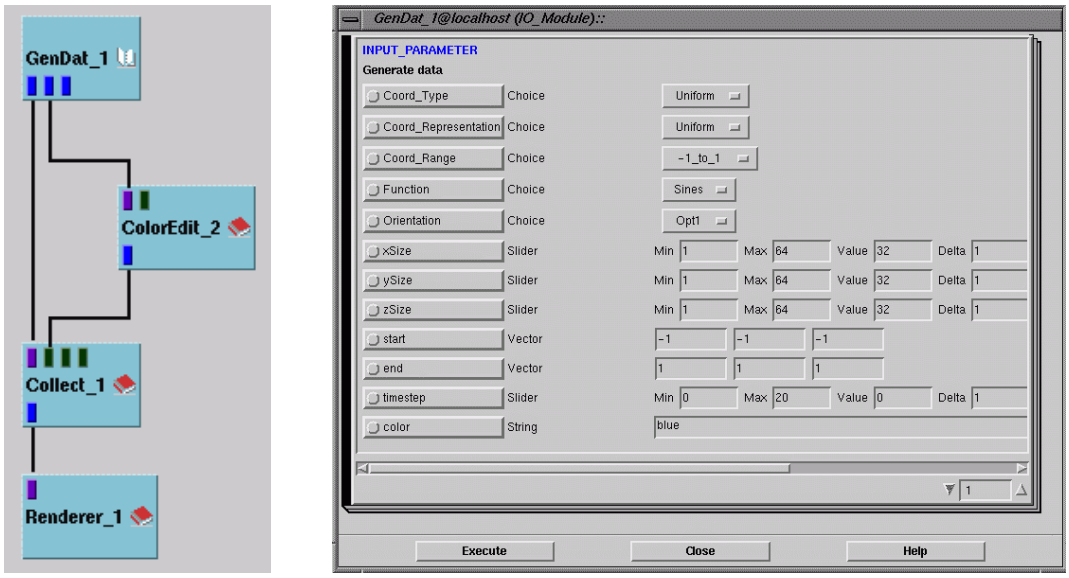
The virtual reality renderer COVER is based on SGI Performer. Performer offers a scene graph and object oriented commands, but its object hierarchy is much less sophisticated than Open Inventor’s. Performer uses parallelization to increase rendering speed on multiprocessor hardware. Three types of processes are distinguished: APP, CULL, and DRAW. The APP process contains the application related code, which includes the modifications of the scene graph. The CULL process traverses the scene graph to find branches which can be left out in the rendering stage because they will be located outside of the viewport. In multi-pipe machines, Performer creates one DRAW process for each pipe. This allows each pipe to render their part of the image independently from the others.

6.5.2 Contributions

This dissertation’s contributions in the fields of rendering and interaction have been described in Chapters 3, 4, and 5. This includes a texturing hardware based and a shear-warp volume renderer, a remote shear-warp renderer which runs on parallel computers, and a variety of interaction methods for the work with volume data in virtual environments.

6.5.3 Integration

All the rendering approaches developed in this dissertation were implemented in Virvo and in the two COVISE renderers. The interaction methods were only integrated in COVER, because they require a virtual environment. The following subsections will describe the integration of the new algorithms in the desktop and in the virtual reality renderer.



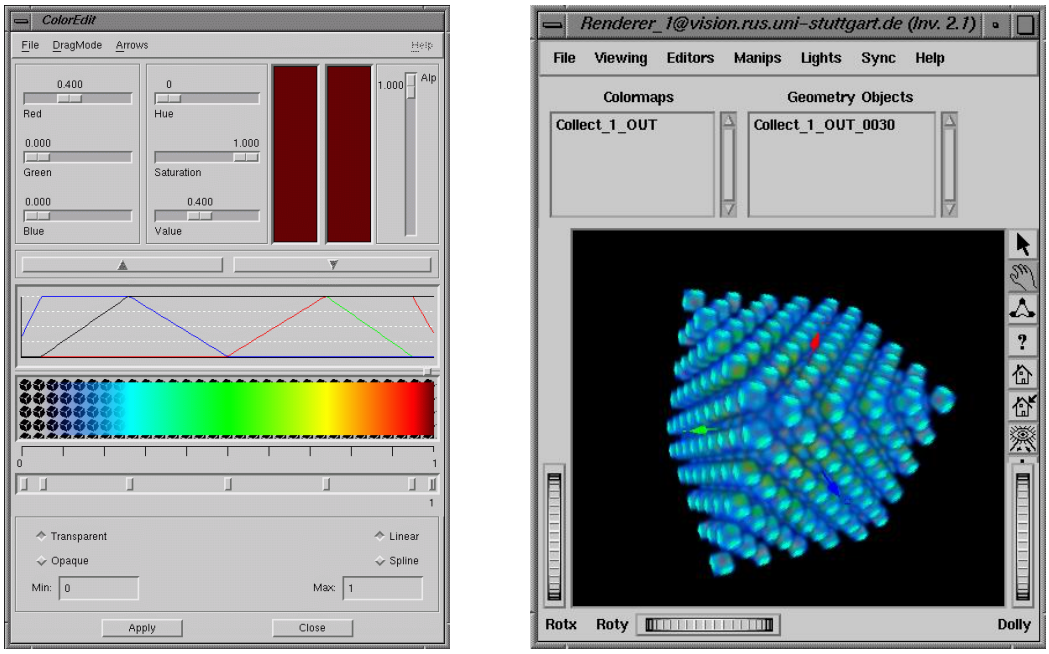
(a) (b)

Figure 6.5: (a) COVISE map with `GenDat` module for volume rendering, (b) `GenDat` parameters.

6.5.3.1 Desktop Renderer

A simple module layout for volume rendering with COVISE and the desktop renderer is depicted in Figure 6.5a. The module `GenDat` generates a scalar data field on a uniform grid, using the parameters shown in Figure 6.5b. `ColorEdit` acts as a transfer function editor, as depicted in Figure 6.6a. To display semi-transparencies correctly, the Transparency check box must be selected. The `ColorEdit` module converts incoming scalar data values to RGBA tuples, which are then passed on to the `Collect` module. This module combines voxel data and grid information and transfers them to the renderer. The renderer output for this module layout is displayed in Figure 6.6b.

For the integration of the new volume rendering algorithms it was beneficial to create a rendering class hierarchy that was independent of the rendering environment, as mentioned in Section 6.1. The rendering classes could be compiled with the Open Inventor files. In order to integrate volume rendering properly with the rest of the rendering routines, a new scene graph node was created, which was called `SoVolume`. It resembles the Open Inventor class `SoCube`, which displays cuboids. The methods that are different



(a) (b)

Figure 6.6: (a) color editor, (b) output in renderer window.

for volume rendering are `init()`, which is called once when the instance is created, and `GLRenderGeneric`. The `init()` method instantiates the Virvo renderer that is most suitable for the computer's graphics hardware, and it transfers the volume data to the renderer. `GLRenderGeneric` calls the renderer's draw routine whenever a redraw is required. Because Open Inventor is based on OpenGL, the viewing matrix is already set correctly when `GLRenderGeneric` is called, and it can be used for displaying the volume.

In the renderer, the volume object is treated like all other COVISE data objects. If both volume data and polygonal data are displayed, occlusion artifacts may occur. For this case, the renderer menu offers several types of transparency sorting.

The renderer offers a specific draw style for volume data: while the data is rotated with the mouse, the volume is drawn at a lower quality to speed up the drawing process, and when the mouse button is released, the volume is displayed at higher quality. This draw mode can be selected in the renderer's pop-up menu. In the preferences window, the sampling rate of the static volume display can be adjusted (see encircled area in Figure 6.7).

6.5.3.2 Virtual Reality Renderer

All the new developments in the fields of rendering and interaction techniques have been integrated into the COVER renderer as a plug-in. User interaction and the interface to COVISE is done in the APP process, while the actual volume rendering is done in the DRAW processes. Due to this design, information has to be exchanged between the APP process and the DRAW processes. This is done via a shared data structure which contains

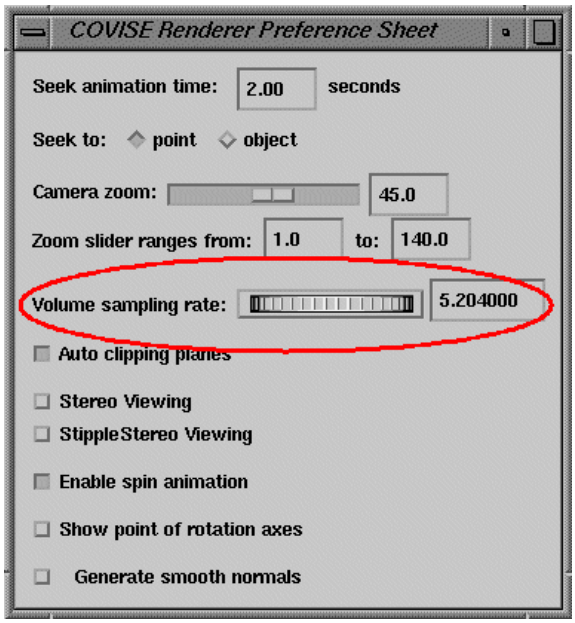


Figure 6.7: Preferences window with volume quality control wheel.

queues and shared states for data objects, the user's position and orientation, and similar data.

The Volume plug-in uses a custom Performer node to represent the volume objects in the scene graph. This allows the usage of Performer's view frustum culling features. Furthermore, draw buckets are used to ensure the correct rendering order, which is important for the concurrent display of volume data and polygonal data.

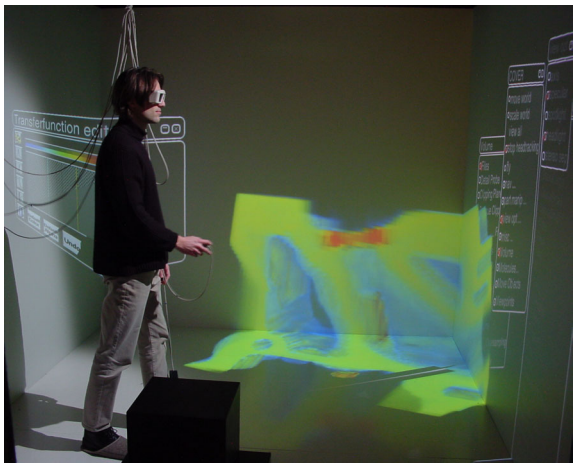


Figure 6.8: The perspective shear-warp algorithm in the CUBE. (See also Color Plate 10 on page 127.)

In COVER, volume data can be loaded in two ways: the first option is to transfer the data with a COVISE network, for example by using the module `ReadVolume`, or by

sampling an arbitrary grid to a uniform grid. The second option is to load a Virvo volume file directly from disk, which can be done from the File menu of the Volume plug-in. This requires, that the desired files be listed in the section `VolumeFiles` in the configuration file of COVISE.

Remote rendering on parallel computers can be selected in COVISE's configuration file. When it is enabled, each rendering process waits for a connection to the remote renderer upon startup. On the parallel computer, one remote renderer process has to be started for each rendering process on the display computer. COVER only proceeds after all remote connections have been established. As an example, Figure 6.8 shows a shear-warp rendering of the Engine dataset [24] in the CUBE at the HLRS. The four displays of the CAVE-like environment are driven by a cluster of four commodity PCs. The volume is composited on 12 processors of an Onyx2.

Chapter 7

Conclusions

The goal of this dissertation is to improve interactive volume rendering in virtual environments. To achieve this goal, research has been conducted in the fields of rendering algorithms, interaction techniques, and distribution methods.

In the field of rendering algorithms, an extension of the CPU-based shear-warp algorithm has been presented, which allows perspective projection. This has been a requirement to use the algorithm in virtual environments. The algorithm has also been examined for optimizations for its usage in CAVE-like virtual environments.

To further improve the image quality of the shear-warp algorithm, pre-integrated rendering was integrated into the compositing stage. Pre-integration imposes a noticeable performance hit on the standard shear-warp algorithm, but it results in substantially improved image quality.

User interaction with volume datasets in virtual environments has been substantially improved by a variety of developments. These include interaction elements for adjusting the transfer function, a detail probing mode, and a clipping plane. After two user studies, the system was attested fairly good usability.

Due to the specific demands of virtual environments, the rendering performance of the perspective projection shear-warp algorithm had to be improved. An implementation for parallel computers has been developed, which can be linked to a display computer via a network connection. Any architecture that supports MPI can be used as a platform for the remote renderer. The remote rendering process scales well up to 12 processors, depending on the parallel hardware. The transfer speed of the remotely computed images has been optimized, and the remotely rendered images can be displayed on a computer with any type of graphics hardware.

All the new developments have been successfully integrated into the visualization system COVISE, which did not previously support volume rendering. Now the users can work with volume datasets in the entire data flow, from loading volume data files from disk to interactive work in virtual environments.

In this chapter, Section 7.1 shows that the work presented in this dissertation address all the stages of interactive volume rendering. Section 7.2 analyzes the scientific relevance of the conducted research. Topics for future works are proposed in Section 7.3.

7.1 Interactive Volume Rendering

Figure 7.1 shows a flowchart of an interactive volume rendering system, as it could be created from the algorithms and approaches described in this dissertation. The chart considers local rendering with texture hardware, local rendering with the shear-warp algorithm, and remote rendering on a parallel computer with the shear-warp algorithm. The diagram is independent from the type of display device connected to the display computer (single or multi-screen). The part of the diagram left of the dashed line represents activities on the display computer, the part right of it contains activities on the remote parallel computer. The boxes on top of the dashed line represent data transfer between the two computers. The bracketed numbers denote the chapters in which more detailed information can be found about the respective activity.

The flowchart can be interpreted for three different volume rendering methods: local texture hardware based, local shear-warp based, and remote shear-warp based. The decision between them depends on the availability of a parallel computer and the capabilities of the graphics hardware on the display computer (see Figure 7.2). The rendering method cannot be changed after it has been chosen.

In the following three sections, the suggested volume rendering methods will be described separately.

7.1.1 Local Rendering with Texturing Hardware

At startup, the system loads a volume dataset from disk into memory. The dataset is then transferred to the texturing hardware. If multiple graphics cards or pipes are used in a multi-screen environment, each of them receives a copy of the dataset. Then the polygonal objects of the scenegraph are rendered. After that, the volume is rendered.

Then user input is being processed. The user can change the transfer functions, explore the dataset, or save the dataset and its transfer function to disk. Exploration includes probe mode and clipping plane, but also movements of the head (if head tracking is available) or input from the pointing device. After the user input has been processed, the rendering loop starts over with rendering the polygonal objects.

7.1.2 Local Rendering with the Shear-Warp Algorithm

If no parallel computer and no texturing acceleration is available, volume rendering can be done with the shear-warp algorithm on the display computer. After the volume dataset has been loaded from disk, it is replicated and permuted such that it is available in a format which the shear-warp algorithm can process efficiently. If multiple graphics pipes are used, each of them needs to receive a copy of the dataset. Then the polygonal objects are rasterized, the volume dataset is rendered locally using the shear-warp algorithm, and the intermediate image is warped either by texturing hardware, or with a CPU-based algorithm.

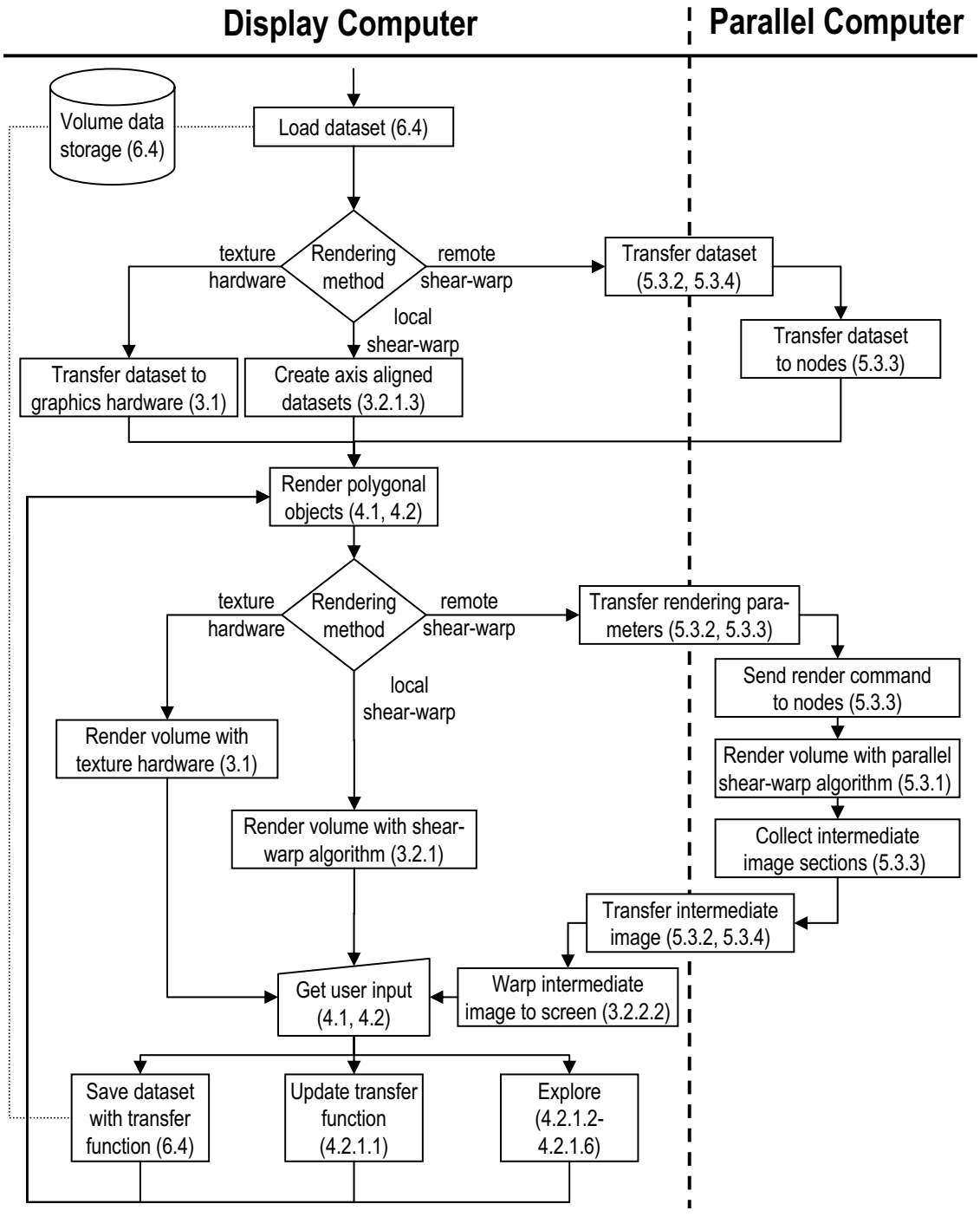


Figure 7.1: Flowchart of an interactive volume rendering system. The numbers in brackets denote the sections in which the respective processes are discussed.

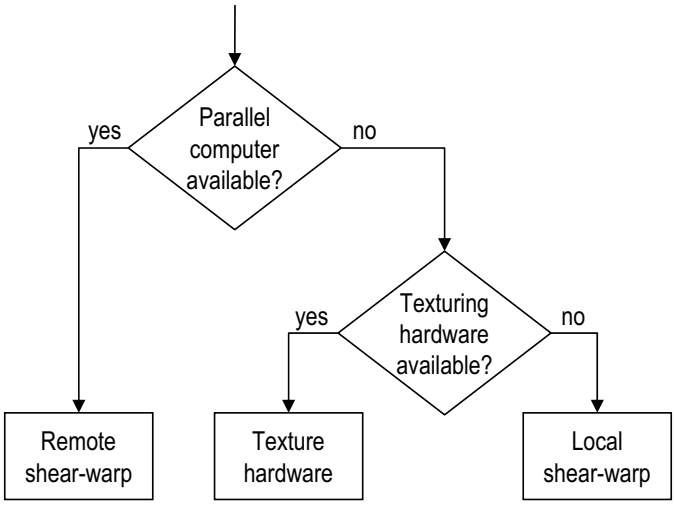


Figure 7.2: Flowchart for the decision on the rendering method.

7.1.3 Remote Rendering on a Parallel Computer

When a parallel computer is available for remote volume rendering, the system is executed in a slightly different way. After loading the dataset from disk, it is transferred via a network connection to the master node of the remote parallel computer. The master node then distributes the dataset to all its rendering nodes, such that each node obtains a copy of the same dataset. After that the interaction loop is entered. The user input is processed in the same way as above. However, now the viewing parameters have to be transferred to the parallel computer via the network. The parallel computer renders the intermediate image with the shear-warp algorithm. Then the intermediate image is compressed and transferred to the display computer. There it is uncompressed and either copied to texture memory and warped by the graphics hardware, or warped with a CPU-based algorithm.

7.2 Relevance Analysis

This section discusses the scientific relevance of three major contributions of this dissertation: the virtual reality user interface, the volume rendering concept and its integration into a visualization system, and remote volume rendering based on the shear-warp algorithm.

7.2.1 User Interface

The volume rendering user interface presented in Section 4.2 is the first published UI for virtual environments which can be used with any 3D input device. At the time of its publication, the only comparable interface provided by the Studierstube [77] required a 3D pen and a custom-made transparent panel, which occupied both hands of the user.

The widget library presented in Section 4.1 has originally been conceived to allow the creation of a volume rendering application for virtual environments. However, it proved to be versatile enough to be used in all virtual reality applications developed at the HLRS. In the meantime, several extensions have been added by other members of the visualization group, for instance a customizable toolbar, a collapse function for menus, and improved layout managing.

7.2.2 Volume Rendering Concept

As mentioned in Chapter 6, most of the developments presented in this dissertation have been integrated into the visualization system COVISE. Some of the algorithms have only been integrated because their development required a virtual reality framework, but they have not become a permanent part of the visualization software. The majority of the developments, however, are available to all COVISE users.

Due to the research carried out for this dissertation, COVISE can now work with volumetric data throughout its entire data chain. Previously, three-dimensional scalar fields could be handled as data on a uniform grid, but there was no way to directly visualize this grid type as a whole. Data on uniform grids can now be displayed with direct volume rendering, and it can be combined with all previous visualization methods of the system.

The author's experience with customers using COVISE's volume rendering capabilities so far is predominantly positive. The volume rendering functionality is mostly employed by research institutes, and in the automotive and oil and gas industries. Most customers are happy with the performance of texture-based volume rendering. Some of them require the simultaneous and overlapping display of volume and polygonal data. There have been no complaints about the transfer function editor being based on widgets instead of the traditional approach of functions with control points. The detail probe mode is typically adopted quickly by the users.

7.2.3 Remote Volume Rendering

The shear-warp-based remote volume rendering technique presented in Chapter 5 is an efficient volume rendering method, but only if a parallel computer is available and it is connected to the display computer via a fast network connection. In the past few years, commodity graphics cards have improved at a much faster pace than CPU technology. Therefore, the necessity for parallel computers to be used as remote volume rendering servers has decreased. However, the advances in graphics processors have been only a recent phenomenon, while CPU power has grown at a constant rate for a much longer time, so parallel computers might become more interesting for real-time volume rendering again in the future.

Today, remote volume rendering is particularly useful for interactive, but non-real-time applications. While graphics cards have fixed accuracy and limited memory, parallel computers can deal with much higher accuracy and larger datasets.

In simulations at the Astronomy Department of the Centre for Astrophysics and Supercomputing of the Swinburne University of Technology at Hawthorn (Australia) large three-dimensional scalar fields are generated, which do not fit into the memory of today's graphics cards [9]. Therefore, the parallelized version of the perspective shear-warp algorithm, which was developed for this dissertation, has been installed on the Linux PC cluster at Hawthorn in collaboration with the author. Furthermore, the astronomers created a TCL/Tk user interface and integrated it with the simulation software. Among other features, the user interface consists of a transfer function editor with the same widgets as those described in this dissertation. Using this system, the astronomers have been able to visualize their results interactively.

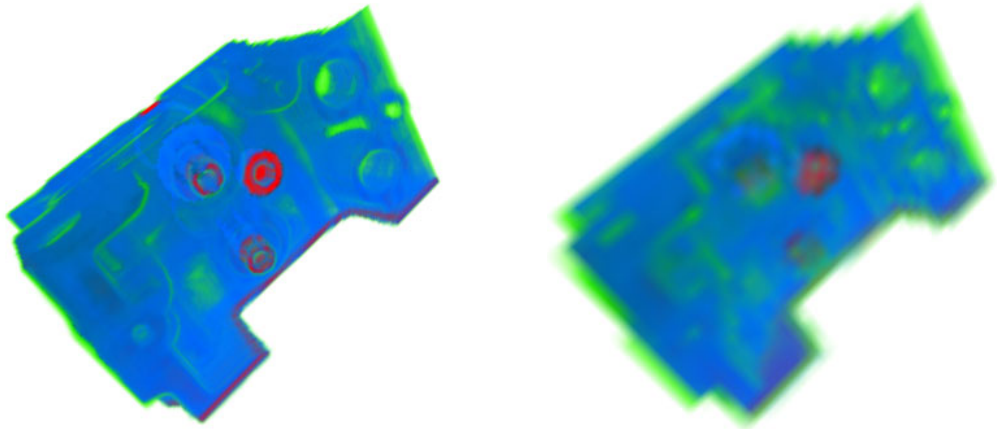
7.3 Future Work

Despite the research conducted for this dissertation, volume rendering remains one of the most compute intensive visualization tasks. Although faster CPUs and faster graphics hardware will increase image quality and rendering performance, future work still needs to address the optimization of the algorithms. For example, the new approaches could be further improved by integrating level-of-detail strategies or intelligent data reduction methods. Optimized load balancing would accelerate the parallelized shear-warp algorithm, albeit only to a limited extent as it was shown. Also, a parallelization of the intermediate image compression algorithm would be beneficial.

In the future, hybrid memory approaches will become the dominant architecture for special purpose parallel computers. The new volume rendering algorithms could take advantage of this development by adding OpenMP support and by restricting MPI usage to processes on separate nodes.

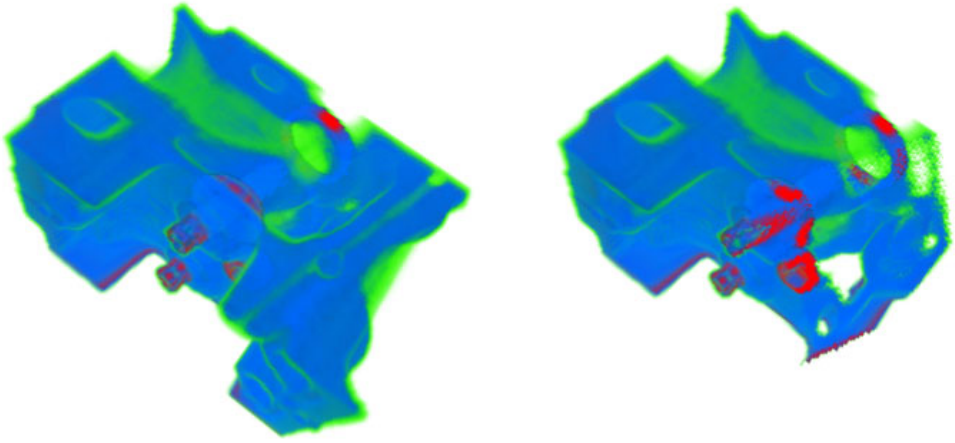
Interaction with volume data in virtual environments could be improved by a more diverse set of direct interaction techniques and more functionality for the transfer function editor. Specialization to the specific requirements of certain professional groups was requested by some of the industrial participants in the user studies.

Color Plates



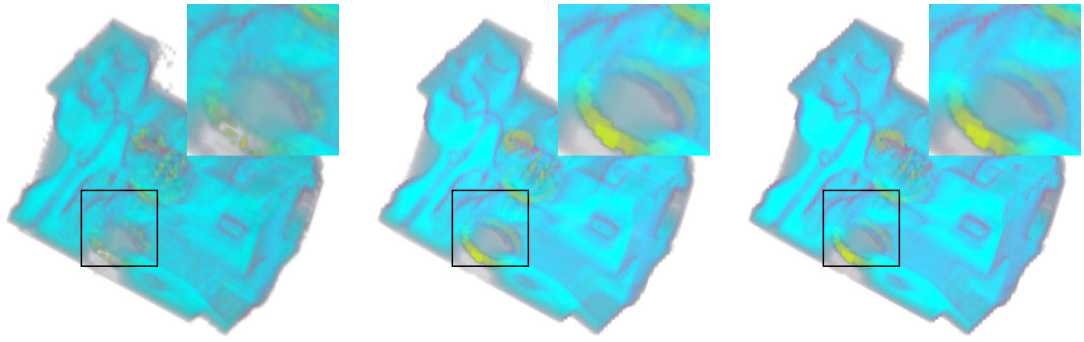
(a) (b)

Color Plate 1: Intermediate image size: (a) 2048^2 and (b) 256^2. (See also Figure 3.3 on page 52.)



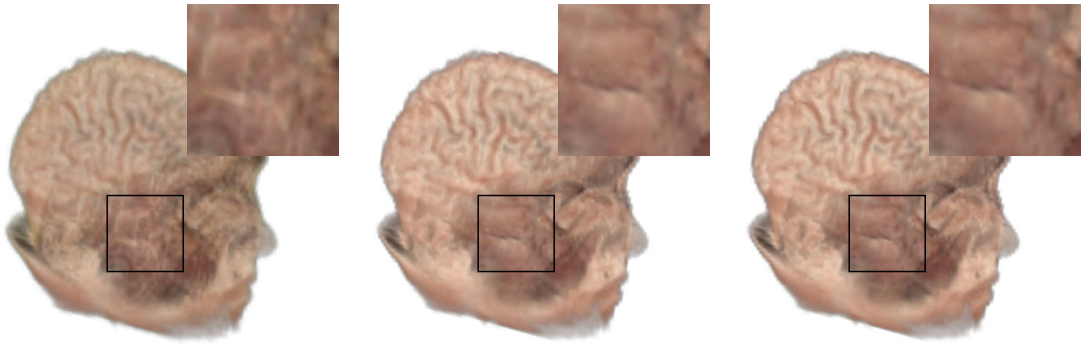
(a) (b)

Color Plate 2: Engine dataset: (a) complete, (b) clipped. (See also Figure 3.4 on page 53.)



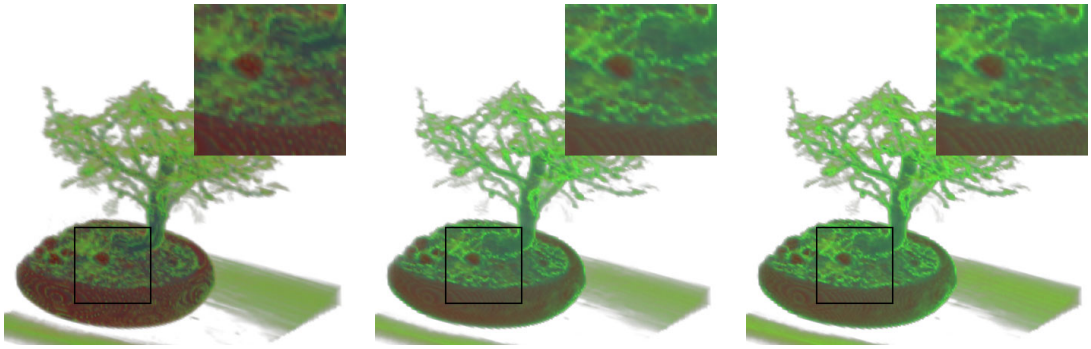
(a) (b) (c)

Color Plate 3: The Engine dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction. (See also Figure 3.11 on page 63.)



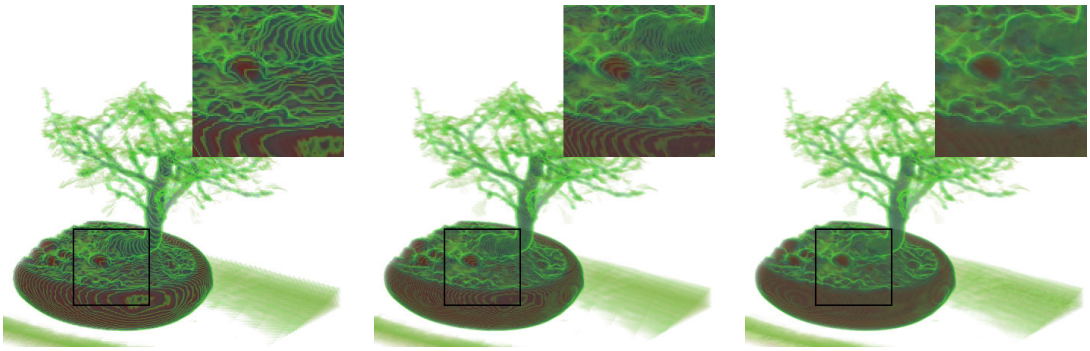
(a) (b) (c)

Color Plate 4: The Brain dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction. (See also Figure 3.12 on page 65.)



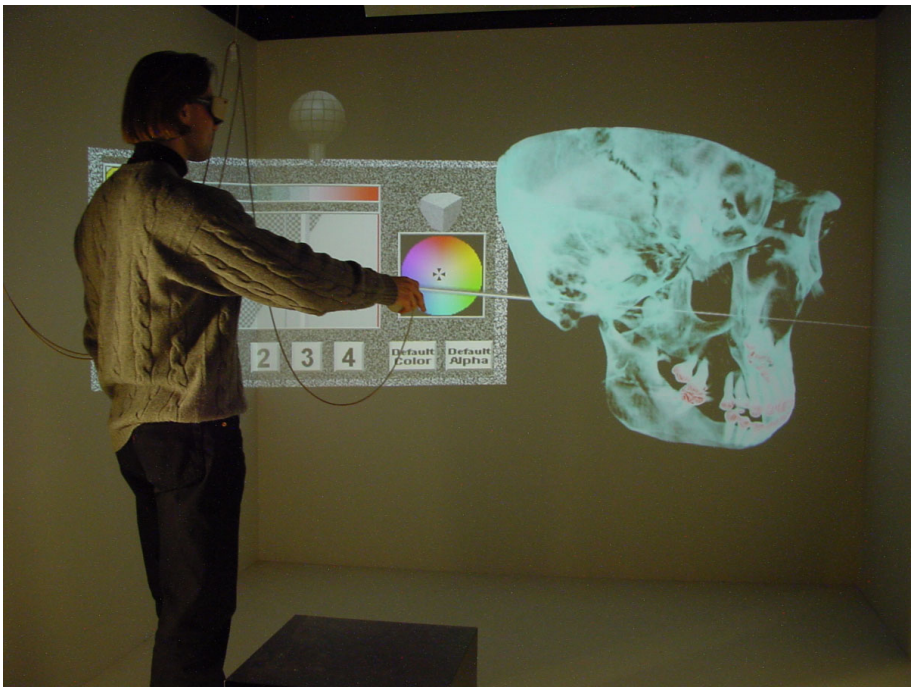
(a) (b) (c)

Color Plate 5: The Bonsai dataset: (a) standard shear-warp, (b) pre-integrated shear-warp without opacity correction and (c) pre-integrated shear-warp with opacity correction. (See also Figure 3.13 on page 65.)

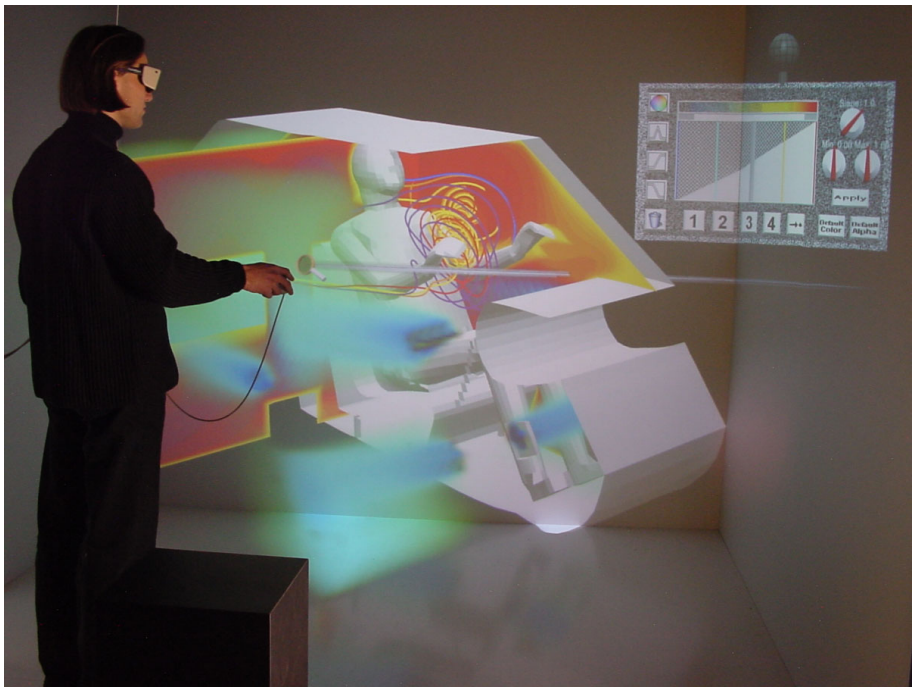


(a) (b) (c)

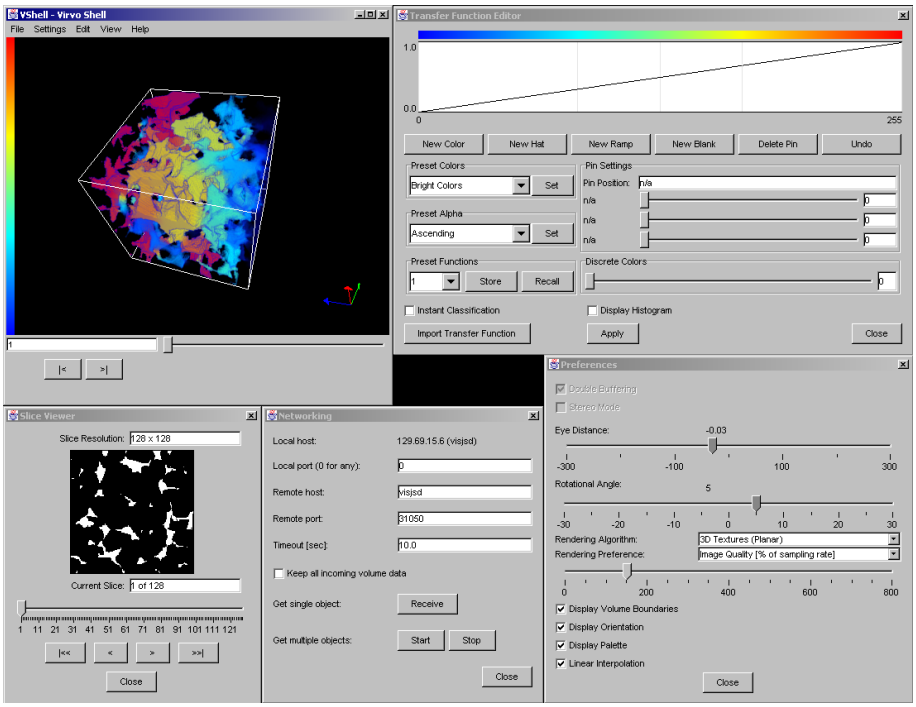
Color Plate 6: The Bonsai dataset rendered with 3D texturing hardware support using different numbers of textured polygons: (a) 128 polygons, (b) 256 polygons, (c) 1024 polygons. (See also Figure 3.14 on page 65.)



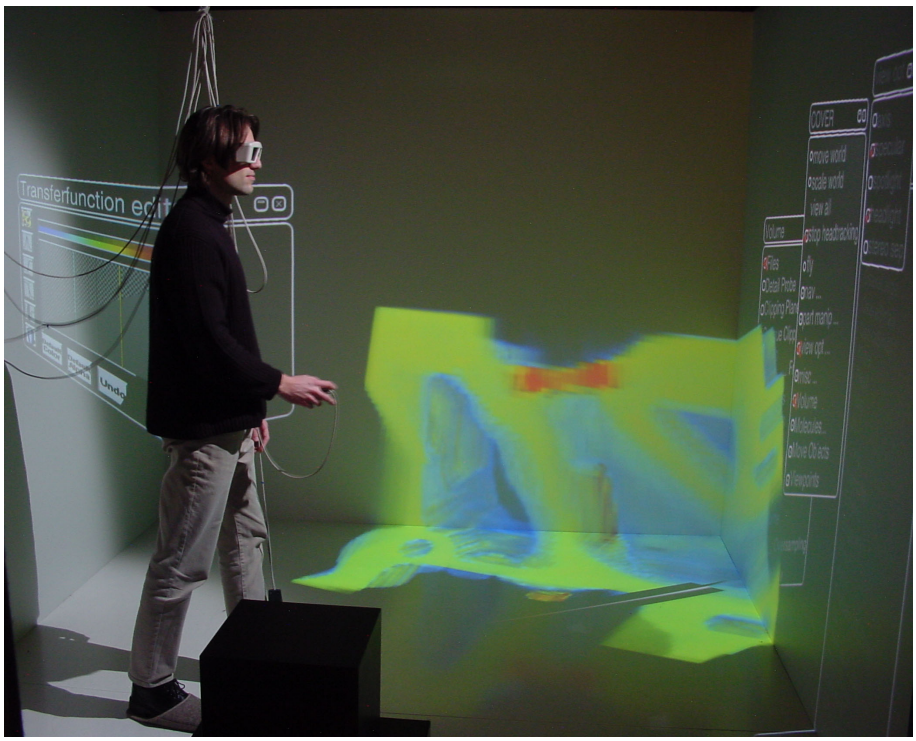
Color Plate 7: The skull of the Visible Human in the CAVE. (See also Figure 4.15 on page 79.)



Color Plate 8: Volume rendered temperature distribution in a polygonal car cabin. (See also Figure 4.16 on page 80.)



Color Plate 9: The rendering front-end VShell. (See also Figure 6.1 on page 104.)



Color Plate 10: The perspective shear-warp algorithm in the CUBE. (See also Figure 6.8 on page 114.)

Bibliography

[1] Java 3D. *Java 3D API home page*. URL: http://java.sun.com/products/java-media/3D/.

[2] K. Akeley. *RealityEngine Graphics*. ACM SIGGRAPH 93 Proceedings, pp. 109–116, 1993.

[3] M.B. Amin, A. Grama, and V. Singh. *Fast Volume Rendering Using an Efficient, Scalable Parallel Formulation of the Shear-Warp Algorithm*. Proceedings of the IEEE Parallel Rendering Symposium, October '95, Atlanta, pp. 7–14, 1995.

[4] K. Anagnostou, T.J. Atherton, and A.E. Waterfall. *4D Volume Rendering With The Shear Warp Factorisation*. Proceedings of the IEEE/ACM Symposium on Volume Visualization, pp. 129–137, 2000.

[5] AVS. *Visualization system, Advanced Visualization Systems, Waltham, MA*. URL: http://www.avs.com.

[6] Java AWT. *Java Abstract Window Toolkit documentation home page*. URL: http://java.sun.com/j2se/1.4.1/docs/guide/awt/.

[7] C.L. Bajaj, V. Pascucci, and D.R. Schikore. *The Contour Spectrum*. IEEE Visualization '97 Proceedings, pp. 167–173, 1997.

[8] J. Beckmann. *3D Interaktionselemente als Benutzerschnittstelle in virtuellen Umgebungen*. Diplomarbeit, Department of Computer Science, University of Erlangen-Nürnberg, Germany, 1999.

[9] B. Beeson, D.G. Barnes, P.D. Bourke, and N. Killeen. *A Distributed-Data Implementation of the Perspective Shear-Warp Volume Rendering Algorithm for Visualization of Large Astronomical Cubes*. White Paper, Centre for Astrophysics and Supercomputing, Swinburne University of Technology, Hawthorn, Australia, URL: http://astronomy.swin.edu.au/staff/bbeeson/dvr/whitepaper.pdf, 2003.

[10] N. Bevan, J. Kirakowski, and J. Maissel. *What is Usability?* In: H.J. Bullinger (Ed.): Proceedings of the 4th International Conference on Human Computer Interaction, Stuttgart, September 1991, Elsevier, 1991.

[11] R. Brady, J. Pixton, G. Baxter, P. Moran, C. S. Potter, B. Carragher, and A. Belmont. *Crumbs: A Virtual Environment Tracking Tool for Biological Imaging*. IEEE Symposium on Frontiers in Biomedical Visualization Proceedings, pp. 18–25, 1995.

[12] B. Cabral, N. Cam, and J. Foran. *Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware*. ACM Symposium on Volume Visualization '94, pp. 91–98, 1994.

[13] B. Chen, F. Dachille, and A.E. Kaufman. *Forward Image Warping*. IEEE Visualization '99 Proceedings, IEEE Computer Society Press, pp. 89–86, 1999.

[14] S. Coquillart and G. Wesche. *The Virtual Palette and the Virtual Remote Control Panel: A Device and an Interaction Paradigm for Projection-Based Virtual Environments*. IEEE Virtual Reality '99 Proceedings, 1999.

[15] COVISE. *On-line COVISE documentation at the web site of the HLRS*. URL: http://www.hlrs.de/organization/vis/covise/support/documentation/.

[16] COVISE. *Visualization system for scientific and engineering data, VirCinity GmbH, Stuttgart, Germany*. URL: http://www.vircinity.com.

[17] C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti. *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*. ACM SIGGRAPH 93 Proceedings, pp. 135–142, 1993.

[18] B. Csebfalvi. *Fast Volume Rotation using Binary Shear-Warp Factorization*. Eurographics Data Visualization '99 Proceedings, pp. 145–154, 1999.

[19] T.J. Cullip and U. Neumann. *Accelerating Volume Reconstruction With 3D Texture Hardware*. Technical Report TR93-027, University of North Carolina, Chapel Hill, 1993.

[20] M. Czernuszenko, D. Pape, D.J. Sandin, T.A. DeFanti, G.L. Dawe, and M. Brown. *The ImmersaDesk and Infinity Wall Projection-Based Virtual Reality Displays*. Computer Graphics, Volume 31, Number 2, pp. 46–49, 1997.

[21] J.M. Danskin and P. Hanrahan. *Fast Algorithms for Volume Ray Tracing*. ACM Workshop on Volume Visualization '92, pp. 91–98, 1992.

[22] Bonsai dataset. *Courtesy of Stefan Röttger from the Computer Graphics Group at the University of Stuttgart and Bernd Tomandl from the Department of Neurosurgery at the University of Erlangen*. URL: http://wwwvis.informatik.uni-stuttgart.de/∼roettger/data/Bonsai/.

[23] Brain dataset. *From the Chapel Hill Volume Rendering Test Datasets*. URL: http://www.siggraph.org/education/materials/vol-viz/mrbrain.zip.

[24] Engine dataset. *CT engine scan by General Electric, part of the Chapel Hill Volume Rendering Test Datasets*. URL: http://www.gris.uni-tuebingen.de/areas/scivis/volren/datasets/data/engine.raw.gz.

[25] R.A. Drebin, L. Carpenter, and P. Hanrahan. *Volume Rendering*. Computer Graphics, Volume 22, Number 4, pp. 125–134, 1988.

[26] K. Engel, M. Kraus, and T. Ertl. *High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading*. Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '01, Addison-Wesley Publishing Company, pp. 9–16, 2001.

[27] EnSight. *Visualization system, Computational Engineering International (CEI), Apex, NC*. URL: http://www.ensight.com.

[28] FLTK. *Fast Light Toolkit*. URL: http://www.fltk.org.

[29] M. Flynn. *Very High-Speed Computing Systems*. Proceedings of the IEEE, Volume 54, pp. 1901–1909, 1966.

[30] F. Föhl. *Geräte- und szenengraphunabhängige grafische Benutzungselemente für VR-Anwendungen*. Diplomarbeit, Institut für Informatik, University of Stuttgart, Germany, 2002.

[31] J. Freund and K. Sloan. *Accelerated Volume Rendering Using Homogeneous Region Encoding*. IEEE Visualization '97 Proceedings, pp. 191–197, 1997.

[32] T. Günther, C. Poliwoda, C. Reinhard, J. Hesser, R. Männer, H.-P. Meinzer, and H.-J. Baur. *VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine*. Proceedings of the 9th Eurographics Workshop on Graphics Hardware, Oslo, Norway, pp. 103–108, 1994.

[33] P. Gußmann. *Erstellung eines verteilten, synchronisierten und kooperativen Renderers*. Diplomarbeit, Institut für Informatik, University of Stuttgart, Germany, 2000.

[34] S. Guthe and W. Straßer. *Real-Time Decompression and Visualization of Animated Volume Data*. IEEE Visualization '01 Proceedings, pp. 349–356, 2001.

[35] T. He, L. Hong, A. Kaufman, and H. Pfister. *Generation of Transfer Functions with Stochastic Search Techniques*. Proceedings of IEEE Visualization '96, pp. 227–234, 1996.

[36] T. He and A. Kaufman. *Fast Stereo Volume Rendering*. IEEE Visualization '96 Proceedings, pp. 49–56, 1996.

[37] L.C. Hill and C. Cruz-Neira. *Palmtop Interaction Methods for the Immersive Projection Technology VR Systems*. Proceedings of the 4th Immersive Projection Technology Workshop (IPTW), Iowa State University, 2000.

[38] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. *WireGL: A Scalable Graphics System for Clusters*. ACM SIGGRAPH 2001 Proceedings, 2001.

[39] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P.D. Kirchner, and J.T. Klosowski. *Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters*. ACM SIGGRAPH 2002 Proceedings, pp. 693–702, 2002.

[40] Open Inventor. *Visualization API, Silicon Graphics, Inc., Mountain View, CA*. URL: http://www.sgi.com/software/inventor/.

[41] D. Jiang and J.P. Singh. *Improving Parallel Shear-Warp Volume Rendering on Shared Address Space Multiprocessors*. ACM Symposium on Principles and Practice of Parallel Programming Proceedings, Las Vegas, pp. 252–263, 1997.

[42] G. Kindlmann. *Nearly Raw Raster Data, volume data format*. URL: http://www.cs.utah.edu/∼gk/teem/nrrd/.

[43] G. Kindlmann and J.W. Durkin. *Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering*. Proceedings of the IEEE Symposium on Volume Visualization '98, pp. 79–86, 1998.

[44] J. Kniss, G. Kindlmann, and C. Hansen. *Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets*. IEEE Visualization '01 Proceedings, pp. 255–262, 2001.

[45] G. Knittel. *The UltraVis System*. Proceedings of IEEE/ACM Symposium on Volume Visualization, pp. 71–78, 2000.

[46] G. Knittel. *Using Pre-Integrated Transfer Functions in an Interactive Software System for Volume Rendering*. Short Papers Proceedings Eurographics '02, pp. 119–123, 2002.

[47] G. Knittel and W. Straßer. *Vizard - Visualization Accelerator for Real-Time Display*. Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '97, pp. 139–147, 1997.

[48] W. Krueger. *The application of transport theory to visualization of 3-D scalar data fields*. Computers in Physics, July/August, pp. 397–406, 1991.

[49] J. Krüger and R. Westermann. *Acceleration Techniques for GPU-based Volume Rendering*. IEEE Visualization '03 Proceedings (to appear), 2003.

[50] W. Krüger and B. Fröhlich. *The Responsive Workbench*. Computer Graphics and Applications 14(3), pp. 12–15, 1994.

[51] P. Lacroute. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. Doctoral Dissertation, Technical Report CSL-TR-95-678, Stanford University, 1995.

[52] P. Lacroute. *Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization*. IEEE Parallel Rendering Symposium '95 Proceedings, pp. 15–22, 1995.

[53] P. Lacroute and M. Levoy. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. ACM SIGGRAPH 94 Proceedings, pp. 451–457, 1994.

[54] E. LaMar, B. Hamann, and K.I. Joy. *Multiresolution Techniques for Interactive Texture-Based Volume Visualization*. IEEE Visualization '99 Proceedings, pp. 355–361, 1999.

[55] M. Levoy. *Display of Surfaces from Volume Data*. Computer Graphics and Applications, Vol. 8, No. 5, May, pp. 29–37, 1988.

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

[57] R.W. Lindeman, J.L. Sibert, and J.K. Hahn. *Towards Usable VR: An Empirical Study of User Interfaces for Immersive Virtual Environments*. CHI 99, 15-20 May, 1999.

[58] W.E. Lorensen and H.E. Cline. *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. ACM Computer Graphics Volume 21, Number 4, July, pp. 93–99, 1987.

[59] T. Malzbender. *Fourier Volume Rendering*. ACM Transactions on Graphics, 12(3), pp. 233–250, 1993.

[60] J. Marks, B. Andalman, P.A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. *Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation*. ACM SIGGRAPH '97 Proceedings, pp. 389–400, 1997.

[61] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. *A Practical Evaluation of Popular Volume Rendering Algorithms*. Proceedings of IEEE/ACM Symposium on Volume Visualization, pp. 81–90, 2000.

[62] S. Muraki. *Approximation and Rendering of Volume Data Using Wavelet Transforms*. IEEE Visualization '92 Proceedings, pp. 21–28, 1992.

[63] OpenDX. *Open source software project based on IBM's Visualization Data Explorer*. URL: http://www.opendx.org.

[64] OpenGL Optimizer. *Visualization API, Silicon Graphics, Inc., Mountain View, CA*. URL: http://www.sgi.com/software/optimizer/.

[65] S. Parker, M. Parker, Y. Livnat, P. Sloan, C. Hansen, and P. Shirley. *Interactive Ray Tracing for Volume Visualization*. IEEE Transactions on Visualization and Computer Graphics, Volume 5, pp. 238–250, 1999.

[66] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. *The VolumePro Real-Time Ray-Casting System*. ACM SIGGRAPH 99 Proceedings, pp. 251–260, 1999.

[67] H. Pfister and A. Kaufman. *Cube-4 - A Scalable Architecture for Real-Time Volume Rendering*. ACM/IEEE Symposium on Volume Visualization '96, pp. 47–54, 1996.

[68] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Sobierajski Avila, K. Martin, R. Machiraju, and J. Lee. *The Transfer Function Bake-Off*. IEEE Computer Graphics and Applications, May/June 2001 (Vol. 21, No. 3), pp. 16–22, 2001.

[69] T.J. Purcell, I. Buck, W.R. Mark, and P. Hanrahan. *Ray Tracing on Programmable Graphics Hardware*. ACM SIGGRAPH '02 Proceedings, pp. 703–712, 2002.

[70] Trolltech Qt. *GUI library*. URL: http://www.trolltech.com.

[71] C.R. Ramakrishnan and C. Silva. *Optimal Processor Allocation for Sort-Last Compositing under BSP-tree Ordering*. SPIE Visual Data Exploration and Analysis IV, 1999.

[72] D. Rantzau, K. Frank, U. Lang, D. Rainer, and U. Wössner. *COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data*. Proceedings of 2nd Workshop on Immersive Projection Technology (IPTW '98), Ames, Iowa, 1998.

[73] D. Rantzau, U. Lang, and R. Rühle. *Collaborative and Interactive Visualization in a Distributed High Performance Software Environment*. Proceedings of International Workshop on High Performance Computing for Graphics and Visualization, Swansea, Wales, '96, 1996.

[74] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. *Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization*. Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '00, 2000.

[75] J. Rohlf and J. Helman. *IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics*. ACM SIGGRAPH 94 Proceedings, pp. 381–394, 1994.

[76] S. Röttger, S. Guthe, D. Weiskopf, and T. Ertl. *Smart Hardware-Accelerated Volume Rendering*. Procceedings of EG/IEEE TCVG Symposium on Visualization, Grenoble, France, 2003.

[77] D. Schmalstieg and M. Encarnacao. *A Transparent Personal Interaction Panel for the Virtual Table*. Computergraphik Topics, Vol. 10., No. 5, Darmstadt, Germany, pp. 19–20, 1998.

[78] J. Schneider and R. Westermann. *Compression Domain Volume Rendering*. IEEE Visualization '03 Proceedings (to appear), 2003.

[79] M. Schreier. *An Audio Server for Virtual Reality Applications*. Master's Thesis, Brunel University, England, May, 2002.

[80] J.P. Schulze. *Virvo - Virtual Reality Volume Rendering*. Project Homepage at URL: http://www.hlrs.de/people/schulze/virvo/, 2003.

[81] J.P. Schulze, M. Kraus, U. Lang, and T. Ertl. *Integrating Pre-Integration into the Shear-Warp Algorithm*. Proceedings of the Third International Workshop on Volume Graphics (VG'03), Tokyo, July 7-8, Published by ACM, pp. 109–118, 2003.

[82] J.P. Schulze and U. Lang. *The Parallelization of the Perspective Shear-Warp Volume Rendering Algorithm*. Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization (EGPGV'02), ACM Press, pp. 61–69, 2002.

[83] J.P. Schulze and U. Lang. *The Parallelized Perspective Shear-Warp Algorithm for Volume Rendering*. Parallel Computing 29, Published by Elsevier Science, pp. 339–354, 2003.

[84] J.P. Schulze, R. Niemeier, and U. Lang. *The Perspective Shear-Warp Algorithm in a Virtual Environment*. Proceedings of IEEE Visualization '01, Published by IEEE, pp. 207–213, 2001.

[85] J.P. Schulze, U. Wössner, S.P. Walz, and U. Lang. *Volume Rendering in a Virtual Environment*. Proceedings of the Fifth Immersive Projection Technology Workshop (IPTW'01) and Eurographics Virtual Environments (EGVE'01), Springer Verlag, pp. 187–198, 2001.

[86] C.W. Sensen, J. Brum, P. Gordon, M. Hood, G. Lindahl, M. Schulman, C. Spindler, M. Stuart, and S. Unger. *Establishment of the First Java 3D Enabled CAVE*. Whitepaper, URL: http://www.visualgenomics.ca/PDF_files/whitepaper.pdf, 2002.

[87] C. Silva, A. Kaufman, and C. Pavlakos. *PVR: High Performance Volume Rendering*. IEEE Computational Science and Engineering, Special Issue on Visual Supercomputing, Winter '96, pp. 18–28, 1996.

[88] H.A. Sovizral and M.F. Deering. *The Java 3D API and Virtual Reality*. IEEE Computer Graphics and Applications May/June '99, 1999.

[89] S. Spitzer, M.J. Ackerman, A.L. Scherzinger, and D. Whitlock. *The Visible Human Male: A Technical Report*. Journal of the American Medical Informatics Association, pp. 118–130, 1996.

[90] K. Swartz, U. Thakkar, D. Hix, and R. Brady. *Evaluating the Usability of Crumbs: a Case Study of VE Usability Engineering*. Proceedings of the 3rd International Immersive Projection Technologies Workshop, May '99, Springer-Verlag, pp. 243–252, 1999.

[91] J. Sweeney and K. Mueller. *Shear-Warp Deluxe: The Shear-Warp Algorithm Revisited*. EG/IEEE TCVG Symposium on Visualization, May '02, Barcelona, Spain, pp. 95–104, 2002.

[92] T. Totsuka and M. Levoy. *Frequency Domain Volume Rendering*. ACM SIGGRAPH 93 Proceedings, pp. 271–278, 1993.

[93] R. Troutman, C. Hansen, M. Krogh, J. Painter, and G.C. de Verdiere. *Binary-Swap and Shear-Warp Volume Renderer on the T3D*. Proceedings of the Thirty-Sixth Semi-Annual Cray User Group Meeting, Fairbanks, Alaska, pp. 121–125, 1995.

[94] OpenGL Volumizer. *Volume visualization API, Silicon Graphics, Inc., Mountain View, CA*. URL: http://www.sgi.com/software/volumizer/.

[95] W.R. Volz. *Gigabyte Volume Viewing Using Split Software/Hardware Interpolation*. Proceedings of IEEE/ACM Symposium on Volume Visualization, pp. 15–22, 2000.

[96] VTK. *Open source graphics API for 3D graphics, image processing, and visualization*. URL: http://public.kitware.com/VTK/.

[97] M. Weiler, R. Westermann, C. Hansen, K. Zimmermann, and T. Ertl. *Level-Of-Detail Volume Rendering via 3D Textures*. IEEE Volume Visualization 2000 Proceedings, 2000.

[98] J. Welling. *VFleet*. URL: http://www.psc.edu/Packages/VFleet_Home/.

[99] S. Wergandt. *Selektion, Extraktion und Aufbearbeitung von Volumendaten auf Multiprozessor-Systemen*. Studienarbeit, High Performance Computing Center (HLRS), University of Stuttgart, 2002.

[100] R. Westermann. *A Multiresolution Framework for Volume Rendering*. ACM Symposium on Volume Visualization '94, pp. 51–58, 1994.

[101] R. Westermann and T. Ertl. *Efficiently Using Graphics Hardware in Volume Rendering Applications*. Computer Graphics (ACM SIGGRAPH '98), 32(4), pp. 169–179, 1998.

[102] R. Westermann, L. Kobbelt, and T. Ertl. *Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surfaces*. The Visual Computer, Vol. 15, pp. 100–111, 1999.

[103] L. Westover. *Interactive Volume Rendering*. Proceedings of the '89 Chapel Hill Workshop on Volume Visualization, ACM Press, pp. 9–16, 1989.

[104] J. Wilhelms and A. Van Gelder. *Octrees for Faster Isosurface Generation*. ACM Transactions on Graphics, 11(3), pp. 201–227, 1992.

[105] M.M. Wloka and E. Greenfield. *The Virtual Tricorder: A Uniform Interface for Virtual Reality*. ACM Symposium on User Interface Software and Technology, pp. 39–40, 1995.

[106] W. Wohlfahrter. *Interactive Volume Exploration on the StudyDesk*. Master's Thesis, Vienna University of Technology, Austria, 2000.

[107] W. Wohlfahrter, L.M. Encarnacao, and D. Schmalstieg. *Interactive Volume Exploration on the StudyDesk*. Proceedings of the Fourth International Immersive Projection Technology Workshop, Ames, Iowa, 2000.

[108] U. Wössner, J.P. Schulze, S.P. Walz, and U. Lang. *Evaluation of a Collaborative Volume Rendering Application in a Distributed Virtual Environment*. Proceedings of the Eigth Eurographics Workshop on Virtual Environments (EGVE'02), ACM Press, pp. 113–122, 2002.

[109] wxWindows. *Cross-platform GUI library*. URL: http://www.wxwindows.org.

[110] R. Yagel and A. Kaufman. *Template-Based Volume Viewing*. Computer Graphics Forum, 11(3), pp. 153–167, 1992.

[111] R. Yagel and Z. Shi. *Accelerating Volume Animation by Space-Leaping*. IEEE Visualization '93 Proceedings, pp. 62–69, 1993.

[112] S.Y. Yen, S. Napel, and G.D. Rubin. *Fast Sliding Thin Slab Volume Visualization*. Symposium on Volume Visualization '96 Proceedings, ACM Press, pp. 79–86, 1996.

Summary

In the following sections, the contents of the chapters of this dissertation will be summarized. Where appropriate, the respective section headings will be mentioned in order to allow the reader the fast access to them.

Chapter 1: Introduction

Chapter 1 begins with a motivation for the conducted research (Section 1.1). The motivation originates mainly from the fact that, at the beginning of the research for this dissertation, virtual environments had been available, but they were almost exclusively used for the visualization of polygonal data. The direct visualization of three dimensional scalar fields (volume data) was very compute intensive, so that it was difficult to achieve interactive frame rates. Furthermore, there were only a few and not very sophisticated approaches which allowed changes of the transfer functions or other ways to explore the data directly from within a virtual environment.

Section 1.2 summarizes the most important scientific contributions of this disseration:

- The perspective shear-warp algorithm for volume visualization.
- The application of parallel computers for volume visualization in virtual environments.
- A device independent user interface for virtual environments.
- A user interface for volume visualization in virtual environments.
- The application of pre-integration with the shear-warp algorithm for volume visualization.

Furthermore, Section 1.3 lists the master's theses and semester projects which the author advised on.

Section 1.4 gives an overview of the structure of this dissertation. It should be mentioned that Chapters 3, 4, and 5 can be arranged in an arbitrary order because their content is not interdependent. The chapters present the conducted research in the fields of rendering, interaction, and parallelization. The results from all these fields merge into a volume visualization system in Chapter 6.

Chapter 2: Foundations

Chapter 2 explains the foundations for all the fields in which research was conducted. Section 2.1 describes the most common approaches for volume visualization: ray casting, splatting, shear-warp, texture hardware based approaches, and special purpose hardware. Ray casting is an image space algorithm, it computes the color of every pixel by casting viewing rays through the dataset. The shear-warp algorithm is a variation of ray casting, but via a conversion of the coordinate systems it becomes an object space algorithm, so that memory cache strategies can be applied efficiently. Splatting is an object space approach as well: for every volume element multiple pixels (the splat) are blended on the output image. By using hardware accelerated texture mapping the slices of volume datasets can be rendered very fast. Special purpose volume rendering hardware can implement one of the above methods and execute it very fast. The most recent graphics hardware can be treated as a fast vector processor which can even be used for hardware accelerated ray casting. Finally, Fourier and compression domain volume rendering are presented.

Section 2.2 describes the state of the art of the devices that are used in virtual environments. This section is subdivided into input and output devices. Input devices are: the desktop mouse; the data glove; and mechanical, electromagnetical, optical, and hybrid tracking systems. Output devices are: the monitor, the head mounted display, the workbench, the power wall, the curved screen, and the CAVE.

Section 2.3 introduces hardware and programming models for parallel computing. The hardware is subdivided into computers with distributed memory, shared memory, and hybrid memory, which is a combination of the above. The programming models are based on either threading, which is used with shared memory architectures, or message passing, which is used with distributed memory computers.

The last section of the foundations chapter presents both the application programming interfaces (APIs) that are employed in visualization, and a number of visualization frameworks. The programming interfaces are subdivided into a lower and a higher level, with respect to the hardware. Low level APIs are OpenGL and DirectX, with OpenGL being available for all operating systems, but DirectX only for Windows. The higher level APIs mentioned are OpenGL Performer, OpenGL Optimizer, OpenGL Volumizer, Java 3D, Open Inventor, and the Visualization Toolkit (VTK).

Visualization frameworks can be structured into modular and integrated tools. In modular tools like COVISE, AVS, Amira, and OpenDX, the data flow is represented graphically with a network of modules which read, filter, or render the data. An integrated tool like EnSight does not offer a visual representation of the data flow, but its functionality is accessible in menus.

Chapter 3: Rendering Methods

In Chapter 3 the results from research in the field of rendering methods and algorithms is described. The goal was to accelerate the interactive and direct visualization of volume

data by the development of efficient algorithms. The chapter is structured into three sections, each focusing on a different topic. Section 3.1 reports on specific adaptations of the hardware accelerated texturing approach to the characteristics of virtual environments. It explains how a constant frame rate can be achieved with a multiresolution technique, and it is elaborated on how the texture orientation should be computed.

Section 3.2 describes the enhancement of the shear-warp algorithm to perspective projection. At the beginning of the research for this dissertation it was limited to orthogonal projection. The approach has been previously described by Lacroute [51], but the mathematical derivation was flawed. The correct derivation was facilitated by the introduction of two new coordinate systems.

A further contribution of this dissertation is the proof that projection and warp can be permuted in the case of perspective projection. This is a requirement for the general applicability of the perspective shear-warp algorithm. Furthermore, in this section the complexity of the perspective warp is compared with the orthogonal projection warp. Differences result from the perspective warp being a non-affine transformation, while the orthogonal projection warp is affine.

The second part of Section 3.2 discusses extensions of the perspective shear-warp algorithm, which optimize its usability in virtual environments. The first extension concerns the compositing of the intermediate image. Due to the observation that the size of the intermediate image is not defined by the volume size in the case of the perspective projection algorithm—each slice is scaled differently—the size of the intermediate image can be adapted to the desired frame rate dynamically. This, however, affects in turn the detail level of the output image.

A further optimization is the execution of the warp with hardware accelerated texturing. At this, the intermediate image is loaded as a texture, and the warp matrix is used as the viewing matrix. Thus, the warp can be executed very efficiently in the presence of hardware acceleration, and the rendering time is nearly independent of the size of the output image.

Generally, a significant drawback of the perspective shear-warp algorithm is that prior to the compositing the dataset has to be subdivided into multiple pyramidal sub-volumes. However, Section 3.2.2.4 shows that this is not necessary in the case of a CAVE-like virtual environment, because due to the occurring viewing angles artifacts can be neglected.

Further topics of research in this section are a clipping plane, which can also be used to allow the viewer to step into the dataset, as well as the concurrent display of shear-warp rendered volume datasets and polygonal objects.

In the last part of Section 3.2, results from a number of performance measurements are presented. It focuses on how the rendering time depends on the size of the intermediate image, compares the algorithms for perspective and orthogonal projection, finds out about the differences of the software based and the texturing hardware based warp, and shows which fraction of the total rendering time the warp accounts for.

Section 3.3 explains how the approach of pre-integration can be combined with the shear-warp algorithm. In this case the slices of the dataset cannot be rasterized independently

from one another, but the slabs between them are rendered. Therefore, an efficient implementation of buffer memory is required. The buffer memory is used twice in each rendering pass, once as the front, and once as the back side of a slab. This dissertation compares two different types of buffer memory implementations: slice aligned and intermediate image aligned buffer memory.

A further aspect of this section is the application of the pre-integration table. It showed that bilinear interpolation of the table values does not yield visual improvements, so that it suffices to use nearest neighbor interpolation.

Also, opacity and color correction are discussed in this section. They are required to avoid artifacts if the viewer does not look at the dataset from the direction of a coordinate axis.

The last part of this section contains a comparison of both rendering speed and image quality with different variations of the algorithm. The result is that the shear-warp algorithm with pre-integration improves the image quality noticeably and reaches 34% to 88% of the speed of the traditional shear-warp algorithm.

Chapter 4: Interaction Methods

Chapter 4 discusses the research that was conducted with respect to user interaction with volume data in virtual environments. This chapter is subdivided into two main sections: the development of a device independent user interface, and the investigation of interaction elements suitable for volume visualization.

Section 4.1 explains the development of device independent widgets and a menu system for both the desktop and virtual environments. The basic idea is that the user should be able to develop user interfaces for virtual reality applications with a single API, which can then be used with arbitrary combinations of input and output devices without specific adaptations. Due to the lack of suitable existing user interfaces a new system had to be developed. Both its API and its visual appearance resemble those of the widgets in traditional window managers. However, there are important distinctive features, for instance the arbitrary placement of menus in three-space, and the introduction of rotary knobs as an alternative to sliders.

The new user interface widgets can be used with a variety of graphics APIs, because all graphics routines are detached from the actual graphics API by an intermediate layer. As a proof of concept, example implementations for OpenGL Performer and OpenGL Optimizer were built.

The other main section of this chapter discusses research on interaction elements for volume visualization (Section 4.2). A two-step approach was used for the development of these elements, both steps comprising user studies.

One of the most important interaction elements in volume rendering is the transfer function editor. It was developed with a focus on issues with imprecise input devices in virtual environments, as well as the low resolution relative to the screen size. It distinguishes itself in that, in contrast to earlier editors, both the opacity and the color transfer functions

are not drawn as lines, but they are composed of a number of basic elements, which are well manageable in virtual environments.

In the case of the opacity the basic elements are the trapezoid, the ramp, and the blank. These elements are adjustable in their width and some in other parameters. The opacity transfer function is composed by the superposition of combinations of these elements. The advantage of this method is, as opposed to freehand drawing techniques, that by the limitation of the degrees of freedom more exact work is possible. Additionally, the exploration of a dataset is substantially facilitated for instance by moving a triangular peak through the data value range, in contrast to traditional techniques, where the peak has to be constructed from multiple vertices or a freehand line.

The basic elements of the color transfer function are control points in the value range of the scalar data. For each control point an arbitrary color can be selected from a colored disk, with an option to select the brightness. The color values between the control points are interpolated in color space.

The new user interface for volume visualization is not limited to the transfer function editor, but it offers further functionality in menus. The user can load datasets from disk, or save the current dataset and the state of its transfer function.

An important functionality of the menu is the selection of the frame rate. It affects indirectly the image quality of the volume dataset. In contrast to a direct selection of the rendering quality, this option was chosen because in virtual environments it is crucial to maintain a constant frame rate in order not to disrupt the effect of immersion.

Another menu option is a flashlight-like probe mode. In this mode, only a cubic subset of the dataset is displayed, so that one can see the inside of the volume object without changing the transfer functions. An alternative to this option is the clipping plane, which cuts off a part of the dataset. Both methods are based on direct interaction in data space.

Two user studies were carried out, in which twelve and ten participants were invited, respectively. The majority of them came from institutes of the University of Stuttgart and had some prior knowledge about volume rendering. In each study the participants should work with volume datasets and our new user interface in two scenarios. Before and after working in the virtual environment they had to fill out questionnaires, and at the end an interview was conducted. The time in the virtual environment was recorded on video tape.

Most of the suggestions for improvement from the first study were put into practice. In the second study the participants were asked explicitly about the usability of the new version. It turned out that the developed volume visualization software is well suited for use with volume data, however, for professional applications in the medical or engineering fields specific adaptations would be desirable.

Chapter 5: Parallelization and Distribution Methods

In virtual environments with multiple stereo display screens the serial shear-warp algorithm is inferior to hardware accelerated texturing approaches with respect to rendering

speed. However, in these cases even the hardware accelerated texturing techniques reach their limits because they are constrained by the hardware's pixel fill rate. Therefore, this chapter investigates the potential of a parallelization of the perspective shear-warp algorithm.

After an introduction in Section 5.1 and a look at previous work in Section 5.2, Section 5.3 describes the parallelized algorithm.

The efficient application of the parallelized shear-warp algorithm requires a parallel computer. In the general case parallel computers do not contain visualization hardware, but they can be linked to a visualization system via a network connection. The approach that is described in this dissertation assumes that both of these systems are available.

The idea of the parallelized perspective shear-warp algorithm which is described here is based on the compositing being done on the parallel computer. It transfers the resulting intermediate image to the visualization system, which displays the image with a texturing hardware accelerated warp. Because typically the intermediate image is smaller than the output image, less data has to be transferred using this approach.

The parallelization of the algorithm is similar to Lacroute's parallelization of the orthogonal projection algorithm [52]: the intermediate image is subdivided into sections of scanlines that are distributed to the nodes of the parallel computer, which do the compositing for their sections and return the result to the master node. In order to use this approach on parallel computers with distributed memory, the volume dataset has to be replicated on all nodes. An optimization of the memory requirements is difficult because the further back the volume slices are located, with respect to the viewer, the more intermediate image lines their voxels contribute to.

Because the intermediate image transfer is a bottleneck when the network is slow, the intermediate image is run-length encoded before its transfer. However, not the entire image is encoded, but only the rectangular window which contains the actual volume. The size of this area is computed from the shear-warp transformation matrix. With this method, depending on the dataset, the viewing angle and the network speed, the transfer speed can be improved by up to 90%.

The visualization system decodes the intermediate image and transfers it to texture memory. In order to correctly blend other objects with the volume dataset, not only color, but also opacity has to be transferred for each pixel. Along with the warp's feature of projecting the texture to the correct Z position, which is described in Chapter 3, the volume dataset can be displayed concurrently with the user interface elements.

Besides rendering, the visualization system has to transfer user commands to the parallel computer. Therefore, a protocol was developed, which allows the transfer of volume datasets, matrices, and changes of the viewing parameters, for instance the interpolation mode. Furthermore, the image computation on the parallel computer and the user interaction on the visualization system are nested, so that the visualization system can display the other objects of the scenegraph and process user input instead of waiting for the parallel computer.

The resulting system was tested on three different types of parallel computers: an SGI Onyx2, a SUN Fire 6800 node, and a cluster of Linux PCs. The Onyx2 achieves 3.4 frames per second with 14 processors, which are reached by the SUN Fire with 6 processors already. The PC cluster renders 7.4 frames per second with 12 processors. Further measurements analyze the load of the processors, the speed of the intermediate image transfer, and the rendering speed in comparison to the texturing hardware accelerated approach.

Chapter 6: Volume Visualization System

The developments of Chapters 3 to 5 were integrated with the visualization framework COVISE, in order to evaluate their practical usability. Previously, COVISE did not offer volume visualization. In Chapter 6 implementational issues and their solutions are described.

The chapter is subdivided into the following sections: the first section shortly describes the software environment which was used for the development of the new algorithms, which is the basis for the integration. Section 6.2 describes why COVISE was selected as the visualization framework in which the new algorithms were to be integrated.

Section 6.3 investigates which previously existing COVISE data formats could be used to represent volume data. In this context, 8 and 16-bit scalar data, as well as 24 and 32-bit data formats for pre-classified or photographic volume data are considered.

Section 6.4 analyzes how the previously existing COVISE file formats could be used to store volume data. Furthermore, new data formats are presented, which were specifically developed for the use with virtual environments, for instance they can store the position of the dataset in space.

The visualization of volume data with different user interfaces for the desktop and virtual environments is described in Section 6.5. Specific facts that result from the features of the input and output devices are elaborated on. For instance, at the desktop the visualization of the dataset can be refined when there is no user input, which cannot be done in a virtual environment because, due to the head tracking, the image never stands still.

Chapter 7: Conclusions

Chapter 7 summarizes the results of this dissertation, discusses their scientific relevance, and gives ideas for future work. The goal of the dissertation was to improve interactive direct volume visualization. This task was approached with three topics: faster visualization algorithms, practical and intuitive interaction techniques, and efficient parallelization methods.

In the field of rendering algorithms the perspective shear-warp algorithm was adapted to virtual environments, so that now a very fast CPU-based visualization approach can be

used. Furthermore, the approach of pre-integration was combined with the shear-warp algorithm, which improved image quality significantly.

The research in the field of interaction techniques with volume data resulted in a usable user interface for the work with volume data in virtual environments. User studies verified and optimized the results.

The parallelization of the perspective shear-warp algorithm allows the efficient application of the algorithm in virtual environments that have access to a parallel computer.

Finally, it is demonstrated how the developed algorithms and techniques can be combined to a visualization system, in order to allow the interactive direct visualization of volume data in virtual environments.

Section 7.1 shows that in this dissertation all the components of an interactive volume visualization system are discussed. The components are put in relation to each other by explaining a typical visualization procedure. Furthermore, the prerequisites for the usage of the developed visualization algorithm are mentioned.

Future work, which is proposed in Section 7.3, could address further performance improvements of the perspective shear-warp algorithm by its combination with level of detail or data reduction approaches. Also, an improved load balancing could increase the performance. Furthermore, the algorithm could be adapted to hybrid memory architectures, which will be used more and more in the future, by a combination of OpenMP and MPI.

Also, the interaction with volume data could be improved by more research for methods of direct interaction or additional functionality for the transfer function editor. Some participants of the user studies requested special functionality, which is required only in their professional fields.

German Summary: Zusammenfassung

In den folgenden Abschnitten werden die Inhalte der einzelnen Kapitel dieser Dissertation in deutscher Sprache zusammengefasst dargestellt. Es werden auch die jeweils angesprochenen Abschnitte der Arbeit genannt, um dem Leser bei Bedarf den schnellen Zugriff darauf zu ermöglichen.

Kapitel 1: Einführung

Kapitel 1 beginnt mit einer Erläuterung der hinter den durchgeführten Forschungsarbeiten stehenden Motivation (Abschnitt 1.1). Diese ergab sich vor allem daraus, dass zwar seit einigen Jahren virtuelle Umgebungen zur Visualisierung dreidimensionaler Daten zur Verfügung stehen, diese jedoch zu Beginn der Forschungsarbeiten dieser Dissertation hauptsächlich zur Visualisierung polygonaler Daten eingesetzt wurden. Die direkte Darstellung dreidimensionaler Skalarfelder (Volumendaten) war sehr rechenzeitintensiv, so dass es schwierig war, damit interaktive Bildraten zu erreichen. Darüberhinaus gab es nur wenige und nicht besonders effektive Ansätze, die Änderungen der Transferfunktionen oder andere Möglichkeiten zur Erkundung der Daten direkt von der virtuellen Umgebung aus zu erlauben.

In Abschnitt 1.2 werden die wichtigsten wissenschaftlichen Beiträge dieser Arbeit genannt. Diese sind:

- Der perspektivische Shear-Warp-Algorithmus zur Volumenvisualisierung.
- Der Einsatz von Parallelrechnern zur Volumenvisualisierung in virtuellen Umgebungen.
- Eine geräteunabhängige Benutzerschnittstelle für virtuelle Umgebungen.
- Eine Benutzerschnittstelle für Volumenvisualisierung in virtuellen Umgebungen.
- Der Einsatz von Vorintegration im Shear-Warp-Algorithmus zur Volumenvisualisierung.

Weiterhin werden in Abschnitt 1.3 die im Rahmen der Dissertation betreuten Diplom- und Masterarbeiten, sowie Semesterarbeiten aufgelistet.

Abschnitt 1.4 erläutert schließlich den Aufbau der Dissertation. Hierzu ist besonders zu erwähnen, dass die Kapitel 3, 4 und 5 bezüglich inhaltlicher Zusammenhänge in beliebiger Reihenfolge stehen könnten. Diese Kapitel präsentieren die durchgeführten Forschungsarbeiten in den Gebieten Darstellung, Interaktion und Parallelisierung. Die Ergebnisse dieser drei Kapitel fließen in Kapitel 6 zusammen zu einem Volumenvisualisierungssystem.

Kapitel 2: Grundlagen

In Kapitel 2 werden die Grundlagen für alle Gebiete gelegt, in denen im Folgenden von Forschungsarbeiten berichtet wird. Abschnitt 2.1 beschreibt die bekanntesten Ansätze zur Volumenvisualisierung: Strahlverfolgung, Splatting, Shear-Warp, texturhardwarebasiert und den Einsatz spezieller Hardware. Strahlverfolgung berechnet im Bildraum für jedes Pixel den entsprechenden Farbwert, indem Sehstrahlen durch das Volumen geschickt werden. Der Shear-Warp-Algorithmus ist eine Variante der Strahlverfolgung, nur dass hier durch Umrechnung der Koordinatensysteme im Objektraum gerechnet werden kann, wodurch Cache-Speicher effizient genutzt werden kann. Splatting ist ein reines Objektraumverfahren, da für jedes Volumenelement mehrere Pixel auf den Bildschirm eingefärbt werden. Mit hardwarebeschleunigter Texturdarstellung können die Schichten von Volumendaten sehr schnell auf den Bildschirm gebracht werden, während spezielle Volumenvisualisierungs-Hardware ein beliebiges dieser Verfahren implementieren kann. Die neusten Grafikkarten können als schnelle Vektorprozessoren aufgefasst werden, mit denen sogar hardwarebeschleunigte Strahlverfolgung möglich ist. Am Schluss werden Fourier und kompressionsbasierte Volumenvisualisierungsverfahren präsentiert.

Abschnitt 2.2 beschreibt den aktuellen Stand der in virtuellen Umgebungen eingesetzten Geräte. Der Abschnitt ist unterteilt in Eingabe- und Ausgabegeräte. Als Eingabegeräte werden genannt: die Maus, der Datenhandschuh, sowie mechanische, elektromagnetische, optische und hybride Tracking-Systeme. Als Ausgabegeräte werden beschrieben: der Monitor, das Head Mounted Display, die Workbench, die Power Wall, die gekrümmte Projektionswand und die CAVE.

Abschnitt 2.3 führt in Hardware und Programmiermodelle ein, die beim Parallelrechnen eingesetzt werden. Die Hardware wird unterschieden in Computer mit verteiltem Speicher, gemeinsamem Speicher, sowie mit hybridem Speicher, der eine Kombination der beiden erstgenannten Methoden darstellt. Die Programmiermodelle werden unterschieden in Threading, was auf Architekturen mit gemeinsamem Speicher eingesetzt wird, sowie Message Passing, was auf Computern mit verteiltem Speicher benötigt wird.

Der letzte Abschnitt, in dem Grundlagen erläutert werden, stellt zum einen die im Bereich der Visualisierung eingesetzten Programmierschnittstellen (APIs) vor, zum anderen eine Reihe von Visualisierungsprogrammen. Die Programmierschnittstellen werden unterschieden in eine niedrige und eine höhere Ebene, relativ zur Hardware. Auf niedriger

Ebene befinden sich die APIs OpenGL und DirectX, wobei OpenGL auf allen Systemen, DirectX jedoch nur auf Windows-Systemen verfügbar ist. Auf höherer Ebene befinden sich die APIs OpenGL Performer, OpenGL Optimizer, OpenGL Volumizer, Java 3D, Open Inventor, sowie das Visualization ToolKit (VTK).

Visualisierungsprogramme werden unterschieden in modulare und integrierte Werkzeuge. In modularen Programmen wie COVISE, AVS, Amira und OpenDX wird der Datenfluss graphisch repräsentiert, wobei unterschiedliche Module eingesetzt werden, um Daten zu lesen, zu filtern oder darzustellen. Ein integriertes Werkzeug wie Ensight bietet nicht die Darstellung des Datenflusses, dafür wird jedoch die jeweilige Funktionalität in Menüs zur Verfügung steht.

Kapitel 3: Darstellungsmethoden

In Kapitel 3 werden die erzielten Forschungsergebnisse im Bereich der Darstellungsmethoden und -algorithmen erläutert. Ziel war es dabei, die interaktive und direkte Visualisierung von Volumendaten mit Hilfe effizienter Algorithmen zu beschleunigen. Das Kapitel ist gegliedert in drei Abschnitte, die jeweils einen Arbeitsschwerpunkt zum Thema haben. Abschnitt 3.1 berichtet über spezielle Anpassungen der hardwarebeschleunigten Darstellung von Volumendaten mit Texturen an die Gegebenheiten virtueller Umgebungen. Hier wird erläutert, wie man mit Multiresolution-Techniken eine konstante Bildrate erzielen kann, und es wird darauf eingegangen, wie die Orientierung der Texturen gewählt werden sollte.

Abschnitt 3.2 beschreibt die Erweiterung des Shear-Warp-Algorithmus, der zu Beginn der Arbeiten zu dieser Dissertation nur für Parallelprojektion zur Verfügung stand, auf perspektivische Projektion. Diese wurde zwar zuvor bereits von Lacroute [51] beschrieben, jedoch fehlerhaft hergeleitet. Die korrekte mathematische Herleitung wurde insbesondere durch die Einführung zweier neuer Koordinatensysteme erleichtert.

Ein weiterer Beitrag dieser Dissertation ist der Beweis der Vertauschbarkeit von Projektion und Warp für den Fall der perspektivischen Projektion. Dies ist eine Voraussetzung für die allgemeine Anwendbarkeit des perspektivischen Shear-Warp-Algorithmus. Weiterhin wird in diesem Abschnitt die Komplexität des perspektivischen Warp mit dem Warp bei Parallelprojektion verglichen. Unterschiede ergeben sich daraus, dass der perspektivische Warp eine nicht-affine Abbildung ist, im Gegensatz zum Warp bei Parallelprojektion.

Der zweite Teil von Abschnitt 3.2 behandelt Erweiterungen des perspektivischen Shear-Warp-Algorithmus, die dessen Einsatz in virtuellen Umgebungen optimieren. Die erste Erweiterung betrifft das Erstellen des Zwischenbildes (Compositing). Aufgrund der Tatsache, dass beim perspektivischen Algorithmus die Größe des Zwischenbildes nicht durch die Größe des Volumens vorgegeben ist, weil jede Schicht unterschiedlich skaliert wird, folgt, dass man die Größe des Zwischenbildes dynamisch an die gewünschte Bildrate anpassen kann. Dies beeinflusst jedoch direkt die Detaillierung des Ausgabebildes.

Eine weitere Optimierung ist die Durchführung des Warp mit Hilfe von hardwarebeschleunigter Texturdarstellung. Hierbei wird das Zwischenbild in eine Textur geladen und

die Warp-Matrix als Abbildungsmatrix benutzt. Somit kann der Warp-Schritt bei vorhandener Grafikbeschleunigung sehr effizient ausgeführt werden und die Bildberechnungsdauer ist nahezu unabhängig von der Größe des Ausgabebildes.

Im allgemeinen Fall ist ein großer Nachteil des perspektivischen Shear-Warp-Algorithmus, dass vor dem Compositing der Datensatz in mehrere pyramidenförmige Teilvolumina aufgeteilt werden muss. Abschnitt 3.2.2.4 zeigt, dass dies zumindest im Fall einer CAVE-artigen virtuellen Umgebung nicht notwendig ist, da aufgrund der hier vorkommenden Sichtwinkel kaum Bildfehler auftreten können.

Weitere angesprochene Forschungsthemen sind die Berücksichtigung einer Schnittebene, die auch dazu eingesetzt werden kann, dass der Betrachter in den Datensatz hineingehen kann, sowie eine Betrachtung der simultanen Darstellung von mit dem Shear-Warp-Algorithmus erzeugten Datensätzen und polygonalen Objekten.

Im letzten Teil des Abschnitts 3.2 werden Ergebnisse mehrerer Leistungsmessungen präsentiert. Es wird insbesondere darauf eingegangen, wie die Bildberechnungszeit von der Zwischenbildgröße abhängt, wie sich perspektivische und Parallelprojektion zueinander verhalten, wie sich der softwarebasierte und der texturhardwarebasierte Warp unterscheiden, und welchen Anteil der Warp an der gesamten Bildberechnung hat.

Abschnitt 3.3 erklärt, wie die Technik der Vorintegration mit dem Shear-Warp-Algorithmus kombiniert werden kann. Da hierzu nicht mehr die einzelnen Schichten des Datensatzes unabhängig voneinander gerastert werden, sondern die Bereiche zwischen den Schichten (Scheiben), wird für eine effiziente Implementierung ein Pufferspeicher benötigt. Dieser wird jeweils zwei Mal hintereinander benutzt, einmal als Vorderseite und einmal als Rückseite einer Scheibe. Die Dissertation vergleicht zwei unterschiedliche Methoden zur Implementierung des Pufferspeichers: schichtbezogene und zwischenbildbezogene Pufferspeicher.

Ein weiterer Aspekt dieses Abschnitts ist die Einsatzweise der Vorintegrationstabelle. Hierbei zeigte sich, dass eine bilineare Interpolation der Tabellenwerte keine sichtbaren Vorteile brachte, so dass es genügt, den nächsten Nachbarwert zu benutzen.

Auch die Korrektur der Opazitäts- und Farbwerte wird in diesem Abschnitt betrachtet, die notwendig ist, um Bildfehler zu vermeiden, wenn der Datensatz in einem schrägen Winkel zum Betrachter steht.

Der letzte Teil des Abschnitts enthält eine Betrachtung sowohl der Darstellungsgeschwindigkeit, sowie der Bildqualität mit unterschiedlichen Varianten des Algorithmus. Ein Ergebnis ist, dass der Shear-Warp-Algorithmus mit Vorintegration die Bildqualität erhöht und je nach Datensatz und Darstellungsparametern zwischen 34% und 88% der Geschwindigkeit des herkömmlichen Shear-Warp erreicht.

Kapitel 4: Interaktionsmethoden

Kapitel 4 betrachtet die Forschungsarbeiten, die im Bezug auf die Interaktion des Benutzers mit Volumendaten in virtuellen Umgebungen stattfanden. Dieses Kapitel ist aufgeteilt

in zwei große Abschnitte: die Erstellung einer geräteunabhängigen Benutzerschnittstelle, sowie die Erforschung von Interaktionselementen zur Volumenvisualisierung.

Abschnitt 4.1 erläutert die Entwicklung von Benutzungselementen und eines Menüsystems für den geräteunabhängigen Einsatz am Arbeitsplatz und in virtuellen Umgebungen. Die Grundidee ist, dass Anwendungen mit einer einheitlichen Programmierschnittstelle erstellt werden und dann mit beliebigen Kombinationen von Ein- und Ausgabegeräten aus dem Bereich der virtuellen Realität eingesetzt werden können. Mangels geeigneter vorhandener Benutzerschnittstellen für virtuelle Umgebungen musste ein neues System erstellt werden. Sowohl dessen API, als auch das visuelle Erscheinungsbild der Benutzungselemente wurden hierbei an herkömmliche Fenster-Manager angelehnt. Es gibt jedoch wichtige Unterscheidungsmerkmale, wie die freie Positionierbarkeit von Menüs im Raum, sowie die Einführung von Drehknöpfen als Alternative zu Schiebereglern.

Die erstellten Benutzungselemente können darüberhinaus mit unterschiedlichen Grafik-APIs eingesetzt werden, da die Grafikroutinen durch eine allgemeine Zwischenschicht von der eigentlichen Implementierung abgesetzt sind. Zur Demonstration dieser Funktionalität wurden Implementierungen für OpenGL Performer und OpenGL Optimizer erstellt.

Der zweite große Abschnitt dieses Kapitels behandelt die Erforschung von Interaktionselementen für die Volumenvisualisierung (Abschnitt 4.2). Die Entwicklung dieser Elemente wurde in zwei Stufen durchgeführt, die jeweils eine Benutzerstudie enthalten.

Eines der wichtigsten Interaktionselemente bei der Volumenvisualisierung ist der Transferfunktionseditor. Dieser wurde besonders im Hinblick auf die Problematik der ungenauen Eingabegeräte in virtuellen Umgebungen, sowie der dort häufig anzutreffenden für die Bildgröße geringen Auflösung entwickelt. Er zeichnet sich dadurch aus, dass im Gegensatz zu bisherigen Editoren sowohl die Opazitäts- als auch die Farb-Transferfunktion nicht als eine Linie gezeichnet, sondern aus unterschiedlichen Grundelementen zusammengesetzt wird, die in virtuellen Umgebungen gut handhabbar sind.

Diese Grundelemente sind im Fall der Opazität das Trapez, die Rampe, sowie der Ausblender. Diese Elemente sind in ihrer Breite und ggf. anderen Parametern verstellbar. Die Transferfunktion entsteht durch die Überlagerung beliebiger Kombinationen dieser Elemente. Der Vorteil dieser Methode ist gegenüber Freihand-Techniken, dass durch die Einschränkung der Freiheitsgrade ein exakteres Arbeiten möglich ist, und dass das Erkunden eines Datensatzes beispielsweise durch das Verschieben einer Dreiecksfunktion im Skalarwertebereich wesentlich gegenüber herkömmlichen Techniken erleichtert wird, wobei die Dreiecksfunktion aus mehreren getrennt anzupassenden Eckpunkten oder einer Freihandlinie besteht.

Die Grundelemente der Farb-Transferfunktion sind Kontrollpunkte im Wertebereich der Skalardaten. Für diese Kontrollpunkte kann anhand eines Farbkreises und einer Möglichkeit zur Einstellung der Helligkeit eine beliebige Farbe ausgewählt werden. Die zwischen den Kontrollpunkten liegenden Werte werden im Farbraum interpoliert.

Die neue Benutzerschnittstelle zur Volumenvisualisierung beschränkt sich jedoch nicht auf den Transferfunktionseditor, sondern bietet weitere Funktionen über Menüs an. So

können Datensätze vom Massenspeicher geladen werden, oder es können der aktuelle Datensatz und seine Transferfunktion gespeichert werden.

Eine wichtige Funktionalität des Menüs ist die Wahl der Bildrate. Durch diese wird indirekt festgelegt, in welcher Qualitätsstufe der Volumendatensatz dargestellt werden kann. Im Gegensatz zu einer direkten Wahl der Darstellungsqualität wurde diese Variante gewählt, weil es in virtuellen Umgebungen wichtig ist, die Bildrate konstant zu halten, um den Immersionseffekt nicht zu gefährden.

Eine weitere Menüfunktion ist der taschenlampenähnliche Erkundungsmodus (Probe Mode). Darin wird ein würfelförmiger Ausschnitt des Datensatzes dargestellt, wodurch man auch ohne spezielle Transferfunktion in das Innere des Datensatzes hineinsehen kann. Eine Alternative dazu ist die Schnittfläche, die einen Teil des Datensatzes abschneidet. Beide Funktionen basieren auf direkter Interaktion im Datenbereich.

Zu den beiden Benutzerstudien wurden zwölf bzw. zehn Probanden eingeladen, von denen die Mehrzahl von Instituten der Universität Stuttgart stammte und Grundwissen über Volumenvisualisierung hatte. Die Probanden sollten in jeweils zwei Szenarien mit Volumendatensätzen und mit der erstellten Benutzerschnittstelle arbeiten. Vor und nach der Arbeit in der virtuellen Umgebung mussten die Benutzer Fragebögen ausfüllen, und am Ende wurde zusätzlich ein Interview durchgeführt. Zudem wurden Audio- und Videoaufnahmen der Arbeit in der virtuellen Umgebung angefertigt.

Die meisten der aus der ersten Studie hervorgegangenen Verbesserungsvorschläge wurden umgesetzt, und die Probanden wurden in der zweiten Studie gezielt danach gefragt, wie gut die neue Version benutzbar ist. Es stellte sich am Ende heraus, dass die erstellte Software im allgemeinen Fall gut zur Arbeit mit Volumendaten einsetzbar ist, jedoch für spezielle Anwendungsfelder wie Medizin oder Ingenieurwesen individuelle Anpassungen wünschenswert sind.

Kapitel 5: Parallelisierungs- und Verteilungsmethoden

Im Umfeld einer virtuellen Umgebung mit mehreren Projektionsflächen und Stereodarstellung erweist sich der serielle Shear-Warp-Algorithmus hardwarebasierten Techniken wie der Volumenvisualisierung mit Texturen als in der Rechengeschwindigkeit unterlegen. Bei den hier auftretenden großen Ausgabefenstern gerät allerdings auch die hardwareunterstützte Technik an ihre Grenzen, da sie durch die Pixel-Füllrate limitiert ist. In diesem Kapitel wird daher das Potential der Parallelisierung des perspektivischen Shear-Warp-Algorithmus erforscht.

Nach einer Einführung in Abschnitt 5.1 und einer Betrachtung früherer Arbeiten in Abschnitt 5.2 beschreibt Abschnitt 5.3 die entwickelte Visualisierungsmethode.

Der effiziente Einsatz eines parallelisierten Shear-Warp-Algorithmus setzt einen Parallelrechner voraus. Im allgemeinen Fall besitzen leistungsfähige Parallelrechner jedoch keine Visualisierungshardware, sie können jedoch über eine Netzwerkverbindung an ein Visualisierungssystem angebunden werden. Der in dieser Dissertation beschriebene Ansatz zur parallelisierten Volumenvisualisierung setzt das Vorhandensein beider Systeme voraus.

Die hier beschriebene Idee des parallelisierten perspektivischen Shear-Warp-Algorithmus basiert darauf, dass das Compositing auf dem Parallelrechner stattfindet, dieser das Zwischenbild an das Visualisierungssystem überträgt, und letzteres mit einem in Hardware ausgeführten Warp den Volumendatensatz darstellt. Da typischerweise das Zwischenbild kleiner ist als das Ausgabebild ergibt sich daraus der Vorteil einer geringeren Datenübertragungsmenge über das Netzwerk.

Die Parallelisierung des Algorithmus wurde angelehnt an die von Lacroute durchgeführte Parallelisierung des parallel projizierenden Algorithmus [52]: das Zwischenbild wird zeilenweise in Abschnitte unterteilt, die von den Knoten des Parallelrechners bearbeitet werden. Um diesen Ansatz auf Parallelrechnerarchitekturen mit verteiltem Speicher durchführen zu können, muss der Volumendatensatz auf allen Knoten zur Verfügung stehen. Eine Optimierung des Speicherbedarfs ist hierbei problematisch, da weiter hinten liegende Volumenschichten auf weniger Bildzeilen abgebildet werden müssen als Voxelzeilen vorhanden sind.

Da die Bildübertragung bei langsamen Netzwerken den Flaschenhals darstellt, wird das Zwischenbild vor der Übertragung mit einem Lauflängencodierungsverfahren komprimiert. Hierbei wird nicht das gesamte Zwischenbild codiert, sondern lediglich der Bereich, in dem sich das Volumen befindet. Dieser wird anhand der Shear-Warp-Transformationsmatrix ermittelt. Dadurch konnte je nach Datensatz, Betrachtungswinkel und Netzwerkgeschwindikgeit eine Beschleunigung der Datenübertragung um bis zu 90% erzielt werden.

Auf der Seite des Visualisierungssystems wird das Zwischenbild entschlüsselt und in eine Textur geladen. Damit andere dargestellte Objekte korrekt vom Volumendatensatz verdeckt werden, müssen nicht nur die Farb-, sondern auch die Opazitätswerte des Zwischenbilds übertragen werden. Zusammen mit der in Kapitel 3 beschriebenen Eigenschaft des Warp, die Textur an die Z-Position des eigentlichen Datensatzes zu projizieren, kann damit der Volumendatensatz zusammen mit Elementen der Benutzerschnittstelle dargestellt werden.

Das Visualisierungssystem hat neben der Grafikdarstellung auch die Aufgabe, die Benutzerkommandos an den Parallelrechner weiterzuleiten. Hierzu wurde ein Protokoll erstellt, mit dem Volumendatensätze, Matrizen, sowie Änderungen der Darstellungsparameter, wie z.B. der Interpolationsmodus, übertragen werden können. Die Vorgänge der Bildberechnung und der Benutzerinteraktion wurden verschachtelt, so dass das Visualisierungssystem, anstatt auf den Parallelrechner zu warten, die übrige Szene darstellt und Benutzereingaben verwaltet.

Das resultierende System wurde auf drei unterschiedlichen Parallelrechnern getestet: einer SGI Onyx2, einem SUN Fire 6800 Knoten, sowie einem Cluster aus Linux-PCs. Hierbei wurde auf der Onyx2 mit 14 Prozessoren eine Bildrate von 3,4 Bildern pro Sekunde erzielt, die auf der SUN Fire bereits mit 6 Prozessoren erreicht wurde. Der PC-Cluster erreichte mit 12 Prozessoren 7,4 Bilder pro Sekunde. Weitere Messreihen untersuchen die Auslastung der Prozessoren, die Übertragungsgeschwindigkeit der Zwischenbilder, sowie die Darstellungsgeschwindigkeit im Vergleich mit dem texturhardwarebeschleunigten Ansatz.

Kapitel 6: Volumenvisualisierungssystem

Um die praktische Anwendbarkeit der in den Kapiteln 3 bis 5 beschriebenen Entwicklungen beurteilen zu können, wurden sie in das bestehende Visualisierungssystem COVISE integriert, welches zuvor keine Möglichkeit zur direkten Volumenvisualisierung bot. In Kapitel 6 werden die bei diesem Vorgang aufgetretenen Schwierigkeiten und deren Lösungsmöglichkeiten beschrieben.

Das Kapitel ist unterteilt in die folgenden Abschnitte: der erste Abschnitt beschreibt die zur Entwicklung der Algorithmen eingesetzte Softwareumgebung, die die Basis des Integrationsvorgangs darstellt. Abschnitt 6.2 begründet die Auswahl von COVISE als das Visualisierungssystem, in das die entwickelten Algorithmen integriert werden sollten.

Abschnitt 6.3 untersucht, welche ursprünglichen Datenformate von COVISE zum Speichern von Volumendaten eingesetzt werden können. Hierbei werden 8- und 16-bit Skalardatenformate, sowie 24- und 32-bit Datenformate für vorklassifizierte oder photographische Volumendaten betrachtet.

In Abschnitt 6.4 wird untersucht, wie die bestehenden Dateiformate von COVISE zur Speicherung von Volumendaten eingesetzt werden können. Darüberhinaus werden neue Datenformate vorgestellt, die sich besonders zur Speicherung von Volumendaten eignen, die in virtuellen Umgebungen betrachtet werden sollen. Besonderheiten sind hier beispielsweise die Speicherung der Position des Datensatzes im Raum.

Die Darstellung der Volumendaten mit unterschiedlichen Benutzerschnittstellen für den Arbeitsplatz und für virtuelle Umgebungen wird in Abschnitt 6.5 beschrieben. Hierbei wird auf die jeweiligen besonderen Gegebenheiten eingegangen, die sich durch die Eigenschaften der Ein- und Ausgabegeräte ergeben. So ist beispielsweise am Arbeitsplatz eine verfeinerte Darstellung des Datensatzes möglich, sobald keine Benutzereingaben mehr erfolgen, während dies in virtuellen Umgebungen nicht möglich ist, da das Bild durch die permanent neu abgefragte Benutzerposition nicht still steht.

Kapitel 7: Ergebnisse

Kapitel 7 fasst die Resultate dieser Dissertation zusammen, betrachtet ihre wissenschaftliche Bedeutung und gibt einen Ausblick auf weitere Forschungsarbeiten auf diesem Gebiet. Die Dissertation hatte zum Ziel, die interaktive direkte Volumenvisualisierung zu verbessern. Diese Aufgabenstellung wurde auf drei Ebenen bearbeitet: mit schnelleren Darstellungsalgorithmen, mit geeigneten und möglichst intuitiven Interaktionstechniken und mit effizienten Parallelisierungsmethoden.

Auf dem Gebiet der Darstellungsalgorithmen wurde der perspektivische Shear-Warp-Algorithmus an virtuelle Umgebungen angepasst, wodurch darin ein sehr schneller CPU-basierter Visualisierungsansatz zur Verfügung steht. Weiterhin wurde die Technik der Vorintegration mit dem Shear-Warp-Algorithmus kombiniert, wodurch sich die Bildqualität signifikant verbessern liess.

Die Forschungsarbeiten im Bereich der Interaktionstechniken mit Volumendaten führten dazu, dass nun eine gut zu bedienende Benutzerschnittstelle zur Arbeit mit Volumendaten in virtuellen Umgebungen zur Verfügung steht. Durch die durchgeführten Benutzerstudien konnten diese Ergebnisse verifiziert und optimiert werden.

Die Parallelisierung des perspektivischen Shear-Warp-Algorithmus erlaubt den effizienten Einsatz des Algorithmus in virtuellen Umgebungen, wenn diese Zugriff auf einen Parallelrechner haben.

Schließlich wurde gezeigt, wie die untersuchten Algorithmen und Techniken zu einem Visualisierungssystem zusammengeführt werden können, um damit die interaktive direkte Volumenvisualisierung in virtuellen Umgebungen zu ermöglichen.

Abschnitt 7.1 verdeutlicht, dass diese Dissertation alle Komponenten eines interaktiven Volumenvisualisierungssystems behandelt. Die einzelnen Komponenten werden anhand eines typischen Visualisierungsablaufs in Bezug zueinander gesetzt, und es wird angegeben, wie der geeignetste Visualisierungsalgorithmus ausgewählt werden kann.

In zukünftigen Arbeiten, die in Abschnitt 7.3 vorgeschlagen werden, könnte der perspektivische Shear-Warp-Algorithmus durch die Kombination mit unterschiedlichen Detailstufen oder Datenreduktionsmethoden weiter beschleunigt werden. Eine verbesserte Lastverteilung könnte den parallelisierten Algorithmus noch schneller machen. Weiterhin könnte der Algorithmus durch Kombination von OpenMP und MPI an die in Zukunft häufiger eingesetzten hybriden Parallelrechner angepasst werden.

Auch die Interaktion mit Volumendaten könnte verbessert werden, indem weitere Techniken zur direkten Interaktion untersucht werden, oder der Transferfunktionseditor weitere Funktionalität erhält. Auch an einer Spezialisierung auf bestimmte fachliche Einsatzgebiete könnte durchgeführt werden, wie verschiedentlich in den Benutzerstudien gewünscht.

Curriculum Vitae

EDUCATION

08/2003	Ph.D., Computer Science, University of Stuttgart
03/1999	Diplom-Informatiker (M.Sc. equivalent), Computer Science, University of Stuttgart
01/1998	Master of Science, Computer Science, University of Massachusetts at Dartmouth, USA
11/1995	Vordiplom (undergraduate degree), Computer Science, University of Stuttgart
05/1993	Abitur (high school diploma), Geschwister-Scholl-Gymnasium Stuttgart

WORK EXPERIENCE

04/1999 – Present	Research assistant in the visualization group at the High Performance Computing Center Stuttgart (HLRS)
12/1998 – 02/1999	Intern at IBM Research and Development, Böblingen
12/1994 – 12/1998	Self-employed software developer for office and entertainment applications
02/1994 – 04/1994	Intern at Daimler-Benz, Stuttgart
09/1993 – 03/1995	Research assistant at the Fraunhofer Institute for Production Technology and Automation (IPA), Stuttgart
06/1993 – 07/1993	Intern at the Fraunhofer IPA