Visualization and Interaction with Medical Data in Immersive Environments

Yngve Devik Hammersland

December 2008

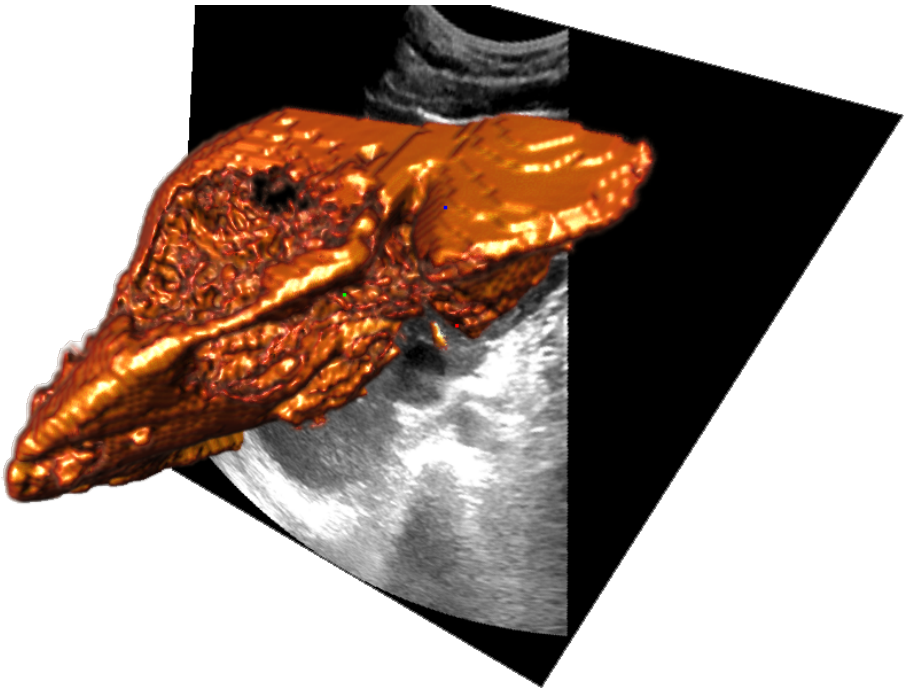
Visualization Group
Department of Informatics
University of Bergen

Master Degree Thesis

Visualization and Interaction with Medical Data in Immersive Environments

by Yngve Devik Hammersland

December 2008



supervised by Ivan Viola

Visualization Group
Department of Informatics
University of Bergen

Abstract

Immersive visualization techniques are just starting to see limited adoption in medical applications. The Visualization Group at the University of Bergen wish to expand its research efforts into such immersive visualization techniques. A new immersive environment installed at the University is meant to be utilized for this purpose.

This thesis presents a solution which enables the use of VolumeShop in the immersive environments for presentation of volumetric data and for combination of 2D and 3D medical imaging modalities. VolumeShop is a rapid prototyping environment for visualization techniques, which is often used by our visualization group.

For general use of the immersive environment, motion tracking support needs to be added to VolumeShop. This motion tracking support can then be utilized to realize the immersive visualization pipeline. This pipeline computes correct perspective projection based on the user's position. Additionally it implements intuitive gesture based interaction with the data using a handheld interaction device.

A multimodal visualization pipeline is described and implemented, which enables the visualization of 2D+time ultrasound images combined with MRI volumes. To realize this pipeline, components for motion tracking of the ultrasound probe, for synchronization of the time offset between the motion tracking and the ultrasound series, picking landmarks in the ultrasound slice and MRI cross-section, and computation of the registration transformation which spatially unifies the two of the modalities.

The result of this thesis is a general purpose motion tracking component as well as the two described pipelines.

Contents

1 Introduction

In recent times, the possibility to acquire volumetric data sets has become a reality. In certain domains it has even become commonplace, e.g. in the medical domain, 3D anatomical scans has partially replaced x-ray. With the availability of such data, the importance of being able to make sense of it in a timely manner has emerged.

The sheer size of volumetric data is staggering. Comparing an image to a volume with 512 units along each of the dimensions, shows how much the size of the data grows by adding an additional dimension of comparable size. For example if a image has $512^2 = 262\,144$ elements whereas a volume has $512^3 = 134\,217\,728$ elements; the volume contains 512 times more data than the image. The most common way to make sense of such data, is to reduce the volume to a set of images representing consecutive cross-sections of the volume. A volume becomes 512 images, and one inspects the images one by one, not being able to look at the big picture. This allows the inspection of the data with a very small amount of processing as volumes are usually stored as a sequence of images.

The science of visualization has the noble goal of making sense of the aforementioned big picture. As the name visualization implies, the data is presented to the user visually taking advantage of the processing power of the human visual cortex. Presenting all the available data may cause information overload, so one of the aspect of visualization is to reduce the amount of data presented. In addition it is essential to present only the important parts of the data, but still present the rest of the data to a lesser degree to enable exploration. This kind of emphasis on the important parts of the data is called focus+context [13].

In recent years, the advance of computers processing power and of dedicated graphics hardware in particular has skyrocketed. These advances have been driven by the consumer market, especially the computer gaming community, which is demanding more powerful graphics hardware to display 3D computer games with higher resolutions and with more details than before. This trend is further reinforced by the manufacturers, since the only real differentiating measure between graphics hardware is its performance. The reason for this is that the feature set have been more or less standardized by Microsoft's Direct3D and the

OpenGL programming interface standards. The result is that most computers sold today, for the domestic market, is powerful enough to perform visualization of volumetric data. On the other hand this has also resulted in reduced focus on high end features in consumer-grade hardware such as true support for stereoscopic displays.

Stereoscopic displays have advantages when it comes to presenting volumetric data. Humans use several depth cues to interpret the depth of the perceived images. The depth cues a monoscopic desktop setup is able to give us are occlusion, perspective, atmospheric fog and lighting. A stereoscopic setup is able to also give us binocular disparity which is the differences in the images perceived by the left and right eyes. Using the stereoscopic display to simulate binocular disparity enables us to use our depth perception to aid the processing of the volumetric data. In our case, we also track the users position which enables us to simulate parallax. That is, objects close to the observer appear to move faster than objects farther away when the user moves.

As a result of the consumer driven market for graphics hardware, readily available and cheap commodity hardware are sufficient to perform research on visualization. Cheap hardware and the lack of support for stereoscopic displays on said hardware, has caused visualization of volumetric data to be realized mostly on desktop, using monoscopic displays and rendering technology. This is evident in our own visualization group at the University of Bergen as most of their research is carried out on desktop systems.

In certain domains — such as seismic data exploration — stereoscopic immersive environments are being used in the workflow. The companies dealing with these kinds of data, are mostly in the oil and gas industry which have the resources to acquire the necessary immersive environments and to develop immersive tools and techniques.

In medicine, monoscopic computer aided diagnosis is mostly used, however there is active research in image guided surgery and mixed medical reality approaches. Such techniques are used especially under neurosurgical intervention which is performed using precise tracking and registration of the patient.

To further aid the understanding and exploration of volumetric data, interaction tools are used to interact with the data. Interaction is an important aspect of data exploration in immersive environments, and is the key to fully utilize immersive environments. An important aspect of interaction is responsiveness. The user is basically blind in the sense that she is not able to see the consequences of her actions until it is presented on the display. This problem can be reduced by using intuitive interaction tools, which increase the chance that the user knows what to expect. If the consequences of the user's actions are immediately apparent, it is much easier to pinpoint where features of interest occur. The responsiveness in an immersive system is defined by the time it takes from when the user moves, to when the display is updated to reflect this. Thus there are two latency factors in immersive environ-

ments; the time to get the position from the motion tracker and the time it takes to render the image and display it. In fact, if this lag is significant, it can cause motion sickness [14].

In addition to the visualization and data exploration mentioned above, a use of such an immersive environment can be used in a collaboration environment. Because of the size of the screen, it is possible to be used by more than one person at a time. Multiple users may collaborate in exploration of data or a user may give a prepared demonstration to a group of people. One example of this is medical use; where doctors may explore a 3D anatomical or functional scan of a patient while discussing the diagnosis.

1.1 Contribution

In this thesis an open-source solution for immersive stereoscopic volume visualization with motion tracking is provided. This thesis aims to use existing solutions where available and to adapt these to work with the rapid prototyping environment already in use within our visualization group. This will allow the use of existing research results in the immersive environment with no changes to the implementation. It will also enable the use of other existing immersive installations, e.g. equipped with optical or acoustic motion tracking.

Our visualization group has done a great deal of research in the fields of direct volume rendering and illustrative techniques amongst others. The research has been focused on visualization on the desktop. We often use the VolumeShop framework [5] for rapid prototyping which was conceived for use on the desktop.

There is ongoing collaboration effort with our clinical partners at Haukeland University Hospital in Bergen. A goal of this collaboration is to combine multiple modalities in visualizations to aid both in diagnosis and treatment planning.

The data acquired from the medical acquisition devices usually does not include information about its spatial-temporal frame of reference. Most data are specified in some implicit frame of reference relative to the acquisition device itself. This means that multiple modalities acquired from different sources are specified in different frames of reference, some of which may not be known. What is needed for combined visualization, is for the data to be specified in a common reference frame, preferably relative to the patient or the object under investigation. Thus multiple modalities must be aligned by changing their frames of reference, so they can be effectively combined. This process of aligning data sets is called registration.

Until recently, there has not been an immersive environment on site at our university. This changed in mid 2007, when the Institute of Informatics at the University of Bergen installed a new immersive environment. It consists of a back-projected stereo wall and a set of magnetic motion trackers.

After this immersive environment was installed, work on development of visualization and interaction techniques for immersive environments could begin. There were still many problems left to be solved, most of which is addressed by this thesis. Of these problems, the biggest was the lack of a software platform for both the stereo wall and the tracking equipment. The installation of this environment and the medical collaboration is the main motivations for this thesis, as it aims to supply the missing components and tries to accommodate existing research. Support for a wide range of tracking equipment will enable the use of this platform on most immersive environments.

Figure 1.1: *A visiting pupil with the head tracker mounted onto a helmet and the hand-held intraction device interaction with a CT scan of a head in the immersive environment*

Using the described solution, we will demonstrate its application in real world scenarios; including in immersive environments and in medical domains to track acquisition of medical data. This thesis has three main contributions which are the following:

1. We demonstrate its use in immersive environments using head tracking and hand tracking. In this scenario, we simply add motion tracking to VolumeShop to allowing us to track the position of the users head and hand relative to the power wall. Using a custom plugin which computes the correct perspective projection for the users position enables us to render the volume as seen from the direction of the user. Such a projection will give the user an illusion of that the volumetric data is positioned in physical space.

   A second tracker is used to accommodate the interaction. We implemented a simple form of interaction, namely letting the rendered volume follow the gestures of the user. The combination of the viewpoint-dependent perspective and the gesture based interaction, enables immersive visualization of volumetric data. See Figure 1.1 for an example of real-world usage.

2. We demonstrate the use of the trackers alone by tracking the position of ultrasound probes. Such tracking allows us to annotate the ultrasound images with its position and play back the ultrasound session by loading the annotated data. This playback is an invaluable tool for research on these kinds of data, as it allows testing of solutions without having to perform an actual ultrasound imaging session.

3. We demonstrate further use of the annotated ultrasound images mentioned above by performing registration of the images with a 3D anatomy scan. Basically this means to position the ultrasound images in the 3D anatomy scan where they were acquired.

These demonstrations will ensure that the software platform provided by this thesis, is general enough to be used for both multimodal visualization and immersive environments. The pipelines and their components will also form implementations on which further work can be based on.

1.2 Thesis outline

Chapter 2 starts out by describing the basics of tracking and registration, as well as some work which has been done in the different fields utilized in this thesis. It continues to give a high level overview of the acquisition and visualization pipelines implemented in this thesis and describe the steps it consists of in Chapter 3. The steps of these pipelines include data acquisition, both medical and motion tracking, registration of said data, as well as the visualization, interaction and immersion. Chapter 4 will go into further details of the steps described in Chapter 3. In Chapter 5 it will present the results of this thesis and Chapter 6 summarizes the work, draws a conclusion and sheds some light on possible future work.

2

State of the Art

This thesis will focus on medical multimodal visualization and immersive visualization techniques. First some background material for the reader is presented, in particular a short description of homogenous coordinates and matrices, as this thesis is using homogenous matrices extensively for its spatial transformations. Then related work as well as display, motion tracking and acquisition technologies being touched upon are presented. Section 2.2 presents an overview of the existing tracking devices and middleware solutions for interfacing said devices, Section 2.4 will present related work in immersion and Section 2.5 will describe techniques relevant for multimodal medical registration.

2.1 Background Material

Homogenous coordinates and matrices are utilized in the pipelines, most notably in Section 3.3.2 which deals with perspective projection matrices. A basic knowledge of this topic is needed to fully understand the presented formulas. See [3] for more thorough discussion on homogenous coordinates and matrices.

In elastography, or other medical imaging technologies, a reference object is often used to calibrate scanners. The reference object used in this thesis, called an elastography phantom, will be mentioned in Section 5.3.2 and is described in this section.

2.1.1 Homogenous Coordinates and Matrices

Graphics pipelines, such as the OpenGL rendering pipeline, usually uses matrices to perform transformations for example rotations and translations on vectors and vertices. In 3D space, using 3×3 matrices is adequate for most linear transformations but not all; translation is a linear transformation which cannot be represented by a 3×3 matrix. Homogenous 3D coordinates can be used to solve this problem. The homogenous coordinate representing

the point $[x, y, z]^T$ is defined by:

$$[x, y, z]^T \longrightarrow [wx, wy, wz, w]^T \quad \text{for all } w \tag{2.1}$$

This means that homogenous coordinates basically represents each point in 3D space as an unique line in 4D space identified by its direction. To convert a homogenous 3D coordinate to a regular 3D coordinate, one divides all the other components by w:

$$[x', y', z']^T = [\frac{x}{w}, \frac{y}{w}, \frac{z}{w}]^T \tag{2.2}$$

In practice only two values for w are used; 1 and 0. Coordinates where $w = 1$ are handled as vertices whereas coordinates with $w = 0$ are handled as vectors. Other values for w can also be used for special purposes such as perspective projection

A homogenous 3D matrix consists of four homogenous coordinates and as such it is 4×4 in size. The first three columns are usually vectors ($w = 0$) which represents any rotation, scaling, shearing, etc. transformations the matrix might contain and the last column are usually a vertex ($w = 1$) which represents the translation. Other values for w are also possible, but are rarely used. For example, projection matrices usually lets the third column vector z have $w = -1$.

It turns out that using homogenous coordinates, both perspective projection and translation can be expressed by homogenous transformation matrices. For perspective projection matrices, the basic principle is to define a transformation matrix, which encodes the distance from the camera in the w coordinate. The division by w performed when converting the homogenous coordinates to regular coordinates will then scale objects farther away accordingly. For translation matrices, the fourth column vector contains the translation. This vector will be added to transformed vertices, because a vertex's $w = 1$, and not added to vectors, because a vector's $w = 0$. The primary reason for using homogenous coordinates and matrices in computer graphics is to unify rotation, scaling, translation and perspective projection into a single type of transformation; matrix multiplication.

2.1.2 Elastography Phantom

The elastography phantom [8] will be encountered later in this thesis and will be shortly described here. The phantom is an elasticity reference tool. This means that it is basically a box shaped rubber object with calibrated elasticity properties to simulate the elasticity properties of soft tissue.

The phantom is made of solid polymer with embedded spherical inclusions made of the same polymer, but with differently tuned elastic properties. The inclusions' hardness

is different from the surrounding material, but exhibit the same acoustic-reflective properties. The result is that the inclusions are almost impossible to spot using regular B-mode ultrasound, but are detectable using elastography. The phantom is therefore suitable as a reference for elastography scanners.

2.2 Motion Tracking Technologies

Tracking is an essential part of an immersive environment. The user of immersive environment needs to be tracked to produce images to provide the illusion of immersion. There exists a number of different manufacturers of tracking equipment. Amongst them are Ascension Technologies which has produced the equipment used in thesis and NDI, vendors of mainly optical and magnetic tracking systems.

2.2.1 Tracking Devices

There are several techniques used to determine the position and orientation of the trackers, each with their own advantages and disadvantages. The main techniques that are used today are described here:

Magnetic Tracking: Magnetic trackers are usually based on a transmitter and a tracker which both consist of three coils oriented perpendicular to each other. The transmitter applies an electric current to these in order to generate an electromagnetic field which is registered by the coils in tracker. The position and orientation of the tracker can then be deduced from the strength of the received magnetic field in the different coils.

The advantages of magnetic tracking are that it does not require line of sight and the tracker can be miniaturized by using small coils. The disadvantages are that it is sensitive to the presence of metals in the proximity [19] and it is affected by temperature changes due to the electrical components.

Systems using alternate current magnetic fields are more sensitive than systems using direct current [1]. Magnetic motion tracking systems available today include Polhemus' Liberty Latus and Ascension's Flock of Birds.

Optical Tracking: Optical tracking systems usually consists of one or more cameras and one or more optical markers. In some systems, the cameras are stationary observing the position of the marked trackers. In others, the camera are mounted on the tracker, such as a helmet for head tracking, and the markers are fixed.

The markers can vary in size and shape relying on its perceived shape to recognize its orientation. Markers may also be active, in the sense that they emit light for the cameras to pick up, such markers are usually infrared so their emitted light does not distract the user. Today, optical tracking setups generally utilize infrared LEDs, which are reflected off hyper-reflective markers instead of utilizing active markers.

The advantages of optical tracking is that it is insensitive to temperature and humidity. Their disadvantage is that the markers may be occluded by the user or other objects in the working area. Adding more cameras reduces the occlusion problem as the cameras observe the marker from different directions. Optical tracking systems include the Spectra by NDI, the HiBall by 3rdTech [26] and the laserBIRD by Ascension.

Ultrasound Based Tracking: Ultrasound trackers work by emitting ultrasonic sounds at predefined intervals. The receiver knows the predefined intervals and is able to compute the time the signal used to reach the receiver. Several receivers are then able to triangulate the position of the tracker. To be able to compute the orientation, the tracker needs three transmitters.

Ultrasound Based Tracking generally suffers from high latency due to the speed of sound in air. Thus such trackers are very dependent on the air temperature, humidity and pressure. Available ultrasound tracking systems include the IS-900 by InterSense.

Mechanical Tracking: Mechanical trackers relies on a physical connection to the tracked point. They are usually constructed by a jointed arm where the angles of the joints are measured. Mechanical Trackers are usually accurate and has little latency but can constrict the user's movement because of the jointed arm. Fakespace makes mechanical trackers, i.e. the Boom.

Mechanical tracking is not widely used any more, except when haptic feedback is required. Available mechanical trackers include the Phantom Omni and Phantom Desktop by SensAble and the Falcon by Novint.

Inertial Based Tracking: Inertial based trackers utilize tri-axis accelerometers to provide the projection of the gravitational field onto the three axes. These three magnitudes are used to compute the trackers orientation much like a magnetic tracker does.

Pure inertial based trackers exists, but generally provides only the orientation due to the lack of a point of reference. This lack of positional tracking is their main disadvantage. The advantages of inertia based trackers is that it is not affected by anything; the force of gravity is constant. The exception is that they are affected when moved as applied acceleration will accumulate on top of the gravity. Available inertia based trackers includes Intersense's InertiaCube product series.

Hybrid Trackers: Hybrid tracker are also available. The most common combination is to combine optical tracking with inertia based tracking to add positional data to inertia based trackers.

The Nintendo Wiimote has this combination. It has a built-in accelerometer for orientation and an infrared camera for position. The sensor bar used in combination with the Wiimote is basically two LEDs which serves as a point of reference. Combining these measurements, it can compute its position and orientation relative to the reference points. Its main advantages is that it is cheap and wireless. On the other hand it is not very accurate, and fails function when the reference points stray outside of the trackers field of vision. Another example of available hybrid trackers are Intersense's IS-1200.

This concludes the overview of the different tracking techniques and some of the available devices. The next section will look at motion tracking middleware solutions.

2.2.2 Tracking MiddleWare

The tracking systems in general needs a software driver to be usable in a software system. Most tracking systems use RS-232 communications ports or ethernet for communication with the client computer and their protocol is often simple. Thus it is feasible to write a custom component for interfacing the tracker.

A middleware library will ease the development of the motion tracking interface, especially if several trackers are used or the immersive environment consists of multiple computers. Middleware solutions generally support most available devices and can be configured by configuration files or run-time by the client program directly.

In this section, a number of available middleware solutions are presented along with some of their advantages and drawbacks. The open source solutions will be presented first followed by a commercial alternative.

Open Source

This thesis focuses on open source motion tracking libraries as they offer the advantage of high customizability. There exists a number of virtual reality toolkits, but since this thesis uses VolumeShop for the visualization, pure tracking device libraries are focused on. The most prominent open source motion tracking libraries are:

Gadgeteer: Gadgeteer is the input manager of the VR Juggler suite [2]. It provides a high-level abstraction from devices categorizing input into the following abstract categories; analog, command, digital, glove, gesture, position, simulator and string. For

example an analog joystick would fall into the analog category, whereas a position tracker is in the position category.

Gadgeteer is tightly integrated into VR Juggler and there exists no documentation that the author of this thesis is aware of that states that it is possible to use it separately from VR Juggler.

Virtual Reality Peripheral Network (VRPN): VRPN [22] is, as its name indicates, a virtual reality peripheral network. That is; a networked library to connect to virtual reality peripheral devices, such as motion tracking devices. VRPN is cross platform and runs on Windows, Linux and a number of different architectures including Linux-based PDAs. It is written in C++ and has bindings to Java and has recent acquired bindings to the .NET platform allowing it to be used in programs written in C#, Visual Basic.NET, etc.

VRPN has separated the client and server components. It can be run in-process and be called directly from client code or as it is designed to do, in its own process, communicating with the client through the network. The server may be hosted on an entirely different machine and it is all handled transparently.

OpenTracker: Similar to VRPN, OpenTracker [21] is a fully networked tracking interface with dynamic configuration of the devices. The configuration is stored in the Xml format. The OpenTracker configuration is generally a directed acyclic graph which acts as a data flow graph and consists of data sources, data sinks, filters and transformations. In fact, OpenTracker has VRPN sink and source so it can act as a VRPN client and server if needed enabling full interoperability between these two libraries.

OpenTracker is possible to set up on several computers, where one or more computers can connect to the trackers and act as data sources supplying a second set of computers with the data. To realize transparent network operation, OpenTracker defines a network sink and a network source. All this is configured at run-time with configuration files in such a way that there is no difference for the application whether the trackers are connected locally or not. As the tracking setups dealt with in this thesis are directly connected to the main computer, these features were not utilized in the scope of this thesis.

The main difference between OpenTracker and VRPN, is that OpenTracker is not split into a server and a client part. The ability to configure network mostly resolves this problem.

Commercial Alternatives

One of the commercial alternatives to the open source libraries just mentioned are trackd. Trackd is a standalone commercial library for interfacing motion tracking devices made by VRCO. It is implemented as a daemon, a background process, which makes the motion tracking data available to other applications. It also ships with a server executable which reads data from the daemon and makes it available over the network.

Trackd supports a wide range of operating systems and architectures and a wide variety of tracking devices. It is used by VRCO in its all virtual reality software solutions, e.g. CAVELib and VRScape.

Decision for Tracking Middleware

The tracking library used for this thesis should be open source, making it possible to fix, modify or extend the library if needed, and additionally, be free of charge. This ruled out the commercial alternatives. Gadgeteer was ruled out due to its integration with VR Juggler. The final choice stood between OpenTracker and VRPN. Finally OpenTracker was chosen due to its support for the wiimote and active cooperation with the OpenTracker development team of Graz University of Technology, even though VRPN is marginally the better technical solution, mainly due to the client-server separation.

It would be possible to use VRPN and then connect it to an OpenTracker server connected to the Wiimote, but it is preferable to only use one of the solutions. Thus OpenTracker was chosen as the motion tracking middleware for this thesis. Chapter 5 will describe the results of this choice.

2.2.3 Tracking in the Medical Domain

In the medical domain, the most important characteristics of tracking is precision and low latency. Precision is needed because tracking is generally used to determine whether the intervention tools are in the proximity of arteries and such to avoid damaging those structures. Low latency is equally important since it is imperative that the surgeon receives immediate feedback from her actions. This also places constraints on the latency of the visualization.

Optical tracking is often used because it has generally low latency and is very precise. In non-invasive surgery, the probe is usually occluded by the patient and in such scenarios, optical tracking is unfeasible. In such cases, magnetic tracking is often used. For minimally invasive robotic surgery, mechanical tracking which mirrors the joints of the robot is often used. Thus the urgeon is restricted from performing movements which the robot is unable to perform. One example of such robots are the DaVinci robotic system by Intuitive Surgical Inc. [12].

2.3 Stereoscopic Display Technologies

Stereoscopic displays are able to present independent images to the user's two eyes. The basic premise is to present an image to each eye, which presents the scene from the direction of the corresponding eye. For this reason, several methods for filtering out the "wrong" image exists. The two main categories of filters are active and passive.

A category of displays, which will not be discussed in detail in this thesis, are head-mounted displays. Head-mounted displays generally mount one screen in front of each eye or use other techniques to directly project the image onto the retina. Traditional displays which are discussed later, needs some kind of filtering to separate the right and left images. The two main filtering techniques are discussed next:

2.3.1 Filtering

Active filters actively changes their state. In stereoscopic display technologies, the display presents the image for the right and left eye sequentially, while the filters are toggled synchronously so that the left eye only observes the left image and equally for the right eye. A problem with this is method is that the refresh rate of the display is effectively halved and may cause eye strain since the eyes are in the dark 50% of the time. The advantage of properly synchronized active filters is that they always filter out 100% of the light from the wrong image which passive filtering does not always achieve.

Passive filters passively block the emitted light usually by employing polarized filters. Polarized filter can be compared with the red-cyan glasses used with anaglyphs. The difference is that they filter vertically or horizontally aligned light instead of colors. Polarized filters can also filter right handed and left handed circulating light which makes them invariant to head tilting. Passive filters needs to have two images presented simultaneously each filtered for the respective eye. The advantage is that they operate in the full refresh rate of the display but may suffer from bleed through, that is; when some of the light reaches the wrong eye. Passive filters are usually paired with back-projected screens which have the screen made of polarization preserving material.

Active and passive filtering may be combined to cater for more than one user [10]. On a related note, solutions exists for three or more users [15] by using a mask and letting each user only see a portion of the display.

2.3.2 Displays

Immersive displays usually consist of back-projected screens with one or more projectors. The reason for this is mainly the size of the displays which leaves back-projection displays

the only feasible option. Front projection is also becoming more common as canvas to cope with the polarization has become available. It has been more difficult to make polarization preserving reflective canvas than to make polarization preserving pass-through canvas.

To increase the resolution of the display, multiple projectors may be used on a single screen. In this configuration, they usually overlap a bit, fading out in the edges to keep the intensity constant. Such multi-projector setups are also utilized on curved screens. Barco's MoVE environment consist of a single screen and three projectors side by side. The side projectors along with a bendable screen can be configured at run-time to be a flat screen (0° bends) to be a three sided CAVE (90° bends).

A CAVE environment [9] is a name given to a setup consisting of three or more back-projected stereo screens configured to enclose the user in a box like fashion where the displays acts as the walls, floor or roof of the box. For full immersion, all six sides are replaced by screens. Such setups are very expensive, usually built into the infrastructure and are not very common.

Autostereoscopic displays are a recent development in display technology and are just now becoming available as consumer products. The basic premise, of such displays is that they direct the emitted light towards the observing eye and are able to direct two different images in two different directions simultaneously.

The Varrier [23], is an immersive stereoscopic environment which does not require the use of trackers or filters. It is a complete system using infrared cameras to track the position of the user, and with the help of this position project the images to the position of the eyes using autostereoscopic displays.

Consumer grade autostereoscopic displays generally do not include head tracking and have a fixed position in front of the center of the display where the user needs to be located to experience stereo, otherwise both eyes will perceive the same image.

2.4 Visualization Techniques

To visualize volumes, one needs to fit the data into the rendering pipeline, e.g. OpenGL. Graphics pipelines are geometry centric in the sense that they only render geometric primitives such as triangles. More complex geometric structures are usually represented by a collection on triangles.

The obvious way to display the data, is to convert it to geometric structures. This can be done by extracting isosurfaces from the volume or by representing the volume as a set of cross section slices textured by a 3D texture containing the volume data [27].

Direct volume rendering (DVR) techniques render the volume directly without converting it to geometric structures first. Ray casting is a method for direct volume rendering,

which casts rays from the view-point through the pixels of the screen, while sampling the volume along the length of the ray. DVR used to be implemented in software, but with recent advances in graphics processors, they have become increasingly programmable and powerful, it has become possible to perform ray casting on the graphics processor.

In DVR, transfer functions are used to map intensities of the measured scalar field to optical properties such as color and opacity. It is also possible to utilize multi-dimensional transfer functions [17] and [16] with which it is possible to map multiple properties to optical properties, i.e. the intensity combined with the length of the gradient vector.

Transfer functions do not need to map from voxels to color and opacity, for example, style transfer functions [6] may be used. Style transfer functions utilize style spheres which encode the color and opacity based on the normal of the data relative to the camera. The normal is used to look up the color, thus it is possible to encode the lighting and shading in the sphere.

Illustrative techniques, also called non-photo-realistic techniques, have also been researched and aim to replicate hand made illustrations. This is a surprisingly difficult task. A technique utilizing style transfer functions to achieve illustrative results has been made [4]. Another technique using the gradient to find contours have also been made [7].

This thesis makes use of color-opacity transfer functions to create the images presented in Chapter 5

2.5 Registration Techniques

As noted in the introduction, registration is the process of aligning data sets so that they share the reference frame. The different registration techniques utilize different methods and work on different sets of data. The main categories of registration techniques are;

Voxel Property Based: Voxel property based techniques operate directly on the intensities in the data sets and utilizes image processing techniques to identify mutual information. Such methods are usually fully automatic, but do not always produce good results.

Segmentation Based: Segmentation based techniques rely on an existing segmentation of the data set and aligns the segments defined by the segmentation. The registration obtained using such techniques are only as good as the initial segmentation.

Landmark Based: Landmark based techniques depends on finding corresponding landmarks in both data sets and aligning these using various methods. Three or more landmarks needs to be identified in order to align the data.

Landmark based registration is used in this thesis and the landmarks are supplied by the user. Voxel property based techniques are most widespread, but require more computations to perform, especially when registering 3D modalities. Previous work has been done in the area of registering ultrasound images against CT volumes using voxel property based methods [25].

A thorough overview of voxel property based techniques, was made by Pluim et al. [20] and an overview of all kinds of techniques used in the medical domain by Maintz et al. [18]. Existing techniques can operate on both 2D or 3D modalities or a combination of both.

Rigid versus Nonrigid Transformations

A rigid transformation is basically a linear transformation and means that one modality is linearly transformed into the reference frame of the other modality. Linear transformations do not deform the shape of the data. Nonrigid transformations apply local transformations to areas in the data set and necessarily deform the data set. Sometimes this deformation is desired, i.e two volumetric anatomical scans of a patient will be impossible to register if the patient is not positioned exactly the same manner for the different scans. Nonrigid transformations are able to align the data even if the data sets differ in shape. Nonrigid transformations are also much more difficult to effectively implement, and requires more information to compute than a rigid transformation. Both kinds of transformations can be used in voxel property based techniques as well as point based techniques.

2.5.1 Modalities

As noted, the acquisition of modalities can result in both 2D images and 3D volumes. Examples of 2D modalities are ultrasound and X-ray imaging and examples of 3D modalities are computed tomography (CT), positron emission tomography (PET) and magnetic resonance imaging (MRI).

In addition they may be time-varying so that they effectively become 2D+time and 3D+time. Time-varying data may also need to be registered from time step to time step if the scanner is not fixed in place in relation to the scanned object. Hand-held ultrasound is an example of a mobile device not spatially fixed in relation to the patient. Here previously reviewed tracking technologies become widely utilized.

In the scope of this thesis, the modalities used were ultrasound images and elastograms registered with MRI volumes. The ultrasound images needed to be spatially defined which was accomplished by attaching a magnetic tracker to the ultrasound probe. A simple landmark-based technique was used as a proof of concept.

The next chapter will describe the pipelines for immersive visualization and multimodal

visualization, which is the theoretical foundation for procedures and results obtained in the scope of this thesis.

3

Immersive Visualization Pipeline

3.1 Overview

As mentioned in Section 1, the need for visualization and immersion increases as the amount of available data increases. The amount of data has long since surpassed the abilities of the human perception and only specialists are able to interpret the data. A host of visualization techniques have been developed and still more are being developed to help humans make sense of the data. Visualization of spatial data, volumetric data in particular, can take advantage of stereoscopic displays to aid in the understanding of the data. Stereoscopic displays combined with motion tracking of the user has great potential to convey spatial relationships within the data.

The collaboration with Haukeland University Hospital in Bergen and the Visualization group of the Institute of Informatics at the University focuses on combining modalities in a single representation. The first issue which needs to be tackled is to align the reference frames of the different modalities to enable them to be presented in the correct positions in relation to each other.

This thesis aims to address both of the above scenarios; the immersive visualization and the combination of the different modalities. Thus there are two different but overlapping pipelines made in this thesis. The first is the motion tracking pipeline used for the immersive visualization whereas the second is concerned with the registration of the volumes with the ultrasound image slices.

Both use the same acquisition device for the motion tracking but in different configurations. They are also both implemented as a set of plugins to VolumeShop so they may easily share implementation details. Otherwise they differ greatly and are in principle two unrelated systems.

3.1.1 Immersive Visualization on Large Screen Stereo Display

One of the goals of this thesis is to enable the user to interact with 3D medical volumes – including manipulation, clipping and changing transfer functions. In an immersive environment the user is usually standing in front of the wall experiencing stereoscopic projection from the large screen. The stereoscopic screen is used to display different images for each eye. The computer computes the correct perspective for each eye and applies it to the data so that the user sees the same scene with both eyes but with a slightly different perspective. The user's position has to be known for the computer to be able to figure out the correct perspective. To be able to interact with the immersive environment the user needs an interaction device which tracks her position in three dimensions.

The immersive environment at our institute is equipped with a motion tracking system with two trackers attached, see Figure 3.1 for details. The two trackers are used to enable

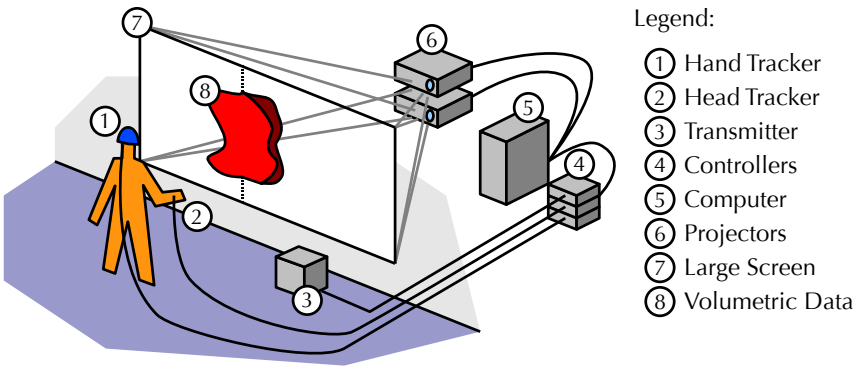
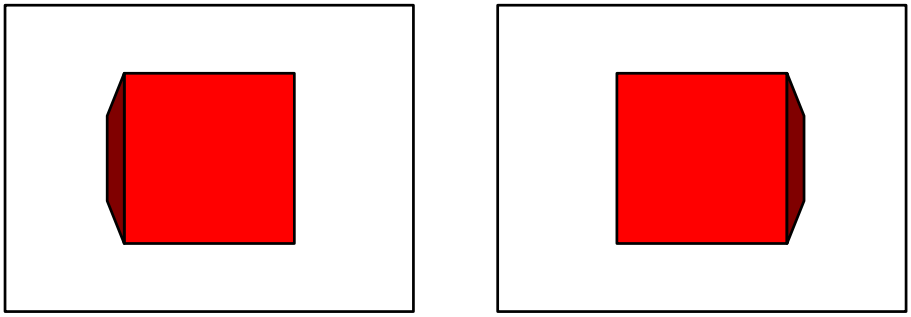


Figure 3.1: *The immersive setup at our institute*

immersion and interaction. The first of these is used to track the position of the user's head enabling the display to reflect the location as illustrated in Figure 3.2. This will enable the user to move around and even into the data. The second tracker is used to track the position of the user's hand enabling the user to interact with the data using physical gestures. The combination of the interaction and immersion will make the user feel he is dealing with real objects.

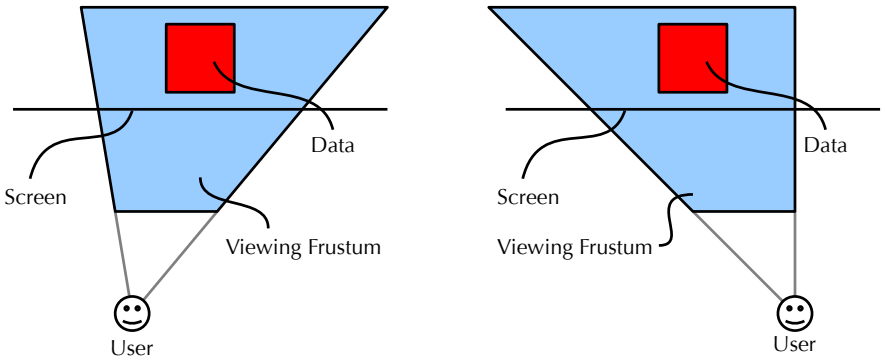
The viewing frustum is the volume the user can see within the frames of the display. It is the pyramid formed by the viewpoint and the screen. If the viewing frustum is updated to follow the user's position as illustrated in Figure 3.3, it will enable the user to move around the data and observe it from several directions. This can be compared to looking through a window rather than looking onto a painting. It is even possible to place object in front of the window which will make the data appear to hover in front of the display.

For interaction, the user will hold the other tracker in his hand allowing the software to



(a) User at the left of the screen (b) User at the right of the screen

Figure 3.2: *The images produced with regards to user position*



(a) User at the left of the screen (b) User at the right of the screen

Figure 3.3: *Viewing frustum computed with regards to user position*

track its position and orientation. The tracker supplies its absolute position and orientation. We can use this to compute the derivative, that is, the amount of motion currently being applied.

Both the absolute and the derivative transformation can be used for interaction. The absolute may be used for pointing, like a laser pen, whereas the derivative can be used to interpret gestures, like making the data follow the user's movement as illustrated in Figure 3.4. Alternatively it may be used to mimic a knob to enable the user to dial in a value or mode, i.e. the user rotates her hand clockwise to zoom in, counter-clockwise to zoom out. These are just a few of the interaction possibilities available when using motion tracking equipment.

To realize the immersive visualization a set of components are created which transforms user input from the trackers to the images presented on the display. See Figure 3.5 for an overview of the components and the data flow from user to image.

As described in Chapter 2 OpenTracker is an open source middleware solution for track-

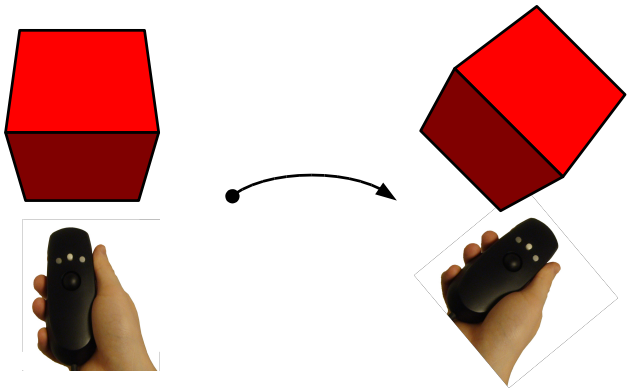


Figure 3.4: *The data follows the gestures of the user*

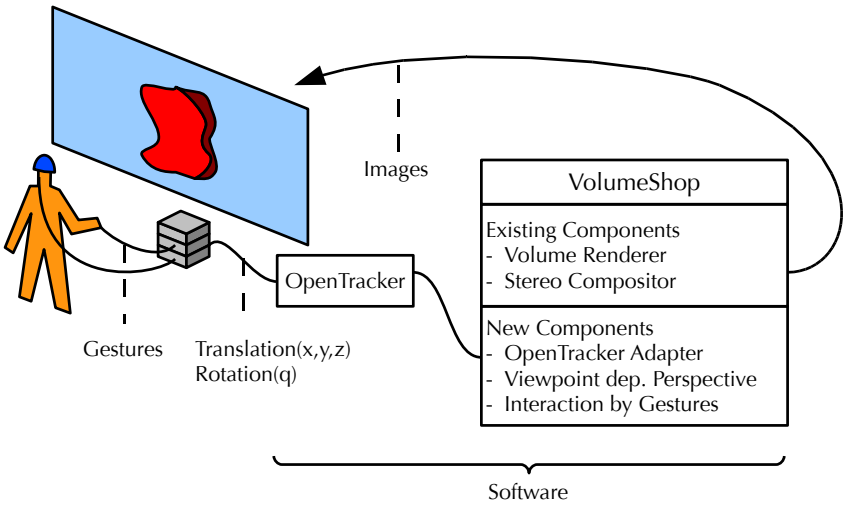


Figure 3.5: *The data flow for the immersive visualization pipeline*

ing which supports a variety of tracking devices which have been chosen to use for this thesis. It supports dynamic configuration of the trackers through xml files. VolumeShop is the application framework used by our visualization group which supports rapid prototyping of visualization applications.

The position of the user's head and hand are picked up by the transmitter and sent to the controller. The positions are then converted to binary form to be transmitted to the computer. The receiving application has an instance of OpenTracker running which is set up by a configuration file to receive data on the appropriate communications port.

The OpenTracker adapter component created for this thesis is given the position in the form of a position vector and the orientation in the form of a quaternion. For further processing, the position and orientation is converted to two separate homogenous matrices representing the respective transformations. The final step for this component is to make

the combined transformation available to the other components. The transformations are named with the names given in the OpenTracker's configuration file so they can be handled separately by the receiving components.

The component computing the viewpoint dependent perspective observes the head tracker's position and computes a corresponding perspective transformation. This component also needs to know the size and position of the screen relative to the reference frame of the trackers to be able to compute the correct perspective. The computation of the perspective projection transformation is discussed in greater detail in Section 3.3.2.

For the interaction, a component similarly observes the transformation of the hand tracker. Whenever the hand transformation changes and the user has trigged the interaction mode, the data transformation is updated to mirror the changes to the hand transformation. The result of this update is that the object on screen follows the user's gestures. The gesture interaction component is described in greater detail in Section 3.3.1.

3.1.2 Visualization of 3D Modality with 2D Ultrasound

As mentioned in Chapter 1, through the collaboration with our clinical partners, there is a desire to perform multimodal visualization of medical data – volumetric modalities combined with 2D ultrasound in particular. It was also noted that to successfully combine data, their relation between the reference frames needs to be known, and this relation is generally unavailable. To realize the combined visualization three problems needs to be addressed.

1. The annotation of medical data with their orientation and position in relation to a stationary frame of reference. This is only necessary for time series acquired using mobile or handheld scanners such as ultrasound probes where the spatial relationship between scans at different times needs to be known.

2. To register the different modalities. That is; to align their frames of reference in such a way that the features in the different data sets aligns.

3. To visualize the combined modalities on a display.

It was decided to use VolumeShop as the framework for the creation of the solutions and to use OpenTracker as the provider of the tracking input. As noted in the previous section, VolumeShop and OpenTracker is also used for the immersive visualization, and using the same for both pipelines will enable sharing of common components. The most prominent of the shared components is the OpenTracker adapter component. An overview of the setup can be seen in Figure 3.6.

In addition to the motion tracking acquisition we also need to acquire data for the registration of slices against volumes, namely acquisition of the time series of slices and the

acquisition of the reference volumes. During the course of this thesis, ultrasound and elastography imaging was used for the time series whereas MRI was used for the reference volume.

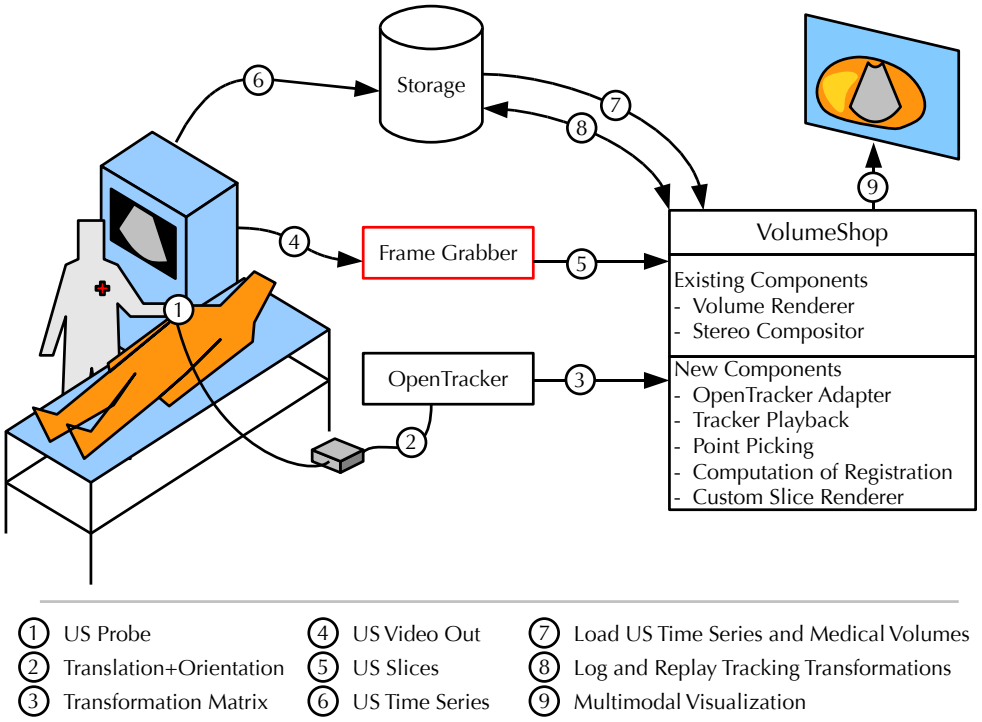


Figure 3.6: *The data flow for multimodal visualization*

To get motion tracked slices, we needed motion tracking of the ultrasound probe. This was solved by mounting the motion tracker onto the ultrasound probe, see Figure 3.7. The resulting position and orientation is then used to create a frame of reference in which to place the ultrasound slice. The placement of the ultrasound slice is based on physical measurements of the probe and its mounting point for the tracker.

As shown in Figure 3.6 it is possible to store the tracking transformations to a file. This file can be read back into VolumeShop for playback. Using this playback in conjunction with stored time series, the multimodal visualization of the session may be repeated at a later time. In the case of this thesis, the frame grabber pictured in the illustration was not available. Therefore, the process of visualizing the multiple modalities was split in two. First acquiring the tracking transformations and the ultrasound slices from VolumeShop and the ultrasound scanner respectively. Second reading the stored ultrasound time series and the tracking transformation for combination and visualization.

The following is a short explanation of the reference frames and matrices involved in the registration process. The goal of this pipeline is to render the slices combined with the

reference volume. If we were to render the volume only, the transformation stack would look like Figure 3.8(a). This matrix stack only contains the data transformation D. The graphics pipeline also implicitly contains the projection matrix P and the model matrix V which forms the full stack $P \cdot V \cdot D$. V and D are omitted from this description as they are irrelevant to the process of registration.

The matrix stack of the ultrasound slice illustrated in Figure 3.8(b) and more detailed in Figure 3.7. Pictured are the transmitter frame of reference T, the motion tracker M and

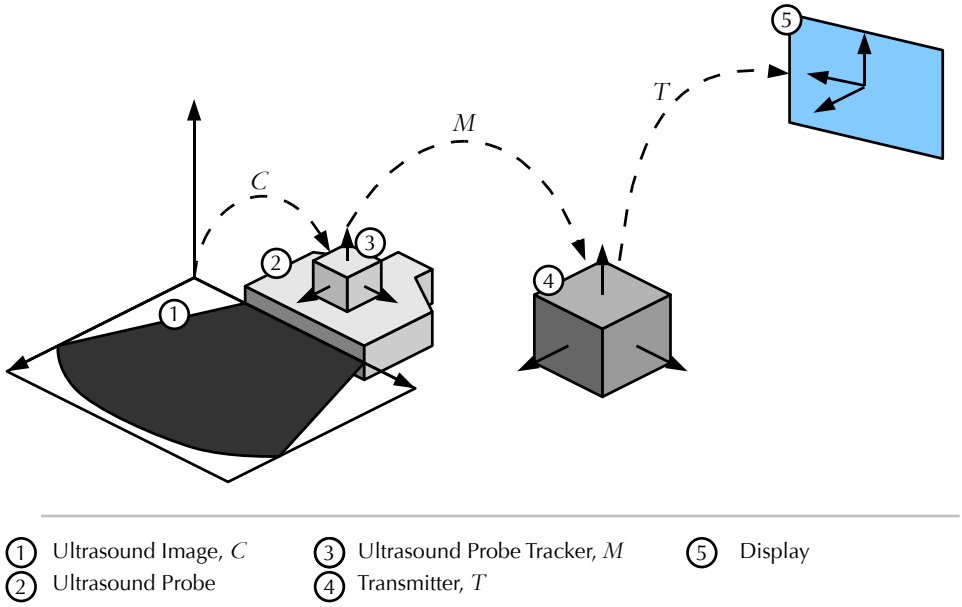
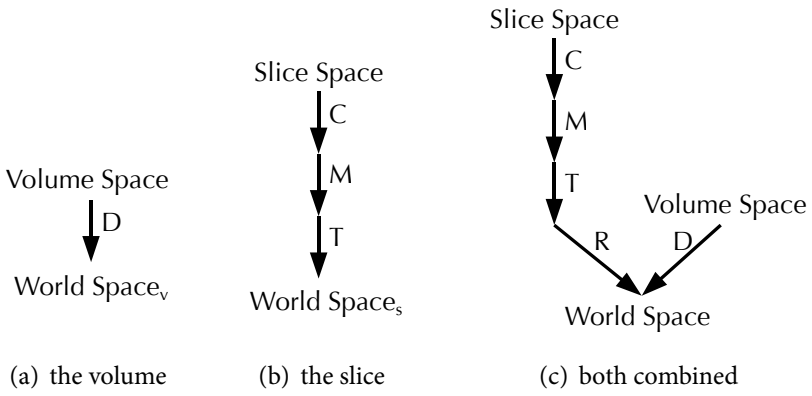


Figure 3.7: *The reference frames and matrices involved with ultrasound*

the probe compensation C instead of just D. These matrices model the spatial relations between the reference frames of the ultrasound image, the tracker, the transmitter and the display. Of these, only M will change during the course of a session, or eventually playback.

The system described so far, is fully able to visualize the ultrasound slices and the volumetric scan separately. It turns out, to combine them, the only thing necessary to do is to align the world spaces of the two modalities. The alignment is done by finding a transformation R, which maps the world space of the slice onto the world space of the volume. The complete matrix hierarchy for the registered ultrasound slice and the volume is shown in Figure 3.8(c).

Before the registration can be performed, the slice time series and the motion tracking needs to be synchronized. If they are acquired real-time, one might suspect that time synchronization is not necessary. Synchronization might still be necessary due to different latencies for the ultrasound scanner and frame grabber compared to the trackers. Laten-



(a) the volume (b) the slice (c) both combined

Figure 3.8: *The matrix stacks for the rendering pipeline.*

cies might also show up within VolumeShop since images might take longer to process than simple matrices do.

In the opposite case where the registration is performed off-line, the modalities must also be synchronized since the tracking and the ultrasound images is performed on separate devices which are not always equipped with synchronization facilities.

The synchronization is performed by manually specifying a time offset for the motion tracking data and also the number of slices per second. These two parameters combined define a customizable linear map between the time parameters of the slices and the motion tracking data.

The specification of these two parameters is done manually by the user. The frame rate of the slices is commonly a parameter of the acquisition device. In our case, it is stored in the DICOM files or overlaid on the image slices themselves. The time offset on the other hand is more difficult to specify correctly and deduced by the user by inspecting the transformed slices for correlations. This is obviously a difficult task and is usually carried out by trial and error. Therefore it is important to try to synchronize the start of the acquisition between the slice acquisition device and the motion trackers as much as possible. A nicely synchronized start can ease the synchronization by narrowing the search domain considerably.

To realize multimodal visualization, integrating 3D modality and 2D ultrasound, the user interface consist of three different views as illustrated in Figure 3.9. The first view (1) renders the ultrasound slice orthogonally scaled to fit. The second view (2) displays a slice of the volume in a similar fashion to view (1). In both of these views, the user is able to move through the data viewing the respective slices at any point independently of each other. In the volume by changing the z-position and in the time series of slices by changing the time parameter.

The third view (3) is the presentation view which renders the multimodal visualization. This view renders the reference volume using direct volume rendering combined with the

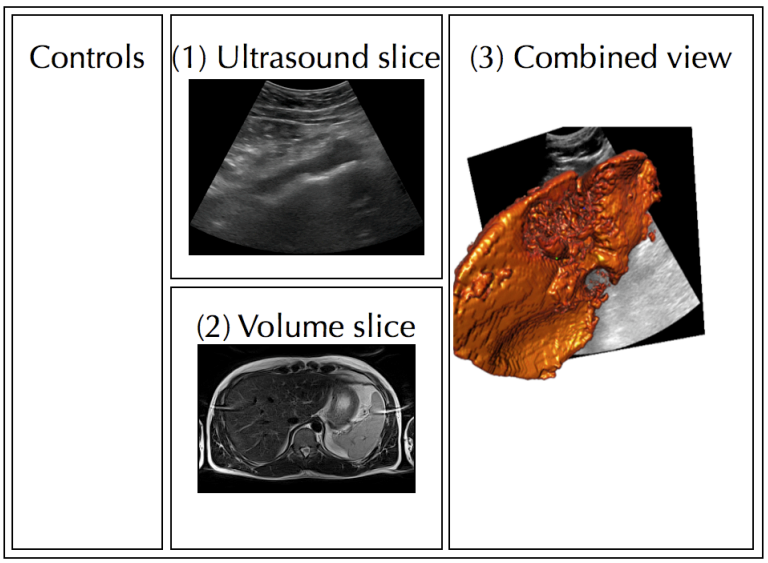


Figure 3.9: *A mock-up of the graphical user interface*

slices presented in the other two views. The slice from view (2) is positioned in the volume in its correct position to aid orientation. The ultrasound slice from view (1) is transformed using the computed slice transformation and registration matrix R to place it in the reference volume in its correct position.

To compute R, three corresponding points needs to be specified in view (1) and (2). These points are placed on corresponding landmarks in the ultrasound slices and the 3D volume. R is defined to be the matrix that maps the points specified in the ultrasound slices to the points in the 3D volume. Thus the landmarks needs to be specified in the same order for both modalities, otherwise the landmarks in the ultrasound slices will map to the wrong landmarks in the 3D volume. Finally when all three points are specified in both modalities, it is possible to compute R which maps the points in the slice time series to the corresponding points in the volume. This transformation is the rigid transform which registers the slices with the volume.

This concludes the overview of the pipelines. Following is a more detailed description of the pipelines and the components they consists of. As the medical data acquisition devices and tracking devices is shared by the two pipelines, it will be described separately in Section 3.2. Section 3.3 describes the components which make up the rest of the immersive visualization pipeline. Finally Section 3.4 describes the multimodal visualization components.

3.2 Acquisition and Tracking Devices

This section will shortly describe the modalities and the corresponding acquisition devices which have been utilized in this thesis. As already mentioned we have used motion tracking, ultrasound imaging which is also used for elastography imaging and MRI for the reference volume.

3.2.1 Motion Tracking

For motion tracking we have two Flock of Birds setups made by Ascension Technologies. One which is used in combination with the large screen immersive environment and a second used for the tracking of the ultrasound probe. Flock of Birds are magnetic tracking devices, which means they use a transmitter to apply a magnetic field to the area being tracked. The trackers have a much smaller but similar transmitter which registers the magnetic field. The position and orientation of the trackers are computed from the perceived magnetic field.

These trackers are coupled together using a proprietary bus called Flock of Birds Bus (FBB) and are connected to the computer using an RS-232 port. It would be possible to write a module to extract the data needed from the birds, but we opted to use an existing library to enable support for several kinds of trackers. There are basically three open source frameworks which deals with tracking equipment suitable for our application. These are OpenTracker, Gadgeteer and VRPN discussed in Section 2.2.

Gadgeteer is a part of the bigger virtual reality library VR Juggler and there was no documentation suggesting it could be used on its own. OpenTracker and VRPN on the other hand is a standalone fully networked motion tracking libraries which it supports a wide range of devices. OpenTracker also provides a bridge to VRPN so it supports all the devices VRPN supports.

At the outset of this thesis, there was a desire to experiment with the wiimote interaction device for the Nintendo Wii gaming console. At that time, only OpenTracker claimed support for the wiimote and thus OpenTracker was chosen. In retrospect, it is apparent that OpenTracker's wiimote support was not finished so this experimentation was put on ice indefinitely. At the time of writing, wiimote support has been developed for VRPN, so it might have been the better choice.

On the hardware side, there is the Flock of Birds setups. There is one installed at the immersive environment which is illustrated in Figure 3.10 and the stand-alone setup is mobile and is used for tracking the ultrasound probe. It consists of a tracker attached to a helmet (1) and a tracker built into an interaction device (3) which doubles as a hand-held mouse. The interaction device is shaped to hold in the hand like a remote control and has replaced

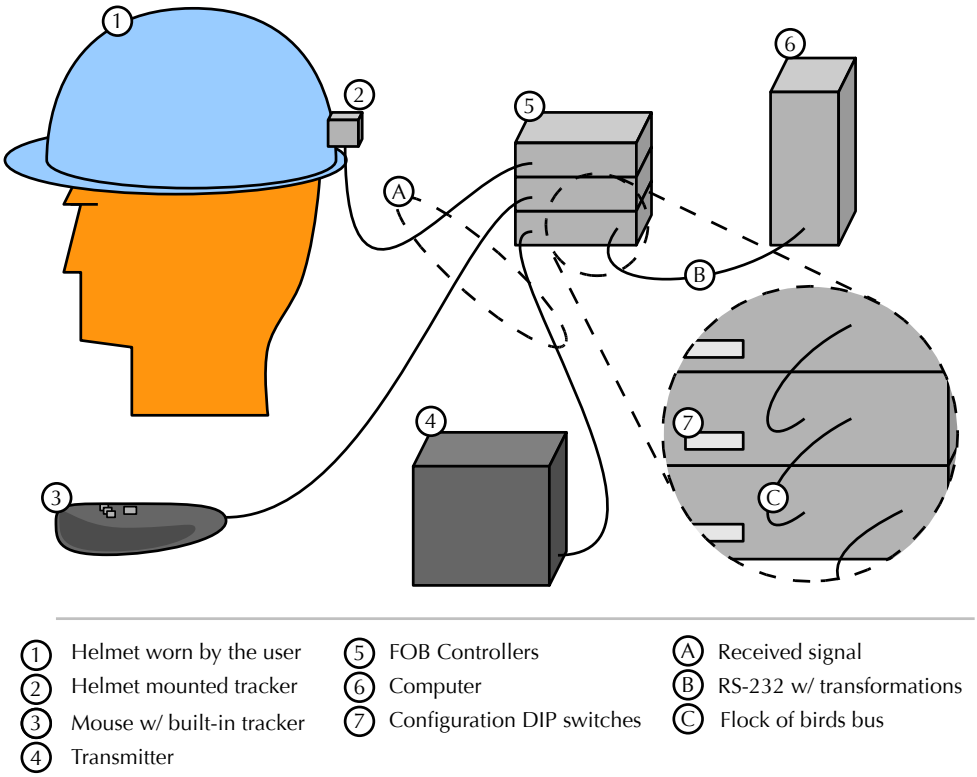


Figure 3.10: *The flock of birds setup*

the ball normally found on a computer mouse with a directional button on top. This button works similarly to a pointing stick found on laptop computers. Nudging this button any direction will move the cursor on screen in the same direction.

The trackers and the transmitter is connected to a corresponding controller (5). Thus we have two Flock of Birds controllers and one Extended Transmitter controller. The controllers are connected to each other with a proprietary serial bus (C) interface called Flock of Birds Bus (FBB). Note that the endpoints of the FBB needs to be electrically terminated and is done by setting a jumper located inside the controllers.

The controllers receive the magnetic signals directly from the trackers (A) and computes the position and orientation of each tracker. One of the controllers act as a master controller which the computer (6) communicates with through an RS-232 port (B). The baud rate of the RS-232 port is configured on the DIP switches (7) on the back of the master. The other controllers are addressed through the master. Each controller needs to be assigned an ID by setting the DIP switches on the back of the respective controllers. The master controller must to be assigned ID 1 and the other controllers must be assigned sequential IDs thereafter. See Figure 3.11 for the DIP switch configuration.

The Flock of Birds are also able to work in stand-alone mode. In this mode, a single bird is directly connected to a transmitter and must be assigned ID 0. This mode was not used during the course of this thesis because OpenTracker does not support the stand-alone mode. It does on the other hand, support a single tracker in group mode where it is assigned ID 1. Note also that OpenTracker needs the Extended Range controller to be connected as the last controller at the FBB, thus being assigned the highest ID.

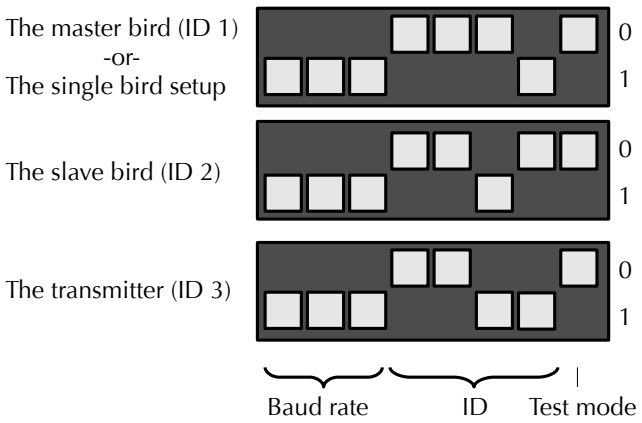


Figure 3.11: *The DIP switch configurations used for the two Flock of Birds setups*

After receiving the transformations from OpenTracker, they need to be transformed into the world space of the graphics pipeline. For the immersive visualization the world space equals the reference frame of the large screen. Thus, as noted in Section 3.1.1 the spatial relations of the tracker, transmitter, large screen and the tracked object needs to be known. The process of transforming the reference frame of the tracked object into world space is henceforth called compensation.

Assuming the transformations of the tracked object in relation to the tracker C, the tracker in relation to the transmitter M and the transmitter in relation to the large screen T. Then the compensated transformation M' is defined by

$$M' = T \cdot M \cdot C \tag{3.1}$$

The implementation of these are discussed in Section 4.2.1 and 4.2.3.

3.2.2 Acquisition of Medical Data

To realize multimodal visualization of medical 3D volumes and ultrasound sessions, data of the respective modalities are obviously needed. In this thesis, the modalities that are used are 3D MRI volumes and 2D+time ultrasound images. The acquisition devices and the process of moving the data modalities into VolumeShop for visualization is described in this section.

Reference Volume Acquisition

The reference volumes can be acquired from Magnetic Resonance imaging (MRI), Computed Tomography (CT) or other 3D anatomical and medical scanning devices. The scanning devices store the volumes in DICOM format. The volumes are then manually converted from DICOM to a set of images, where each slice is stored as a separate image. These images are then finally converted to a volume and stored in the `dat` format for easy loading into VolumeShop.

Converting the volume to `dat` format is not really necessary as VolumeShop can load volumes from a set of images, but it is more convenient to handle a single file than hundreds of images.

In short, the `dat` format is a non-standard format which is basically a raw volume prefixed by the dimensions. The individual values as well as the dimensions are stored in two bytes. The values only use 12 bits of the 16 available in the two bytes.

2D Ultrasound Acquisition

The ultrasound sessions were performed on a General Electric Logiq 9 ultrasound scanner for the ultrasound imaging and a Hitachi 900 HI VISION scanner for the elastography sessions. Both of them are ultrasound acquisition devices as the elastography are computed from the ultrasound data. The ultrasound and elastography time series is generally available in the same manner as the volumes; as DICOM files. It is also common to be able to burn the session as a DVD movie.

The Logiq 9 device is able to store the sessions directly to an USB thumb-drive in the DICOM format. From DICOM the time series are converted to images and finally to a volume in the `dat` format in the same manner as the MRI volume.

The Hitachi device stores the elastography sessions as DVD movies. The elastography time series are converted to images in the same manner as the ultrasound time series are, but they are not converted to `dat` volumes. The reason for this is that the elastography are color coded and as a color image it has 3 channels (red, green and blue) instead of 1 and VolumeShop's `dat` format importer only supports a single channel.

After this process, we have both the ultrasound and elastography time series converted to suitable formats. These volumes are not really volumes but rather time series. They are only loaded into VolumeShop as volumes but are treated by my plugins as time-series.

Real-time Acquisition of Ultrasound Slices Real time acquisition of slices is possible with the Logiq 9 device. The Logiq 9 device has a VGA output which mirrors its screen and can be connected to a frame grabber which itself is connected to the PC via USB. The components described later in Section 3.4 is fully capable of utilizing real-time ultrasound slices.

The only thing missing is a plugin for VolumeShop which interfaces with the frame grabber and updates its slice. Such a plugin would essentially act as the slice provider component described in as the slice provider and work with the rest of the framework without any further modifications. Due to the lack of a frame grabber this feature was not implemented.

For the implementation of the acquisition VolumeShop has existing importers for the `dat` format and image series as needed. See Section 3.4.4 for details on the visualization of the data.

3.3 Immersive Visualization

Utilizing the opentracker adapter described in Section 3.2.1 we can define the viewing frustum and the model matrix to enable the perception of

1. looking through a window in that the perspective projection changes as the user moves, and

2. interacting with a real object giving the data 6 degrees of freedom, i.e. make the object follow the user's motions.

The main goal of immersive visualization is that the user can use her skills of spatial orientation and visual processing to easier gain a better understanding of the data. By the end of this section, it becomes apparent that the immersive visualization can be realized by implementing two components which is together maintaining three matrices; a projection matrix P, a camera matrix C and a model matrix V as illustrated in Figure 3.12.

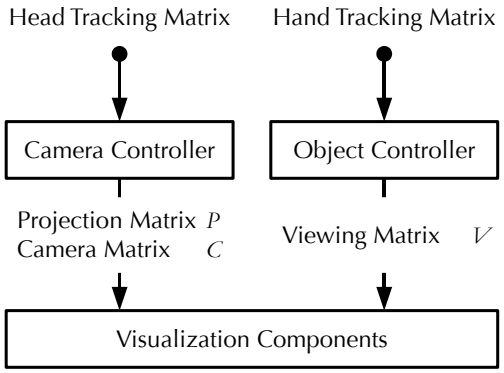


Figure 3.12: *The components and matrices involved with the immersive visualization*

3.3.1 Interaction

For interaction, this means that the user should be able to handle objects intuitively, just like in real life. For example, selecting an object and translating it to match the user's gestures should be a possible interaction mode.

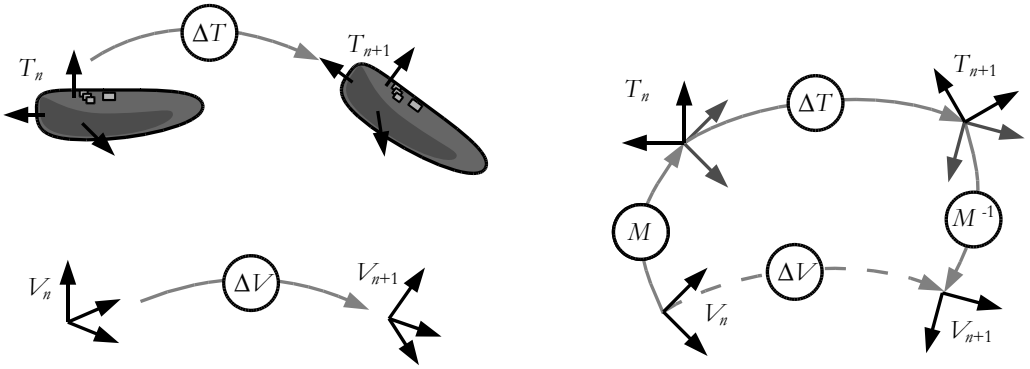
Model Matrix

The model matrix M determines how the object is viewed. It dictates the position and orientation of the object. Traditionally, computer graphics pipelines have two main matrix stacks. The first is the projection matrix stack which dictates the projection, the second is the model-view matrix stack which is the camera matrix and the model matrix combined.

To make the interaction work, the M needs to be updated with the transformation mirroring the change in the tracker transformations. The translation and rotation of the consecutive tracking matrices needs to be preserved and applied to M independent of M's value as illustrated in Figure 3.13(a).

Unless otherwise noted, all variables described in this section are matrices. The tracking input is a live feed of motion tracking matrices from the opentracker plugin. Assume T_n and T_{n+1} are any two consecutive motion tracking matrices from the opentracker adapter originated from the interactor, ΔT which transforms T_n to T_{n+1} and M_n is the model matrix corresponding to T_n.

To preserve rotation and translation of ΔT, M_n is simply translated to T_n's position. Then ΔT is applied and the result is finally translated back by the inverse of the first translation as illustrated in Figure 3.13(b). ΔM will not be explicitly computed, but it is the concatenation of the above transformations.



(a) Preservation of translation and rotation independent of reference frames

(b) Computation of the updated model transformation

Figure 3.13: *The relationship between the tracking and model transformations*

As noted above, ΔT transforms T_n to T_{n+1}. Similarly, there exists a matrix ΔM which

transforms M_n into the desired M_{n+1}. Thus, we have:

$$T_{n+1} = \Delta T \cdot T_n \iff \Delta T = T_{n+1} \cdot T_n^{-1} \tag{3.2}$$

$$V_{n+1} = \Delta V \cdot V_n \tag{3.3}$$

To define the transformations a function is needed which extracts the translation part of any matrix into a translation matrix. It is defined:

$$\mathrm{Tr}\left(\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & l \\ 0 & 0 & 0 & p \end{bmatrix} \tag{3.4}$$

First U is computed as the transformation translating M_n to T_n and the translated model matrix M'_n:

$$U = \mathrm{Tr}(T_n) \cdot \mathrm{Tr}(M_n)^{-1} \tag{3.5}$$

$$M'_n = U \cdot M_n \tag{3.6}$$

Applying ΔT from Equation 3.2, M'_{n+1} is obtained which is basically M'_n moved from T_n's reference frame to T_{n+1}' reference frame. Finally, the initial translation is undone and M_{n+1} is obtained:

$$M'_{n+1} = \Delta T \cdot M'_n \tag{3.7}$$

$$M_{n+1} = U^{-1} \cdot M'_{n+1} \tag{3.8}$$

Using the previously defined equations, it is possible to define a function f which computes V_{n+1} based on V_n, T_n and T_{n+1} in the following way by Equation 3.8:

$$\begin{aligned} = M_{n+1} &= U^{-1} \cdot M'_{n+1} \\ \text{(substitution by Eq. 3.7)} &= U^{-1} \cdot \Delta T \cdot M'_n \\ \text{(substitution by Eq. 3.6)} &= U^{-1} \cdot \Delta T \cdot U \cdot M_n \\ \text{(substitution by Eq. 3.2)} &= U^{-1} \cdot T_{n+1} \cdot T_n^{-1} \cdot U \cdot M_n \\ \text{(substitution by Eq. 3.5)} &= \mathrm{Tr}(M_n) \cdot \mathrm{Tr}(T_n)^{-1} \cdot T_{n+1} \cdot T_n^{-1} \\ &\quad \cdot \mathrm{Tr}(T_n) \cdot \mathrm{Tr}(M_n)^{-1} \cdot M_n \qquad (3.9) \\ &= f(M_n, T_n, T_{n+1}) \qquad (3.10) \end{aligned}$$

As observed in Equation 3.9, f is defined entirely in terms of M_n, T_n and T_{n+1}. The function

f also preserves translation and rotation as specified earlier in this section. This concludes the discussion on the interaction. For details on the implementation of this component, see Section 4.2.5.

3.3.2 "Looking Through a Window"

As stated in Section 3.1.1, for the immersive display to be convincing, the display must act the role of a window. This can be compared to the normal use of a display in which it acts the role of an interactive image.

Since the immersive visualization uses a stereo display. The visualization needs to be rendered twice for each frame, that is; once for each eye. The only difference between the frames are the perspective projection matrices.

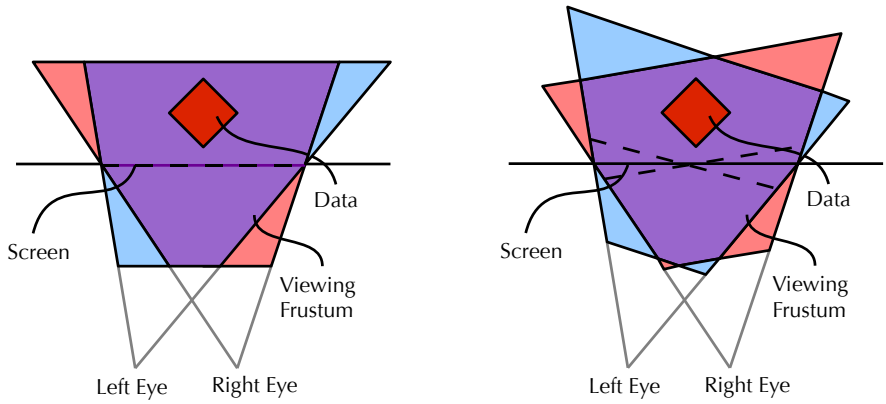
One could easily assume that the difference is the model matrix specified by the position of the eye and a focus point as illustrated in Figure 3.14(a).. The shape of the viewing frustum would then be the same for both eyes while the position differ. The problem with this approach is that the plane of focus is not parallel for the two eyes as illustrated by the dotted lines in Figure 3.14.

To remedy this problem, the plane of focus needs to be aligned with the display. The solution is to shape the viewing frustum to fit the viewpoint and the display [11] instead of translating and rotating it as illustrated in Figure 3.14(b). The viewing frustum is shaped so that:

1. The viewing direction is orthogonal to the picture frame. This ensures that the rendering plane is parallel to the picture plane. More importantly, it ensures that the right and left rendering planes are parallel.

2. The shape of the viewing frustum matches the shape of the pyramid defined by the picture frame and the eye position.

For the stereoscopic visualization, it is required to not only take into account the position of the head, but the individual eyes as well. These positions are acquired from the OpenTracker adapter through the tracking compensation component described in Section 3.2.1. The probe offset in the tracking compensation component is configured to place tracking point approximately at the nose ridge of the user. This point is used as the viewpoint is used when using a monographic display. By placing the point here we may use a fixed offset right and left to obtain the approximate positions of the eyes. This offset is basically half of the distance between the user's eyes. These points are suitable for a basis when computing the stereo projection matrices. Given the motion tracker matrix T and the eye distance d, then



(a) asymmetric viewing frustums (b) symmetric viewing frustums

Figure 3.14: *The right and left viewing frustums.*

the viewpoints for the head p_h, the left eye p_l and the right eye p_r are:

$$p_h = T \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad p_r = T \begin{bmatrix} d/2 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad p_l = T \begin{bmatrix} -d/2 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

With the position of the eyes in hand it is possible to proceed to define the projection matrix for each eye separately. The position of the eye in question will be denoted the *viewpoint* and is denoted p. The frame of reference will be the center of the picture where x points right, y up and z towards the user. The frame of reference is then completely defined by its width W and height H.

Taking the formula for the viewing frustum from the OpenGL reference manual [24]:

$$\begin{bmatrix} \frac{2z_n}{x_r - x_l} & 0 & A & 0 \\ 0 & \frac{2z_n}{y_t - y_b} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \text{where} \quad \begin{aligned} A &= \tfrac{x_r + x_l}{x_r - x_l} \\ B &= \tfrac{y_t + y_b}{y_t - y_b} \\ C &= -\tfrac{z_f + z_n}{z_f - z_n} \\ D &= -\tfrac{2z_f\ z_n}{z_f - z_n} \end{aligned} \tag{3.11}$$

where x_r and x_l are the x-coordinate of the right and left sides, y_t and y_b are the y-coordinate of the top and bottom sides and z_n and z_f is the z-coordinate of the near and far plane. Note that x_r, x_l, y_t and y_b are the coordinates at the near plane; $z = z_n$. Using the definition above for the frame of reference, W and H and the viewpoint $v = (x_v, y_v, z_v)$, we can compute

the sides of the view frustum as follows:

$$\text{right side:} \quad x_r = W/2 - x_v \tag{3.12a}$$

$$\text{left side:} \quad x_l = -W/2 - x_v \tag{3.12b}$$

$$\text{top:} \quad y_t = H/2 - y_v \tag{3.12c}$$

$$\text{bottom:} \quad y_b = -H/2 - y_v \tag{3.12d}$$

To be able to place objects in the space in front of the display we need to move z_n accordingly so the objects are not clipped as z_n defines the near clipping plane.

Additionally, whenever z_n changes, the coordinates for the sides need to scale proportionally relative to the viewpoint. Given a side u_s, the new z'_n and old z_n, and the appropriate viewpoint coordinate u_v where $v \in \{x, y, z\}$, the new u'_s is computed as:

$$u'_s = (u_s - u_v)\frac{z'_n - z_v}{z_n - z_v} + u_v \tag{3.13}$$

The only element not defined yet is the far plane z_f. Since it is not used to determine the shape of the frustum, it can potentially assume any value. Generally, the far plane should be farther away than any object rendered to ensure that no objects are clipped.

Equipped with these formulas, it is possible to define a viewing frustum which adheres to the points outlined above. As noted above, the viewpoint v is provided by the OpenTracker adapter and the tracking compensation component. The size of the display is configured by the user and the near and far plane are set to appropriate constants.

The formula computing the perspective transformation matrix (Equation 3.11) places the camera at the origin which is fine for monoscopic displays. For stereo displays it is important that the position and orientation of the screen in the right and left viewing frustums are the same.

The solution is to translate both viewing frustums so that the screens are positioned at the origin instead of the camera. The coordinates of the center of the screen are basically the translation part of the projection matrix P. Applying the inverse of the translation part of P to the model transformation M will yield the corrected model matrix M' which has the screen positioned at the origin. M' is computed by applying a corrective translation matrix C as follows:

$$C = \mathrm{Tr}(P)^{-1} \tag{3.14}$$

$$M' = C \cdot M \tag{3.15}$$

Finally, to avoid that the size of the rendered object on screen depends on the screen's

physical size, all sizes are normalized by the screen height, that is; divided by H. This has the advantage that the rendered objects scale with the screen height and avoids the problem of rendering only a tiny area of the big screen. This problem can of course be fixed by scaling the model matrix, but it will require rescaling at every computer with different screen sizes which our approach removes the need for doing.

To sum up, the component needs to supply two matrices; the projection matrix M' and a corrective translation matrix C which needs to be applied to the model matrix stack. See Section 4.2.6 for details on the implementation of this component.

3.4 Multimodal Visualization

The second scenario addressed by this thesis is the multimodal visualization of 2D ultrasound combined with 3D anatomical scans. The acquisition of the required data has already been described in Section 3.2. In short, the Flock of Birds are used in conjunction with the OpenTracker adapter for tracking, whereas the medical data are acquired by ultrasound scanners and MRI scanners.

As described in Section 3.1.2, there are four main obstacles to overcome and these will be addressed in this order in the next sections:

1. The acquisition of the required data; tracking, ultrasound and MRI.
2. The time synchronization of the tracking and the ultrasound time series.
3. The specification and computation of the registration transformation matrix.
4. The multimodal visualization of the combined data.

The components described in this section are a component for playing back tracking transformation from log files, a component for displaying a slice image, a component which extracts a slice image from a 3D volume, a component for picking points on the slice image and a component for computing the registration transformation.

3.4.1 Time Synchronization

The ultrasound slice images and motion tracking data are acquired from two different sources. This means that the data are not synchronized as we did not realize any mechanism to perform this across the devices – mainly due to lack of synchronization facilities in the acquisition devices we used. To make the synchronization process easier, it was attempted to synchronize the acquisition devices as much as possible during the sessions. In practice, this means agreeing on the time of start and stop of the acquisition. This is by no means a

perfect solution but it makes it easier to perform the synchronization when it is known the modalities are only offset by at most a couple of seconds.

If the acquisition is performed in real-time during the visualization, the need for synchronization is removed, but due to latencies in the system, a delay mechanism needs to be implemented instead. Since real-time acquisition and visualization was not performed due to lack of appropriate hardware, the rest of this section will describe an off-line visualization which is realized by playing back a log file containing the tracking data.

The tracking transformations read from the log file are named and time stamped. This time stamp is defined as the time since VolumeShop started in seconds, with millisecond resolution. The time series of slices are also time stamped as the DICOM header includes the time of acquisition, but this time stamp is lost when converting the data to the `dat` format. The DICOM header also includes the total session time and the number of frames captured which can be used to compute the frame rate of the slices.

The manual synchronization effort described above ensures that the first image slice is approximately acquired at the time the first tracker transformation were time stamped. Combining these facts, it is possible to synchronize the tracking transformations with the slice time series.

The three components responsible for rendering the image slices in the correct positions are the slice renderer, the slice provider and the tracking transformation playback components. The time synchronization is implemented in slice provider and the tracking transformation playback components. Basically, the slice provider component supplies a slice of the ultrasound time series volume from a specified time.

As noted above, the frame rate of the slice images are known and the frame rate value of the slice provider component is set to match it. This leaves only the time offset to be specified.

To find the time offset a motion has to be found which can be recognized in both the slice images and the transformation. The best motions are abrupt change of direction or speed. Both these kind of change are easy to spot under visual inspection of the tracker data but are not always as easy to spot on the ultrasound slice.

When such a motion is found, the time offset can be set to match these motions together. The way this is done is to move the time slider so that the ultrasound slice shows the slice at initiation of the motion. Now move the time offset slider so that it also shows the transformation at the initiation of the motion. Note that moving the time offset slider results in changing the time for the transformation but not the slice. So the slice will move around, but the ultrasound image will be still which makes it possible to align the motion of the two modalities.

Assuming the framerate is correct, then the modalities should be properly synchronized.

The implementation of the components discussed in this section are found in the following sections; the tracking transformation playback are described in Section 4.2.2, the slice provider is described in Section 4.2.4 and finally the slice renderer in Section 4.2.7.

3.4.2 Registration

With the synchronization complete, it is possible to register the data sets. As noted in Chapter 1, registration is the process of aligning different data sets, generally with different reference frames, so that they share reference frame. In this thesis, a semi-automatic rigid registration method has been implemented.

The basic premise of this method is to locate three landmarks which is recognizable in both data sets. All three features are then marked in both data sets by specifying their location on the display. These points are specified in world space. Note that we operate with two world spaces prior to a successful registration; one for each data set which equates to their respective frames of reference. Using these points, we compute a matrix R aligning the two world coordinate systems.

Picking Reference Points

The user interface for the reference point picking is designed to be simple to use. It is configured to map a key on the keyboard to the picking mode, usually the keys `1`, `2` and `3` are mapped to the properties "Point 1", "Point 2" and "Point 3" respectively. The user then presses and holds the key to activate the point picking for the associated point which is active until the user releases it again. Clicking the left mouse button while in picking mode sets the point. When the mouse is clicked, the point associated to the key will be updated with the result of the intersection computation. All the points are made available in both world space and data space. The data space representations are postfixed with "(data space)" to differentiate them from the world space variants.

To pick reference points, the projection matrix P, the model matrix M, and the data matrix D and the normal of the plane n_D specified in data space needs to be taken into account. As noted in Section 3.1.2, the line defined by the mouse cursor (Figure 3.15) and the plane defined by the slice is transformed into world space for intersection testing.

To simplify things, P, M and D are converted into two matrices which performs the desired transformations, namely converting the input parameters to world space:

- M_{ws} which transforms world coordinates to screen coordinates.
- M_{dw} which transforms data coordinates to world coordinates.

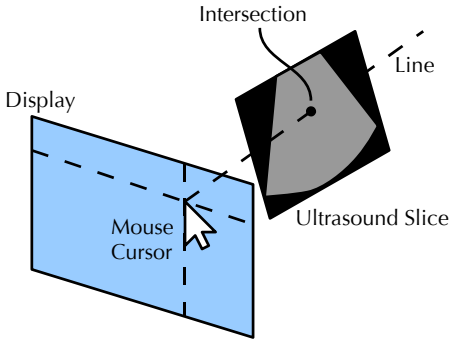
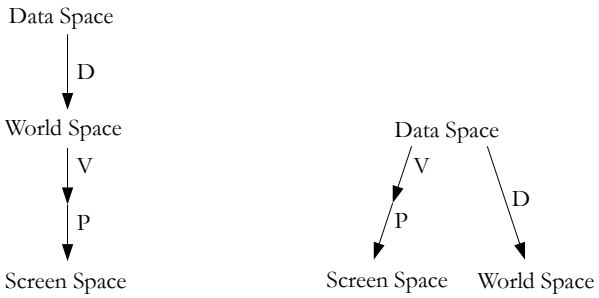


Figure 3.15: *Picking reference points on the ultrasound slice*

This simplification turns out to be helpful for another feature of this component. Due to the differences of two slice views in the user interface, this component needs to have two operation modes. Both works as already described picking points at the plane defined by the slice. The difference between the modes is how the relations between the world space and camera space is defined. The first mode (mode A) assumes that the world space relates to the screen space through the $P \cdot V$ transformation as illustrated in Figure 3.16(a). The second mode (mode B) on the other hand, assumes that the world space is not related to the screen space at all and that the plane is initially defined in world space illustrated in Figure 3.16(a) and is thus parallel to the xy plane. D^{-1} is still used to convert the intersection point to data space.



(a) Plane displayed in data space (b) Plane displayed in screen space

Figure 3.16: *Two modes of operation.*

Depending on the mode of operation , M_{ws} and M_{dw} will be defined differently. In mode A where the plane is rendered in data space is the traditional mode where the full transformation stack is applied to the plane. In this case the matrices can be computed straight forward, like this:

$$M_{ws} = V \cdot P, \quad M_{dw} = D \tag{3.16}$$

In mode B, where the plane is rendered in world space some more thought is required. D is not applied to the plane when it is being rendered so one can say that the world space *is* the data space and the true world space is something entirely different defined by D. By this reasoning the matrices is computed as follows.

$$M_{ws} = P \cdot V \cdot M^{-1}, \quad M_{dw} = D \tag{3.17}$$

Once M_{ws} and M_{dw} are defined, it is possible to carry on with the rest of the computations and can the mode of operation can be disregarded.

The input is the point $p = [x, y]$ which is the position of the mouse cursor is screen space. We then create two points in 3D space which forms the basis of the line through p perpendicular to the screen. These points are defined by $p_0 = [x, y, 0, 1]^T$ and $p_1 = [x, y, 1, 1]^T$. These points are then transformed into world space where the rest of the computation will take place. Their world space representations are defined as $w_0 = M_{ws}^{-1} \cdot p_0$ and $w_1 = M_{ws}^{-1} \cdot p_1$. The direction of the line is defined $d = w_1 - w_0$ and the line itself:

$$l(t) = w_0 + t \cdot d \tag{3.18}$$

The plane is defined by the normal $n = [x_n, y_n, z_n, 0]^T$ and the point $v = [0, 0, 0, 1]^T$ in the data space. These needs to be converted to world space like the points p_0 and p_1. The plane in world space is defined by the normal $n_w = M_{dw} \cdot n$ and the point $v_w = M_{dw} \cdot v$. The plane is then defined by:

$$n_w \cdot (x - w_0) = 0 \tag{3.19}$$

With both the point and plane defined in world space the intersection computation may be carried out. Assuming the plane is not parallel to the line we know that there is a single point of intersection. Substituting x with $l(t)$ and again $l(t)$ with its definition by Equation 3.18 we get an equation with t as the only unknown:

$$t = \frac{n_w \cdot (v_w - w_0)}{n_w \cdot d} \tag{3.20}$$

Plugging t back into $l(t)$ gives us the point of intersection. The resulting point is stored in one of the point properties of the plugin as indicated by the user.

The description of the implementation of this component can be found in Section 4.2.9.

3.4.3 Computing the Registration Matrix

By marking the required number of landmarks, two sets of points (a_0, a_1, a_2) and (b_0, b_1, b_2) is acquired for the image slices A and the medical volume B. Once these points have been

identified, a transformation matrix R is computed which transforms a_i into b_i. This transformation is then used to transform points from A's to Bs reference frame.

For affine transformations in three dimensions, we really need four points to unambiguously define the transformation. The reason for this is that the transformations are specified by 4×4 matrices and basically one point is needed to for each column to unambiguously define. Thus 4 points is needed.

Assuming that A and B are orthogonal – that is; all three axes are orthogonal to each other, not counting the translation part – and uniformly scaled, the fourth point can be synthesized. This assumption is not unreasonable as A and B is the reference frames of the ultrasound images and the MRI volume respectively and both are real world reference frames and disregarding relativistic space-time bending of space, they can be assumed to be orthogonal and uniformly scaled.

Specifying three points automatically defines the translation part and two of the axes of a matrix. The assumption of orthogonality fixes the direction of the third vector and similarly the assumption of uniformly scaled vectors fixes its length. Thus the entire matrix is unambiguously defined by only three points.

Assuming the three points (p_0, p_1, p_2) specified by the user we are able to construct a matrix that is a reference frame for the points. We assume p_0 to be the origin of the reference frame. Then two of the axes are defined by:

$$v_1 = p_1 - p_0 \quad \text{and} \quad v_2 = p_2 - p_0 \tag{3.21}$$

The last axis is computed from these as:

$$v_0 = \frac{v_1 \times v_2}{|v_1 \times v_2|}|v_1| \tag{3.22}$$

By crossing v_1 with v_2 we obtain a vector perpendicular to both v_1 and v_2. This vector is then normalized and scaled to the length of v_1 which ensures that the new vector's length is proportional to v_1. It is important that the constructed vector is proportional to the other two vectors, otherwise the resulting registration transformation would scale non-uniformly and would contradict our assumption of the reference frames being uniformly scaled. The choice of v_1 is arbitrary, it can be any linear combination of v_1 and v_2. Having computed the reference origin p_0 and three vectors (v_0, v_1, v_2) we can proceed to construct the matrix defining the reference frame M as follows:

$$M = [v_0\ v_1\ v_2\ p_0]$$

With the described method, we can construct a reference frame for the points specified

in A and B separately. The resulting matrices; M_A and M_B like the points they are based on are specified in world coordinates of the different data sets. Defining the matrix R as the transformation mapping M_A to M_B as:

$$M_B = R \cdot M_A \tag{3.23}$$

$$\text{which yields} \quad R = M_B \cdot M_A^{-1} \tag{3.24}$$

We find that R not only maps M_A to M_B but also maps A's reference frame to B's reference frame because M_A and M_B are defined in those reference frames respectively. Plugging this matrix into the matrix stack in Figure 3.8(c) as the matrix R we obtain correct registration so long as the points are properly defined by the user.

As described in Section 3.4.2, the point specifier uses the world space as a reference frame when specifying points. The result is that it is not necessary to pick all three points in the same slice easing the process of finding suitable features since the odds of finding a single feature in a slice is much greater than finding three.

This computation is carried out by the registration component and is described in detail in Section 4.2.10.

3.4.4 Multimodal Visualization

The final stage in this pipeline is the multimodal visualization of the combined modalities. As already noted, the user interface consists of three views. One presenting the ultrasound slice, a second presenting a slice of the volume and the last displays the combined modalities. To realize these views, visualization components from VolumeShop and custom made components were utilized.

VolumeShop comes with a set of standard components for visualization of 3D volumes in different ways. For this pipeline, two custom visualization components were made; a slice renderer and a matrix renderer. The two built-in renderers used in this pipeline are the volume slice renderer – called the *volume slice renderer* to differentiate it from the custom *slice renderer* – and the volume renderer.

The reason a custom slice renderer were created was that the built-in volume slice renderer applies a transformation to the slice based on the z-coordinate to place it inside the volume in the correct location. That translation is incompatible with the goals of this thesis and since the slice needs to be placed according to the tracking information instead. Additionally, to support real-time images from an ultrasound scanner, the slice renderer needs to be able to accept a single image instead of a whole volume.

The three views each utilizes the following components:

Ultrasound Slice View shows the untransformed ultrasound slice and uses the custom slice

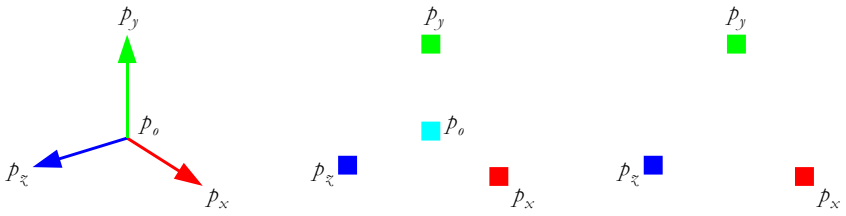
renderer to render the slice and the matrix renderer to render the picked points.

Volume Slice View shows the volume slice and uses the built-in slice renderer to render the slice and renders the picked points in the same manner as the ultrasound slice view.

Combined View shows the combined modalities and uses the built in volume renderer to render the reference volume and the custom slice renderer to render the slice.

Matrix Renderer

The matrix renderer was initially created to visualize matrices for debugging purposes. It evolved to be able to visualize matrices using different representations so it could be used in more scenarios, these representations are illustrated in Figure 3.17. The matrix renderer renders the origin of a matrix and its axes relative to the origin.



(a) Vector mode (b) Point mode (c) Point mode without the origin

Figure 3.17: *The different display modes for the matrix renderer*

Given the matrix $M = [x\ y\ z\ w]$, the reference points of the representation are:

$$p_0 = w \tag{3.25a}$$

$$p_v = w + v \quad \text{where } v \in \{x, y, z\} \tag{3.25b}$$

Where p_0 is the origin and p_x, p_y and p_z is the position of the axes.

The different representations have the coloring in common. The component always renders p_x as red, p_y as green and p_z as blue. In the single case where the origin p_0 is rendered separately, it is drawn using cyan.

This exact coloring was chosen because it is intuitive to map the vectors to colors, as colors are really vectors in color space. Thus the vector $[x, y, z]$ is mapped to the color $[r, g, b]$. The cyan color were chosen with two criteria in mind. It must not be the same color as the other three colors and neither be black or white, at the same time being bright and easily discernible.

Arrows which is three arrows pointing from p_0 to each of the three points p_x, p_y and p_z.

Points which the four point is rendered as a point each. The points are made 2 pixels in diameter to make them easier to spot.

Points (no origin) which is the same as the previous, but omitting v_0. This style is used to draw the three points picked by the user.

Slice Renderer

As noted above, the built-in slice renderer was not adequate for the needs of the multimodal visualization pipeline so a custom slice renderer was created. The slice renderer used for the multimodal visualization had three requirements.

- To render the slice according to the projection transformation, the model transformation and the slice transformation. In particular, it should not apply a translation based on the z-position of the slice in the volume.
- To render the slice with the original colors and intensities or by applying a transfer function.
- To accept a single image for rendering as opposed to the built-in slice volume renderer which accepts a volume.

It is fairly straightforward to define this component. Basically, it applies the projection matrix, the model matrix and the slice matrix to the matrix stacks in the OpenGL pipeline and renders the slice as a textured quadrilateral with the following vertices:

$$v_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad v_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.26}$$

Matching texture coordinates will ensure that the slice texture is stretched across the whole quadrilateral. The size of the quadrilateral is defined by the length of the x and y axes of the slice matrix, see Figure 3.18, which in contrast to the model matrix do not have to be uniformly scaled.

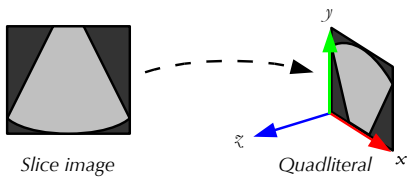


Figure 3.18: *The ultrasound slice placed in the slice matrix*

The implementation of the slice renderer is described in greater detail in Section 4.2.7 and the matrix renderer component in Section 4.2.8.

With this the description of the pipelines is concluded. The next chapter will describe the implementation details of these pipelines and after that Chapter 6 will discuss the results.

4
Implementation

The implementation of this thesis consists of a set of components which works together to produce immersive visualizations on large stereo screen and multimodal visualization of 2D ultrasound with 3D modality. The components in the framework for the pipelines have been described on a conceptual level in Chapter 3 and now the implementation aspects will be presented.

The pipelines consists of a shared OpenTracker adapter which is configurable for the different setups.

The immersive visualization pipeline is made up of a camera controller component and an interaction controller component, both of which utilize the OpenTracker adapter for tracking input. These plugins defines the projection transformation, the camera transformation and the model transformation matrices. Any visualization can be utilized with these plugins, all that is needed is for the renderers to make use of the supplied matrices as illustrated in Figure 4.1.

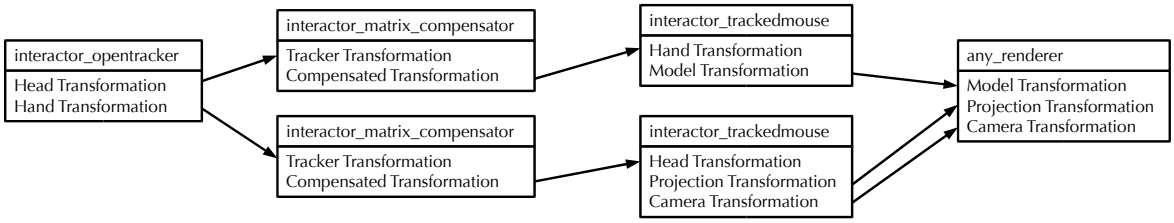


Figure 4.1: *The immersive visualization pipeline and its components*

The multimodal visualization pipeline illustrated in Section 4.2 utilizes the OpenTracker adapter for the tracking of the ultrasound probes. The tracking data is stored to a log file and read by the transformation playback component for off-line processing. The visualization of the slice is performed by a custom slice renderer which gets the slice images from a slice provider and the tracking data from the transformation playback plugin. The visualization

of the 3D modality is performed by the built-in volume renderer and volume slice renderer. The landmark picking is performed by a point picker component and the registration is performed by a registration component which computes the registration matrix based on the picked points.

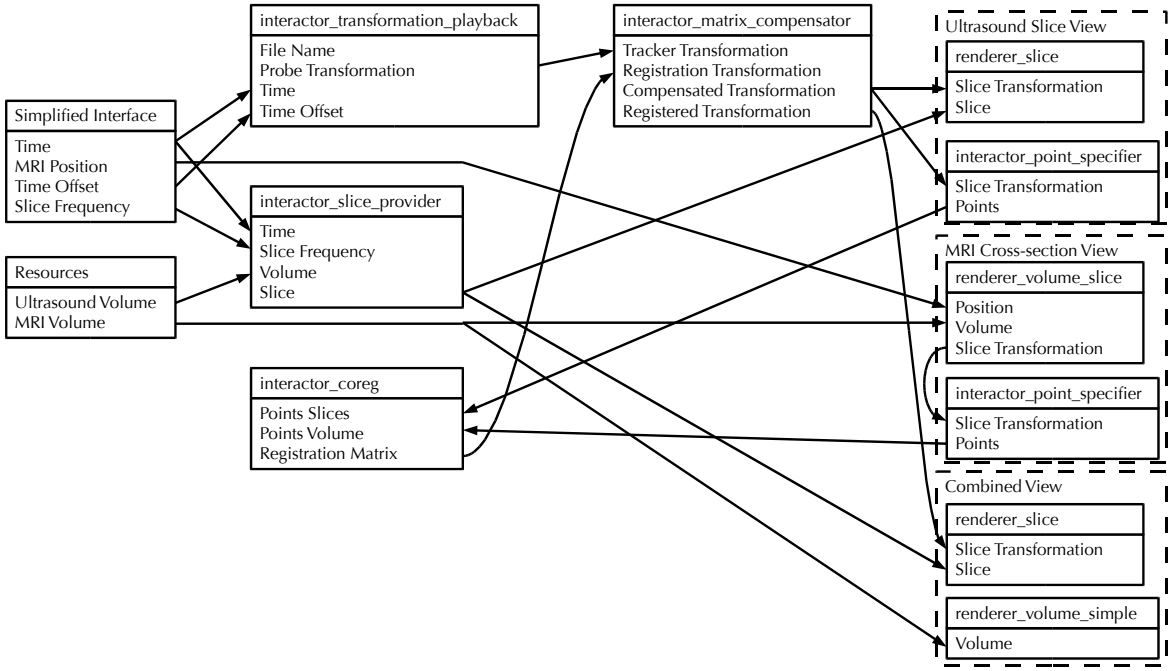


Figure 4.2: *The multimodal visualization pipeline and its components*

4.1 The VolumeShop Framework

The implementation has been realized as a set of plugins for a volume visualization framework called VolumeShop. The VolumeShop framework is designed to enable rapid prototyping of visualization applications. The most important features are features are:

Plugin Architecture Almost all the functionality of VolumeShop is defined by plugins with a core component responsible for communication between the plugins. VolumeShop itself serves as little more than a scaffold for the plugins to be placed in. The plugins come in several different categories including data importers, rendering compositors and the ones used most in this thesis; interactors which are responsible for user interaction and renderers which are dedicated to the display.

Properties Every plugin and a few built-in types have a set of properties. Properties are basically named variants which can be added or removed at run-time. Variants have

dynamic type in that they may hold any type of values as long as that particular type is supported. Variants can also be typed so that it may only hold data of a single type. Common value types are integers, floating point numbers, vectors, matrices, images and volumes.

Variant Observing and Linking Variants may be observed by one or more observers. By adding an observer to a variant, the observer will receive notifications when the value of the variant changes. Observers are light-weight objects which forwards the notification to the desired destination. Observers are used when implementing a plugin to get notifications when the input properties change.

Linking is defined at run-time and basically uses observing to enable the user to link the values of two properties. When two properties are linked, their values will always be the same. Change the value of any of them and the other will change to match the first. It is also possible to make a chain of links to ensure that any number of properties have the same value. This technique is used for communication between plugins.

An example of linking is the link from the model matrix of an interactor to the model matrix of the renderer. The result is that the interactor doesn't need to know about the renderer. It only hosts its own model matrix which anyone can observe when desirable.

The plugins are compiled against VolumeShop's core library and are stored in a folder containing all the executable files. These plugins are then loaded at runtime into VolumeShop by the user and are linked to create a configuration which produces the desired results. The configuration of plugins and the data links can at any time be stored as a project as an `xml` file. These projects stores all plugins, the links between the plugins and all references to external files.

VolumeShop also has an export/import feature which basically creates a bundle containing the `xml` project file and all files referenced by the plugins. This feature allows projects to be safely moved between folders and even computers.

The result, illustrated in Figure 4.3, is a dynamic and loosely bound system which enables rapid prototyping by letting the plugins be developed in isolation without having to take considerations to other plugins.

4.2 Plugins

The components described in Chapter 3 are implemented as individual plugins for the VolumeShop framework. The following sections will describe the implementation of all the plugins made for this thesis. The properties of the plugins are listed in the following style:

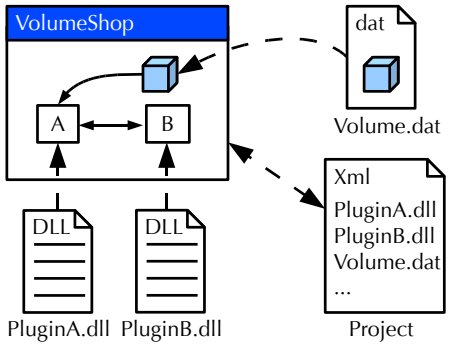


Figure 4.3: *The VolumeShop architecture*

[in,out] Property Name here is a short description of the property. The **in** and **out** denotes input and or output properties.

4.2.1 plugin_interactor_opentracker

This plugin's responsibility is to interface with the OpenTracker library to supply uncompensated real-time tracking information. As noted in Section 3.2.1 the process of compensation is basically to transform the raw tracking matrix into the appropriate frame of reference. The implementation of the OpenTracker adapter has the following properties:

[in] Configuration File is the path to the OpenTracker configuration file.

[in] Log File Name is the path to the log file.

[in] Is Logging is a boolean indicating whether or not it is logging the transformation to file.

[out] <name of transformation> is the uncompensated transformation matrix of the tracker. There is one property for each tracker specified in the OpenTracker configuration file.

The OpenTracker library is wrapped by the OTWorker class. This class supplies a simplified interface to the OpenTracker library. It takes care of the initialization, running and tear-down of the OpenTracker instance.

The name OTWorker stems from the fact that it also runs in its own worker thread. The reason for this is that the OpenTracker actively polls the inputs while waiting for data to arrive. Without a worker thread, this activity would reduce the performance of the host application which in addition would need to do the polling itself. The thread also converts the incoming data from position+quaternion to matrix representation which is the representation used throughout our pipelines.

OTWorker is hosted in the plugin_interactor_opentracker plugin. Its responsibility is to supply a matrix property for each tracker which is updated real-time. The actual update is performed in the `idle()` method of the plugin. This is a method which is called by the system continuously while it is idle. The `idle()` method is used for the polling of the OTWorker is that it will not impact the performance of the application as the method will only be executed while the system is inactive. This ensures that the polling of the motion tracking will not contest with other parts of the application for processing resources.

There is also an option to record the motion tracking data to a log file. This data needs to be stored to file to enable replay of the transformation later as part of the registration process of the multimodal visualization pipeline. All that is needed to log is for the user to supply the path of the log file. The logging can be started and stopped as appropriate. In our case we attempted to synchronize the start of the ultrasound acquisition with the start of the logging of the tracking transformations to reduce the manual effort needed to synchronize the two data streams.

4.2.2 plugin_interactor_transform_playback

The plugin_interactor_transform_playback plugin is playing back a tracking session previously logged by the plugin_interactor_opentracker plugin. The interface to these two plugins for other plugins is the same, and is basically only to link to its supplied matrix which makes them interchangeable. This plugin has the following properties:

[**in**] **File Name** is the path to the log file. The history will be read from this file whenever this path changes.

[**in**] **Time** is the point in time from when the transformation matrices is acquired.

[**in**] **Playback** is the current mode of operation. Its value can be “Playback” or “Paused”.

[**out**] **<name of transformation>** is the uncompensated transformation matrix of the tracker at the time specified **Time** property. There is one property for each tracker named in the log file. Thus, it supports any number of trackers.

This plugin can either play back the motion tracking data automatically or it can be controlled by the user with a slider controlling the **Time** property in the user interface. The latter is used when performing multimodal visualization since the ultrasound slices needs to keep still when the user is pinpointing landmarks in them. This is because a moving target is harder to hit than a still target.

Essentially, the number of tracking records can be different from the number of images in the ultrasound time series. This is because the frame rate of the ultrasound is different

from the update frequency of the tracking device. To cope with this, the transformation matrices, and ultrasound slices are addressed by time instead of directly indexed. Because of this, there is a need to map the time into indices and further, to transformation matrices.

Algorithm 1 is used for finding the corresponding transformation matrix for the requested time. the algorithm starts with finding the record just prior to or exactly at that time. This record is containing a (time stamp,transformation) pair and is denoted $R_n = (t_i, M_i)$ for $0 \leq i \leq N$ where N is the total number of records. The records are sorted so that $i < j \iff t_i < t_j$. If the requested time is outside the range $[t_1, t_N]$, where N is the number of records, it returns M_1 and M_N respectively.

Algorithm 1: *ComputeMatrix(t)*

Input: the records $R = (R_1, R_2, ..., R_N)$
Input: the requested time t
Result: the interpolated matrix at time t

```
/* Return endpoints when outside range                    */
if t ≤ t_1 then return M_1
if t ≥ t_N then return M_N
/* approximate and find the correct index                  */
i ← floor (t/(t_N − t_1))
if t_i = t then
    return M_i
else if t_i < t then
    while i < N ∧ t_{i+1} < t do
        i ← i + 1
    end
else
    while t_i > t do
        i ← i − 1
    end
end
/* Now linearly interpolate                               */
s ← (t − t_i)/(t_{i+1} − t_i)
return (s − 1)M_i + sM_{i+1}
```

It is assumed that the records are uniformly distributed over time so it is possible to compute the approximate index. From this approximation we start to search for the correct record.

When R_n is found, the algorithm linearly interpolates from M_n to M_{n+1} to get an approximate transformation at the exact time requested. It was opted not to use spherical linear interpolation (SLERP) as the trackers are updated about 30 times per second and the tracking matrices vary relatively little from one update to the next.

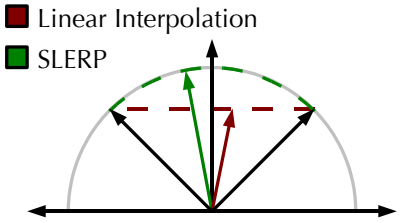


Figure 4.4: *SLERP vs. regular linear interpolation*

The problem, illustrated in Figure 4.4, that occurs with regular linear interpolation of matrices is that when the rotation component is large, the length of the axes are reduced. Since the matrices operate with only small rotations, it is assumed that the reduction in axis lengths is small enough to disregard. In practice, this was the case.

4.2.3 plugin_interactor_matrix_compensator

The tracking transformation matrices from OpenTracker needs to be compensated. That is; to be transformed into the appropriate frame of reference. Basically the tracking transformation specifies the position and orientation of the tracker in relation to the transmitter. What is needed is the position and orientation of the tracked object in relation to a target reference frame, e.g. the reference frame of the dislay. This plugin has the following properties:

[in] Tracker Transformation is the uncompensated transformation matrix of the tracker and is denoted M.

[in] Transmitter Reference Frame is the matrix specifying the position and orientation of the transmitter and is denoted T (See Section 3.2.1).

[in] Probe Transformation is the matrix specifying the position and orientation of the tracked point in relation to the tracker and is denoted C (See Section 3.2.1).

[out] Compensated Transformation is the compensated tracker transformation and is denoted M_C.

[in] Registration Transformation is the registration transformation matrix R.

[out] Registered Transformation is the compensated and registered tracker transformation and is denoted M_R.

Given the reference frame of the tracker M and the two user defined matrices, the reference frame of the tracked object M_C is:

$$M_C = T \cdot M \cdot C \tag{4.1}$$

T and C is able to compensate for every possible linear transformation to the data needed under any circumstances. Any additional pre or post multiplied transformations can be concatenated into T or C.

For example, if M_C needs to be further compensated by A and B like so:

$$M'_C = A \cdot M_C \cdot B \tag{4.2}$$

Then, by Equation 4.1:

$$\begin{aligned} M'_C &= A \cdot T \cdot M \cdot C \cdot B && (4.3)\\ &= (A \cdot T) \cdot M \cdot (C \cdot B) && (4.4)\\ &= T' \cdot M \cdot C' \quad \text{where } T' = A \cdot T \text{ and } C' = C \cdot B && (4.5) \end{aligned}$$

Thus, by assigning $T \leftarrow T'$ and $C \leftarrow C'$ it becomes apparent that any linear transformation needed to compensate M further can be included in T and C.

In practice, C either models the small offset from the tracker to the ultrasound probe or to the eyes, whereas T models the offset and axis permutation from the transmitter to the reference frame of the screen.

The registered transformation matrix M_R is M_C transformed by the registration matrix:

$$M_R = R \cdot M_C \tag{4.6}$$

4.2.4 plugin_interactor_slice_provider

For the multimodal visualization pipeline, a plugin which supplies the appropriate ultrasound slice is needed as described in Section 3.4.1. This plugin is used in conjunction with the slice renderer plugin. Its task is to supply an image slice of a volume and a slice transformation for placing the slice in the volume.

In practice this plugin is used to extract an ultrasound slice from a time series stored as a volume specified by time. In this scenario the slice transformation is not used as it is only used for visualization of regular volumes; that is, true 3-dimensional volumes and not time series.

This plugin has the following properties:

[**in**] **Volume** is the volume to extract slices from and is denoted V.

[**in**] **Time** is the time coordinate of the slice to extract from the volume. This value is converted to a z-coordinate in the volume from which a x-y slice is extracted.

[**in**] **Slice Frequency** is the number of slice images to display per second. This is used to compute the time interval between the slices which is used when computing the z-coordinate of the slice in the volume.

[**out**] **Slice** is the slice image for the position.

[**out**] **Slice Transformation** is the reference frame of the slice assuming the volume is a regular volume. This transformation makes it possible to use the slice renderer described in Section 3.4.4 to visualize slices of regular volumes.

Since the volume is a time series, the time coordinate needs to be specified in seconds to be able to synchronize it with other time dependent data. This time coordinate is converted to a z-coordinate which determines which slice to fetch from the volume. The slice frequency is thus user configurable and will in most cases, if not all, match the actual frame rate of the acquisition device.

This plugin also computes an affine transformation which describes the position of the slice in the volume. This is included to make it possible to use this plugin to provide slices of regular 3-dimensional volumes as well. This transformation is needed to place the slice correctly when it is rendered using plugin_renderer_slice which is described in this chapter.

4.2.5 plugin_interactor_trackedmouse

Described in Section 3.3.1 is a component which controls the position and orientation of the rendered objects based on the user's gestures. This plugin maintains a model matrix which mirrors the changes to the input matrix when the left mouse button is pressed. This plugin has the following properties:

[**in**] **Hand Tracker** is a matrix containing the reference frame of the hand tracker and is denoted T.

[**in**] **Screen Height** is the height of the screen in centimeters. This is used to normalize the coordinates. As described in Section 3.3.2 all coordinates are normalized by dividing them by the screen height to remove the dependence of the size of the screen.

[**in**] **Camera Transformation** is the camera transformation matrix supplied by the camera component which was described back in Section 3.3.2.

[**out**] **Model Transformation** is the model transformation maintained by this plugin. It is used to place the data in the scene. As pointed out in Section 3.3.1, this is really the object transformation.

[**out**] **Camera * Model Transformation** is the concatenation of the camera and model transformation. It is combined with the model matrix for plugins which do not support separate camera and model matrices. In OpenGL terms this matrix is called the model-view matrix.

This plugin observes all input parameters and recomputes the model matrix whenever a change occurs while the left mouse button is pressed. The model matrix is updated to reflect the changes to the tracker matrix according to the formula described in Section 3.3.1.

Since the update is not triggered until after the motion tracking matrix is changed, the previous tracker matrix T_{n-1} is not available during the update. Thus when an update is completed it stores T_n internally so that it is available at the next update as T_{n-1}.

For this plugin to work properly, the reference frame of the transmitter defined in the matrix compensator plugin needs to be axis aligned with the screen's reference frame. Otherwise the object does not follow the user's motions as the user is observing it but as the transmitter observes it.

4.2.6 plugin_interactor_trackercamera

This plugin is the implementation of the camera controlling component described in Section 3.3.2. Its responsibility is to compute projection matrices for monoscopic and stereoscopic displays based on the users position. This plugin has the following properties:

[**in**] **Head Tracker** is a matrix containing the reference frame of the head tracker and is denoted T.

[**in**] **Screen Width** is the width of the screen in centimeters and is used with the height to determine the aspect ratio of the screen. They are manually set by the user.

[**in**] **Screen Height** is the height of the screen in centimeters, denoted h.

[**in**] **Stereo.Separation** is the distance between the eyes of the user specified in centimeters. It is set manually and should ideally be adjusted for every user.

[**out**] **Projection Transformation** is a computed projection matrix suitable for use on monoscopic screens and is denoted P.

[**out**] **Camera Transformation** is the camera transformation which specifies the position of the camera and is denoted C.

This plugin has in addition properties for the right and left eye containing the projection and camera transformation. In total there are three pairs of projection and camera transformations. All the inputs are observed and triggers a re-computation of the outputs when a change is observed. The workhorse, which does the heavy lifting of this plugin, is the `computeProjectionMatrix` method and its algorithm is described in Algorithm 2. It implements the formulas described in Section 3.3.2 to provide the projection matrix according to the viewpoint which is given as a parameter.

Algorithm 2: *computeProjectionMatrix(w,h,p)*

Input: the height and width of the screen w and h
Input: the position of the viewpoint $p = [x_p, y_p, z_p]$
Result: a projection matrix matching the viewpoint and the screen size

`/* First normalize by the screen height. */`
$p_n \leftarrow p/h$
$w_n \leftarrow w/h$
$h_n \leftarrow 1$

`/* Set reasonable near and far clipping planes */`
$z_0 \leftarrow \max(z_{p_n} - 5, 0.1)$
$z_1 \leftarrow z_{p_n} + 5$

`/* Compute the rest of the clipping planes at` z_0. `*/`
$x_0 \leftarrow (-w_n/2 - x_{p_n}) \cdot z_0/z_{p_n}$
$x_1 \leftarrow (w_n/2 - x_{p_n}) \cdot z_0/z_{p_n}$
$y_0 \leftarrow (-h_n/2 - y_{p_n}) \cdot z_0/z_{p_n}$
$y_1 \leftarrow (h_n/2 - y_{p_n}) \cdot z_0/z_{p_n}$

`/* Finally, compute the matrix */`
$a \leftarrow (x_1 + x_0)/(x_1 - x_0)$
$b \leftarrow (y_1 + y_0)/(y_1 - y_0)$
$c \leftarrow -(z_1 + z_0)/(z_1 - z_0)$
$d \leftarrow -2 \cdot z_0 \cdot z_1/(z_1 - z_0)$
return $\begin{bmatrix} \frac{2 \cdot z_0}{x_1 - x_0} & 0 & a & 0 \\ 0 & \frac{2 \cdot z_0}{y_1 - y_0} & b & 0 \\ 0 & 0 & c & d \\ 0 & 0 & -1 & 0 \end{bmatrix}$

The three projection matrices are computed by plugging three different viewpoints into `computeProjectionMatrix`. As noted in Section 3.3.2, given the motion tracker matrix T and the eye distance d, then the three viewpoints are

$$p_h = T \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad p_r = T \begin{bmatrix} d \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad p_l = T \begin{bmatrix} -d \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.7}$$

Also noted in the same section are the camera matrix C_n which is simply a translation by p_n where $n \in \{h, r, l\}$, the normalized viewpoint coordinates. C_n is computed by:

$$C_n = \mathrm{Tr}(\texttt{computeProjectionMatrix}(p_n))^{-1} \quad \text{where } n \in \{h, l, r\} \tag{4.8}$$

This plugin computes viewpoint-dependent perspective matrices. There is one limitation which originates in the VolumeShop framework. To properly compute the projection matrix, the screen size (w, h) and orientation needs to be known, and is supplied by the user. If the view only covers the screen partially this plugin needs additional information. Specifically, the view's size (w_v, h_v) and position (x_v, y_v), illustrated in Figure 4.5, needs to be known.

Unfortunately, VolumeShop only supplies the size of the view and since the size and position changes dynamically it is unfeasible to let it be supplied by the user. Thus it is not currently possible to achieve correct perspective projection when the view is non-fullscreen.

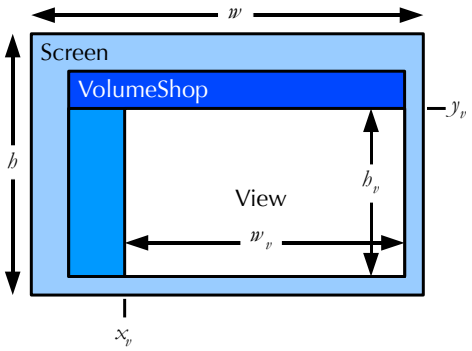


Figure 4.5: *The dimensions needed for computing a proper perspective projection matrix*

4.2.7 plugin_renderer_slice

This is the custom slice renderer described in Section 3.4.4. The reason the built-in slice renderer is not used is that it translates the slice according to the z position of the slice in the volume. Time series are not volumes but are still stored as volumes. Their time axis is mapped to the z-axis of the volume and thus, the slice is translated along the z-axis based on the its time coordinate which does not make any sense.

There was no way to disable this translation and therefore a custom slice renderer was made which takes the slice transformation as an input parameter. The properties of this plugin are:

[in] Projection Transformation is the projection matrix and is denoted P.

[in] Model Transformation is the model matrix and is denoted V.

[in] Slice Transformation is the transformation of the slice and is denoted S.

[in] Slice is the image to render onto the slice and is denoted I. In our case it is limited to plugin_interactor_slice_provider, but can easily come from a real-time source.

[in] Transfer Function is the transfer function to apply when rendering the slice. If this is not present, then the intensity or color of the slice will be used depending on the number of components. One component gives intensity only whereas three components gives full RGB color.

[in] Orientation is a matrix used to transform S, the reference frame of the slice. It is meant to only be used to change the orientation of the slice.

This plugin basically draws a textured quadrilateral which represents the slice with the slice image applied as a texture. The quad is drawn as an axis-aligned rectangle with corners at $[0, 0, 0]$, $[1, 0, 0]$, $[0, 1, 0]$ and $[1, 1, 0]$. Thus, the size of this quad is determined by the length of the x and y axes of S. This means that the configuration of the probe transformation matrix P from the matrix compensator plugin dictates the size of representation of the slice.

By linking the transformation computed by the slice provider, this plugin can render volume slices positioned in the volume as well. In the combined view, the slice transformation is linked to the compensated and registered output from the plugin_interactor_matrix_compensator. This allows the slice to be placed in the reference frame of the transmitter, or when the registration transformation is applied, in the reference frame of the 3D modality.

4.2.8 plugin_renderer_matrix

This plugin is the implementation of the matrix renderer component described in Section 3.4.4 and visualizes a frame of reference defined by a matrix M. It has three matrices as inputs, the first being M while the two others are the projection and model matrices. The two latter matrices is needed to transform M into screen space for rendering. This plugin has the following properties:

[in] Projection Transformation is the projection matrix and is denoted P.

[in] Camera Transformation is the camera positioning matrix and is denoted C.

[in] Model Transformation is the model matrix.

[in] Transformation is the transformation to visualize and is denoted D.

[in] Normalize is a boolean variable which when true, will cause the axes to be normalized, i.e. their length will be changed to 1.

[in] Scale is the scale to draw the axes in and is denoted s. It is only in effect when normalization is enabled.

[in] Representation is a choice of how the reference frame should be rendered. See below for description.

[in] Z Culling Enabled is a boolean that, if true, will enable the depth-buffer. It is sometimes desirable to disable the depth-buffer when the reference frame should not be obscured.

Basically, this plugin renders the translation of the matrix and each of the axes relative to it. The translation v is the fourth column of the matrix and the axes x, y and z are the first, second and third columns respectively. The point v is used as the base for the vectors. Thus the actual points which are rendered are v as the origin and $x + v$, $y + v$, $z + v$ and v as the axes.

When the normalization is enabled, the axes x, y and z will be assigned:

$$x \leftarrow \frac{x}{|x|} \cdot s, \quad y \leftarrow \frac{y}{|y|} \cdot s, \quad z \leftarrow \frac{z}{|z|} \cdot s \tag{4.9}$$

This plugin supports three rendering styles to cater different usages. The vector style renders a frame of reference where the axes are rendered as arrows pointing away from the origin. The point styles plots a point at each of the endpoints of the arrows. The rendering of the origin can be toggled on or off by choosing the appropriate rendering style. It is the latter of these two that are used to draw the points picked by user in the multimodal visualization pipeline. The depth buffer can be disabled for this plugin which is useful when the occlusion of the points is not wanted.

The last configuration option is the ability for this plugin to use or disregard the depth buffer. This is used to be able to see these points when embedded in volumes or obscured by slices.

4.2.9 plugin_interactor_point_specifier

As described in Section 3.4.2 there is the need for a component which can compute the coordinates of the point under the mouse cursor which intersects an arbitrary plane. The properties of this plugin are the following:

[in] Projection Transformation is the projection transformation matrix of the parent view and is denoted P.

[in] Model Transformation is the model transformation of the parent view and is denoted V.

[in] Data Transformation is the data transformation matrix of the slice and is denoted D.

[in] Plane Normal is the normal of the plane specified in D and is denoted $\vec{n}$

[in] Data Transformation Applied is a boolean indicating whether or not the slice is rendered in camera space or data space and is denoted b and is true if the latter is the case.

[out] Point n is the three resulting points specified in world space and is denoted p_n, where $0 \leq n \leq 2$.

In addition to these properties, there are three more points and two more matrices. The three points d_n are the same as p_n but specified in data space, so:

$$d_n = D \cdot p_n \quad \text{where } 0 \leq n \leq 2 \tag{4.10}$$

The two matrices are basically the points p_n and d_n packed into separate matrices M_p and M_d. These matrices are only used with the matrix renderer component to plot the points in the views.

This plugin also needs user input in the form of mouse cursor position, mouse button clicks and keyboard key-presses and key-releases. Three actions are set up, one for each point. When the key associated with a point is pressed, the plugin enters reception mode for that point. This plugin will accept and process mouse button clicks only when it is in this mode. Releasing the key will make this plugin leave reception mode. This ensures that one can use the mouse for other purposes without disabling this plugin, i.e. it may be desirable to have a track ball interactor plugin enabled to be able to rotate the view.

The plugin_interactor_point_specifier basically implements the formulas described in Section 3.4.2, see Algorithm 3 for the actual implementation. The points are updated when the appropriate key is pressed on the keyboard and the user clicks the mouse so none of the properties are observed for change. When a point is specified by the user, the position of the mouse cursor is converted to an intersection point and stored in the appropriate property.

Note that when the plane normal is perpendicular to the direction of the line, the algorithm will result in a division by zero on line 15 of Algorithm 3. This causes t to become *infinite* and the result is then discarded.

After the point p is computed using Algorithm 3 it is converted to world space and stored in the point property p_n and d_n with n being the index of the point associated with pressed key. The points are finally packed into the matrices mentioned earlier. In the case that p is *infinite*, which as noted above is caused by t being *infinite*, it is discarded and no further action is taken.

Algorithm 3: *ComputeIntersection(p,d)*

Input: the matrices P, V and D
Input: the boolean b determining the reference frame of the slice
Input: the position of the mouse cursor $p = (p_x, p_y)$
Result: the intersection of the plane and the mouse cursor in world space

1 `/* Figure out the correct matrices */`
2 $M_{dw} \leftarrow D$
3 **if** b **then**
4 $\quad M_{ws} \leftarrow P \cdot V$
5 **else**
6 $\quad M_{ws} \leftarrow P \cdot V \cdot D^{-1}$
7 **end**
8 `/* Convert the geometry to world space */`
9 $w_0 \leftarrow M_{ws}^{-1} \cdot [p_x, p_y, 0, 1]^T$
10 $w_1 \leftarrow M_{ws}^{-1} \cdot [p_x, p_y, -1, 1]^T$
11 $n \leftarrow M_{dw} \cdot \vec{n}$
12 $v_0 \leftarrow n \cdot M_{dw} \cdot [0, 0, 0, 1]^T$
13 `/* Compute the intersection */`
14 $\Delta w \leftarrow w_1 - w_0$
15 $t \leftarrow (n \cdot (v_0 - w_0))/(n \cdot \Delta w)$
16 **return** $w_0 + t \cdot \Delta w$

4.2.10 plugin_interactor_coreg

As described in Section 3.4.3 this plugin has the responsibility to compute the registration matrix from two sets of three points usually supplied by two separate plugin_interactor_point_specifier plugins, one for the image slice and the second for the 3D modality. It has the following properties:

[in] Point n Ultrasound the nth point in the ultrasound image data set and is denoted u_n, where $0 \leq n \leq 2$.

[in] Point n Volume the nth point in the volume data set and is denoted v_n, where $0 \leq n \leq 2$.

[out] Coreg Transformation The resulting registration transformation R.

As with the plugin_interactor_point_specifier the main algorithm for this plugin is a straight forward implementation of the formulas described in Section 3.4.3. This plugin observes all its input parameters for change and invokes the computation of the registration transformation described in Algorithm 4 on that event.

As noted in Section 3.4.3 the choice of scale on line 4 and 9 of Algorithm 4 is not relevant as long as the choice is consistent for both U and V.

Algorithm 4: *ComputeRegistrationTransformation()*

Input: two sets of points u_n and p_n, $0 \leq n \leq 2$
Result: the intersection of the plane and the mouse cursor in world space

```
/* Compute the reference frame of the slice; U                    */
```
$u_{v1} \leftarrow u_1 - u_0$
$u_{v2} \leftarrow u_2 - u_0$
$u_{v0} \leftarrow (u_{v1} \times u_{v2})/|u_{v1} \times u_{v2}| \cdot |u_1|$
$U \leftarrow [u_{v0}\ u_{v1}\ u_{v2}\ u_0]$
```
/* Compute the reference frame of the slice; V                    */
```
$v_{v1} \leftarrow v_1 - v_0$
$v_{v2} \leftarrow v_2 - v_0$
$v_{v0} \leftarrow (v_{v1} \times v_{v2})/|v_{v1} \times v_{v2}| \cdot |v_1|$
$V \leftarrow [v_{v0}\ v_{v1}\ v_{v2}\ v_0]$
return $V \cdot U^{-1}$

Also noted in the aforementioned section is that this algorithm aligns u_n to v_n so the user must be careful to let the primary, the secondary and the ternary points in both data sets specify the corresponding landmark in the two modalities. That is; when specifying features A, B and C with the points u_A, u_B, u_C, v_A, v_B and v_C, then if the point $u_i = u_X$ implies that $v_i = v_X$ and vice versa. The algorithm will still perform the registration if this is not the case. Tt will just align the wrong points and the result will be nonsensical.

5 Results

The results obtained with the implementation described in Chapter 4 will be presented in this chapter. The implementation was used in three different scenarios:

Immersive Visualization with Large Stereoscopic Screen The immersive environment at the University of Bergen gave ample opportunity to use the implementation for immersive visualization as described in Section 3.3. The results of this scenario is presented in Section 5.2.

Using the immersive visualization pipeline, the immersive environment at our University was used to give a presentation to a group of students from secondary school and proved to be a huge success with the visitors.

Multimodal Visualization of 2D Ultrasound with 3D Modality Working with our clinical partners at Haukeland University Hospital a pipeline for multimodal visualization of multiple modalities was realized which is described in Section 3.4.4.

5.1 General Purpose Tracking

The first tangible result is a general solution for tracking input for the VolumeShop framework described in Section 3.2.1 and its implementation in Sections 4.2.1, 4.2.2 and 4.2.3. It has been successfully utilized during the course of this thesis by enabling the user tracking on the immersive environment at our university and the tracking of the ultrasound probes, using essentially the same magnetic tracking technology. The software has proven reliable in both cases.

Because of the run-time configuration of the component, such as, the arbitrary configuration of reference frames, and the wide support for tracking devices by the utilized OpenTracker library it is easily configured for almost any tracking hardware.

The latency of this solution, i.e., the time from when the tracker moves until the client is notified, is low. It is hard to measure this latency because the total latency of the system

is the time from when the tracker moves until it is reflected on the display which makes it hard to measure the latency of the two components separately. The time to render a frame is effectively added to the latency of the tracking system when discussing perceived.

To minimize this effect, VolumeShop was configured to run with only the tracking and a matrix renderer to visualize the tracker's frame of reference. The result was a barely noticeable delay from when the user moved the tracker to it was presented on the screen. No good way to measure this latency has been made, but it is at least more than good enough for the scenarios described in this thesis. Essentially it was not noticeable for the viewer.

5.1.1 Precision

As noted in Section 2.2 the precision of the magnetic trackers used in these scenarios are susceptible to metals in the proximity of the tracker and the transmitter. A second problem with the precision and magnetic tracking is that it deteriorates as the distance from the transmitter to the tracker increases.

The problem was especially apparent in the immersive environment at our University as the transmitter is placed on the floor which most certainly contains iron for reinforcement. Moreover, the transmitter for the magnetic trackers is placed at the very right end of the screen which gives the setup a unsatisfactory precision at the opposite end. The setup used for ultrasound tracking did not suffer from these symptoms as the probe was well handled within the range of the transmitter and there was limited amounts of magnetic metals in the proximity. There was some steel tubing in the bed where the volunteer was lying, but the transmitter used in this case were especially designed to handle the interference of metals.

It is at the time of writing plans to hang it from the roof above the user's head to rectify both the problem of proximity to metals and to bring it closer to the user.

5.2 Immersive Visualization

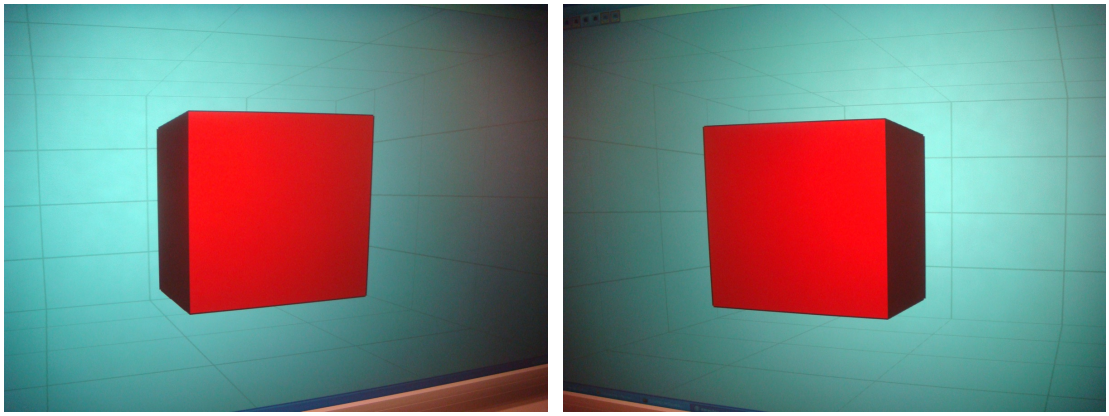
The immersive visualization pipeline described in Section 3.3 has been utilized to give a presentation. It uses the general purpose tracking module described in the previous section for the tracking of the user's head and hand-held interaction device.

As noted in Section 3.3, it consist of two components; the camera controller and the object controller. The camera controller is responsible for computing the correct perspective projection based on the user's position whereas the object controller maintains a model matrix which defines the position and orientation of the object on screen.

5.2.1 Viewpoint Dependent Perspective Projection

As noted in the Section 3.3.2, the image rendered on a display should change according to the position of the user. To verify that this works as described, a cube was rendered which was transformed by the projection matrix and camera matrix computed by the camera controller (described in Section 4.2.6).

Rendering a cube makes it easier to verify the projection because it should at all times look like all its faces are parallel to the screen, floor and walls of the immersive environment. Notice the produced images look similar to Figure 3.2 in Section 3.1.1. Pictures of the resulting images is presented in Figure 5.1. As seen in the pictures, the result were pleasing



(a) From the left (b) From the right

Figure 5.1: *Pictures taken from the right and left side of the display respectively*

and principally correct. Notice also the grid behind the cube which is a part of a box of which the screen is the front face. This box helps to convey the illusion of depth into the screen.

The presented scene appear volumetric due to the stereoscopic display and also act as if they were positioned in front of the user in real-world spatial coordinates. Due to the lack of precision in the left and back of the interaction area and the limitations with non-fullscreen views mentioned in Section 4.2.6, there were slight offsets perceived. These offsets were minor and did not cause any loss of immersion with the exception of the far left hand side where the trackers were out of range of the transmitter. All in all, the illusion created by the viewpoint-dependent perspective projection were believable.

5.2.2 Gesture Based Immersive Interaction

Having correct perspective projection is only half way to full interactive immersive. Interaction with the data, as described in Section 3.3.1, is also needed to achieve believable

immersion. As with the perspective projection, a cube was rendered and the interaction was tested on it. The results of this can be seen in Figure 5.2.

Figure 5.2: *The data follows the gestures of the user*

Observe that the cube follows the user's motion in the same manner as illustrated in Figure 3.4 in Section 3.1.1. The interaction handles as specified in the aforementioned section; in that it preserves translation and orientation changes independent from the reference frames involved. It turns out to be very intuitive to use and as noted in the next section, the students from secondary school who tried it had a learning period measured in seconds.

5.2.3 Demonstration of the Immersive Environment

When the Institute of Informatics was visited by a group of pupils from several secondary schools in the region, the different research groups were asked to make a demonstration to showcase their domain. For the visualization group, the immersive environment was used to host the presentation. It was used to demonstrate the immersive visualization pipeline created in the scope of this thesis and proved to be a impressive experience with the audience.

As noted above, the pupils quickly grasped how the interaction worked as the interaction is aligned with the handling of real-world objects. Pictures of the event can be seen in Figure 5.3. Notice the polarized glasses worn by the visitors which work in tandem with the projectors of the back-projected screen which is similarly polarized to achieve separation of the left and right images.

5.3 Multimodal Visualization

The multimodal visualization of 2D ultrasound and 3D modality described in Section 3.4.4, was successfully utilized to combine ultrasound or elastography with MRI volumes. The

(a) Presentation utilizing the immersive visualization of a 3D CT scan to the visiting pupils demonstrating gesture based interaction

(b) One of the visiting pupils gets a hands-on experience with the immersive visualization pipeline

Figure 5.3: *A presentation of the immersive environment at our University*

general tracking solution was used to track the ultrasound probes and the tracking information was used to successfully register the slices with the MRI volumes.

5.3.1 Ultrasound Scan of a Liver

For the visualization of the ultrasound, there were several acquired ultrasound series which focused on the liver region. In addition an MRI scan was acquired from the same abdominal region of the same volunteer.

The specification of points are done in a straightforward manner, see screenshot Observe that the landmarks are presented as colored dots emphasized by the rings and that they specify the corresponding landmarks in both data sets. The similarity is hard to see in the screen shot since the landmarks in the MRI is not all located in the same cross-section. On the left hand side, there is the controls for the time, the position of the MRI cross-section, the time offset and the frequency. Figure 5.4.

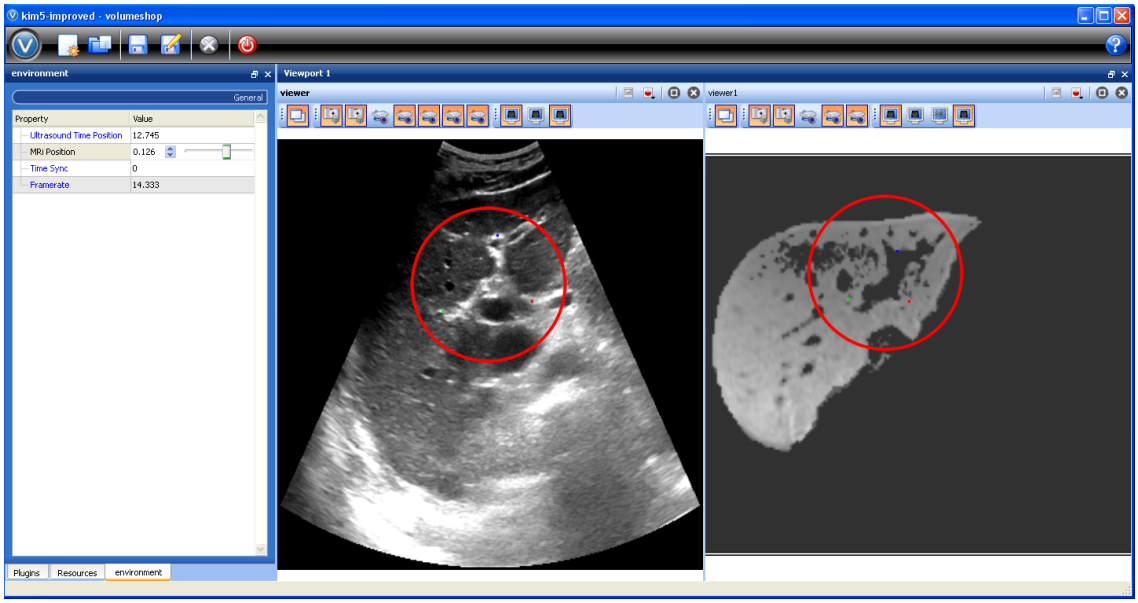


Figure 5.4: *The specification of the landmarks, the colored dots in the rings denote the specified landmarks*

Next, two sets of images are presented which presents the combined visualization with the final registration. The first set of images which shows the spatial location of the registered slice in the MRI volume at a fixed time coordinate are presented in Figure 5.5. Observe that the ultrasound slice is placed, as expected, skin deep into the MRI volume. The different images include only increasingly dense tissue, starting with everything, including the skin, to showing mostly muscle tissue to end up with the segmented liver only.

The second set of images shows the ultrasound slice with only the liver segment visualized in various time steps, see Figure 5.6. It can be seen that the features in the ultrasound slice roughly follow the features of the MRI volume. Notice especially the veins which are black round patterns on the ultrasound slice match closely the veins which show up as cav-

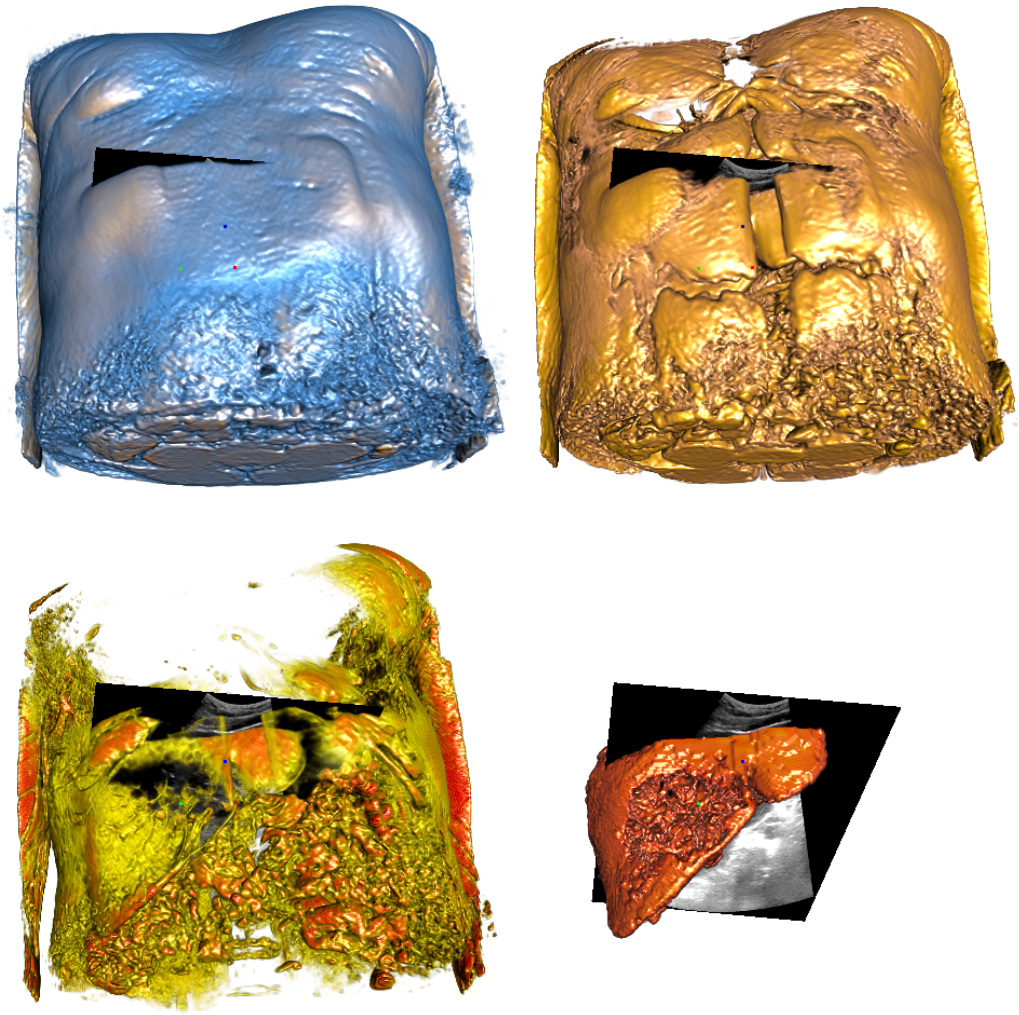


Figure 5.5: *The registered ultrasound slice located in the MRI volume showing the entire volume with various densities omitted, from including skin to only including a segmentation of the liver*

ities in the MRI volume. There are several reasons why the registration is not exact, i.e., the features in the slice only roughly follows the features in the volume. First of all, it is impossible for the volunteer to lie in exactly the same position for both acquisitions, secondly there are parameters such as the amount of air in the subject's lungs, the pressure from the ultrasound probe amongst other things.

Nevertheless, this registration is very close to ideal, but due to the fact that it was the author of this thesis that carried out most of the registration process without any medical background it certainly could be done better. The author is sure that a trained clinician could perform a much better job in identifying and pinpointing landmarks and features in the modalities.

When the ultrasound series is loaded into the pipeline, it becomes apparent that the

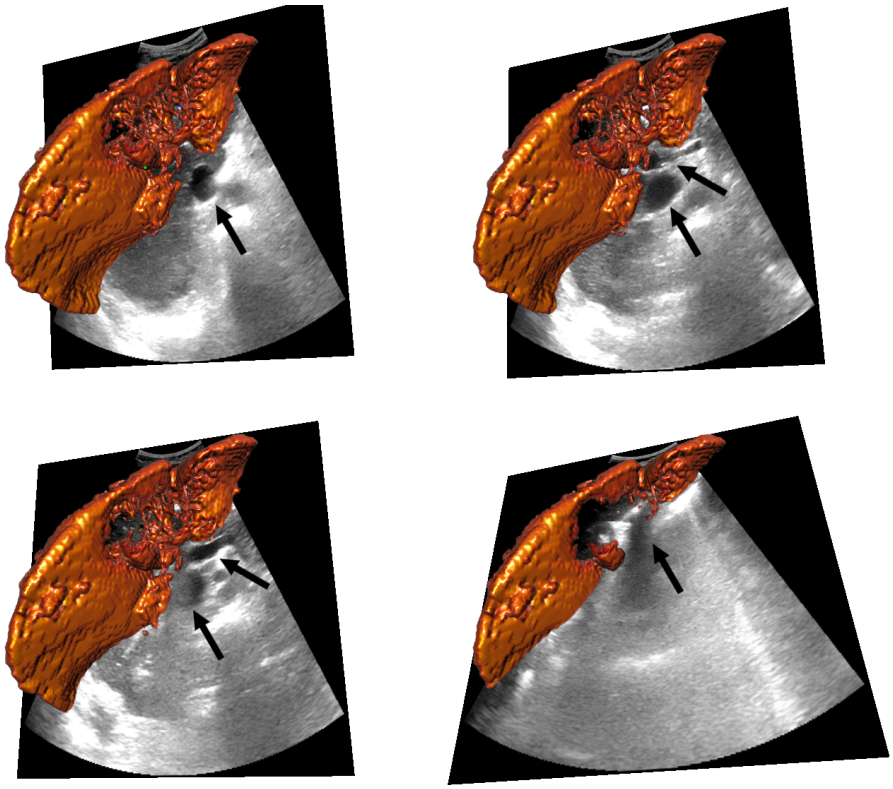


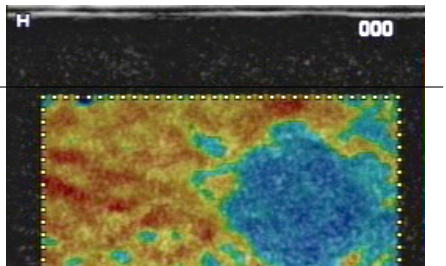
Figure 5.6: *The registered ultrasound slice located in the MRI volume showing only the liver*

slice is transformed according to how the clinician handled the probe during the session by visual inspection of the results of the combined visualization. Notice that the features in the data other than the ones picked for registration are also roughly aligned. The conclusion is then that the registration provides principally sufficiently correct results as described in Section 3.1.2. The following section will discuss the multimodal visualization of the phantom data sets.

5.3.2 Multimodal Visualization of the Elastography Phantom

A very similar scenario, the multimodal visualization of the elastography phantom sessions, is utilizing the same pipeline with similar results. The elastography phantom is basically an elasticity reference tool used to test elastography ultrasound scanners amongst other things and is described in Section 2.1.2. The elastography phantom was scanned by an ultrasound scanner capable of producing elastograms from the scanned images. The phantom was also scanned in an MRI scanner to obtain a reference volume.

The elastography images are basically processed B-mode ultrasound images. The clinician pushes the probe gently into the tissue to obtain ultrasound



images of the tissue with varying pressure applied. These images are the processed to produce the elastograms. The clinician can only select a portion of the slice to be analyzed. For this reason, the images has been cropped so they only include the elastogram and the image up to the top, see Figure 5.7 for an example.

The first set of images, see Figure 5.8, shows the phantom slightly transparent with the embedded elastography slice. Note that the slice follows a linear

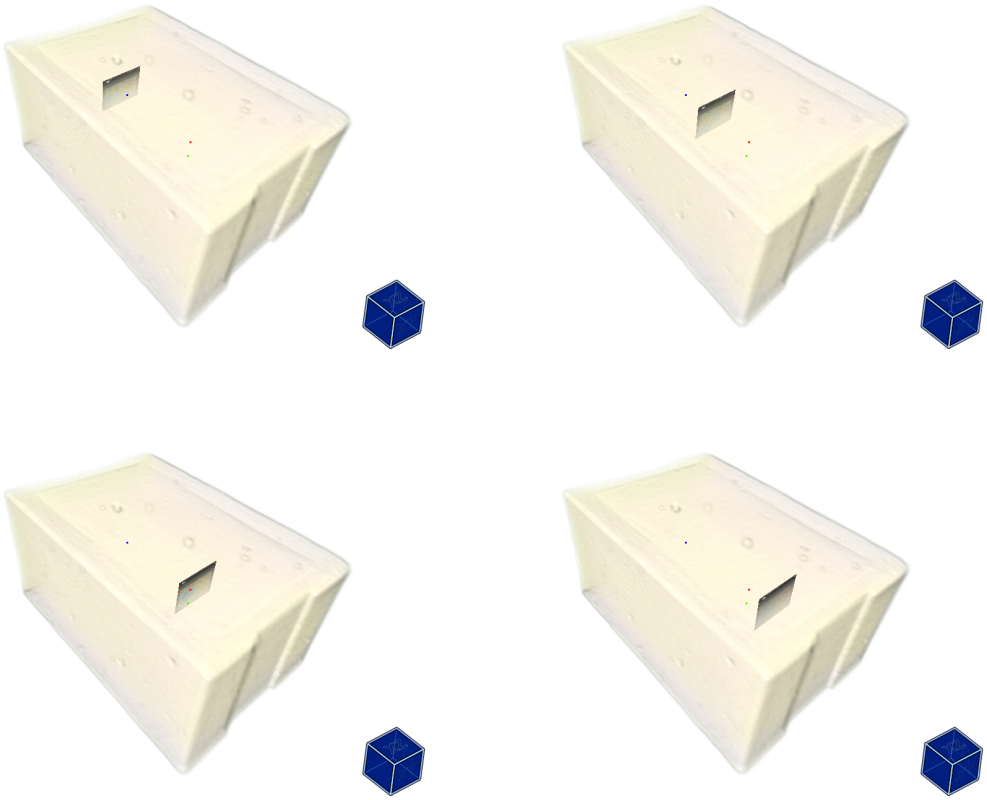
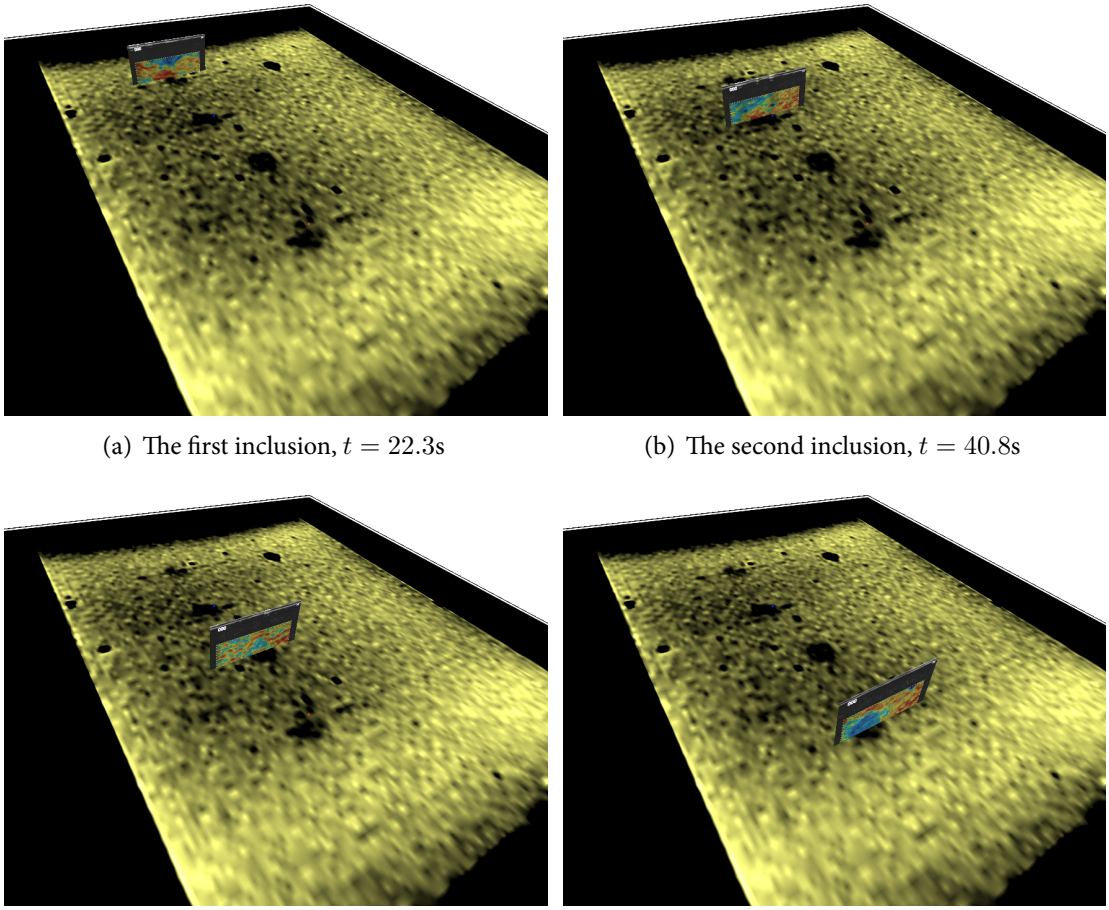


Figure 5.8: *The registrered elastography slice located in the MRI volume of the phantom*

motion along the length of the phantom which is exactly what the clinician did during the acquisition. Observe also that the inclusions are impossible to discern in the MRI volume. The reason for this is that the spherical inclusions have approximately the same intensity in the volume as the surrounding matter and that the direct volume rendering technique is not well suited to emphasize such small differences.

The inclusions are on the other hand possible to discern when the MRI is rendered as a cross-section, shown in Figure 5.9. In these images the elastography slices are rendered



(a) The first inclusion, $t = 22.3$s

(b) The second inclusion, $t = 40.8$s

(c) The third inclusion, $t = 65.2$s

(d) The fourth inclusion, $t = 97.9$s

Figure 5.9: *The registered elastography slice located in the MRI volume of the phantom*

together with a perpendicular cross-section of the phantom which is placed at the depth of the inclusions, effectively making them visible. Visualized are the time steps where the slice passes the inclusions. The second and third inclusions are difficult to spot in the elastograms since their elasticity are much closer to that of the surrounding matter than the first and fourth inclusions.

It can be observed that the inclusions align fairly accurately in the two modalities. In this case, the fourth and second inclusions were used as the registration features. The result is that the first, second and third inclusion are spot on, whereas the first ball has a small very slight as can be seen in Figure 5.9(d).

This elastography result concludes this chapter. All in all, the pipeline showed promising results in all scenarios. From intuitive immersive visualization and interaction with the immersive visualization pipeline and to the multimodal visualization of ultrasound and

elastography slices with MRI volumes the pipelines did what they were designed to do. In the medical domain related example, a thorough clinical examination will be essential to draw conclusions with respect to its utility in the daily clinical routine. In the next chapter a short summary of this work is presented, followed by a conclusion.

6 Summary

6.1 Summary

This thesis has presented two pipelines. One for immersive visualization on a large screen stereo display described in Section 3.3 and another of multimodal visualization of 2D ultrasound and 3D modality described in Section 3.4. The pipelines are implemented as a set of plugins for the VolumeShop framework.

6.1.1 Motion Tracking

The tracking hardware utilized in this thesis is the Flock of Birds made by Ascension described in Section 2.2. The Flock of Birds is a magnetic tracker which provides fairly precise, low latency tracking but are susceptible to magnetic distortions caused by nearby metals.

Both pipelines utilize a common component for interfacing the motion tracking equipment described in Section 3.2.1. The motion tracking component utilizes the OpenTracker library for the tracking input and combines the position and quaternion acquired from OpenTracker into a homogenous 3D matrix which it makes available to the other plugins in the VolumeShop framework. As noted in Section 2.2.2, OpenTracker was chosen because it was open source, has support for a wide array of devices and can be run-time configured.

The tracking component is also able to write the transformation to a log file. A second component is able to replay this log file which makes it possible to perform multimodal visualization off-line after the acquisition session. This replay component were used extensively in the multimodal visualization pipeline during the course of this thesis.

A third component compensates the tracker matrix M for any difference in the reference frames of the tracker versus the tracked object and of the transmitter versus the target reference frame, i.e. the display. It works by letting the user specify two homogenous matrices which define these spatial relationships, T and C for the transmitter and the tracked object respectively. The resulting compensated matrix is then $M' = T \cdot M \cdot C$. The compen-

sation component also includes a registration matrix R which is applied to the compensated matrix M' to obtain the registered compensated matrix $M'' = R \cdot M'$.

6.1.2 Immersive Visualization

The immersive visualization pipeline uses the tracking component described above to provide head and gesture tracking. A head mounted tracker provides the position and orientation of the users head and an additional tracker is incorporated into the interaction device to provide hand gesture tracking.

The position and orientation the two trackers are compensated for the offset to the tracked objects. In the case of the head tracker the tracked object is the nose ridge and in the case of the interaction device tracker no compensation was necessary. They both share the same transmitter to screen space transformation which basically translates the origin from the transmitter to the center of the display.

View-point Dependent Perspective Projection

Using the compensated head position and orientation, the position of both eyes is computed and is used to compute the perspective projection independently for each eye. This is done to achieve the "looking through a window" illusion which makes it possible to view the rendered objects from different angles by moving around the data.

To accomplish this illusion, the perspective projection needs to match the view-point which then defines the viewing frustum as the pyramid formed by the viewpoint and the edge of the screen. The projection matrix places the camera in the origin. Since the screen is defined to be the origin, a translation matrix is created to translate the camera back so that the screen becomes the origin. These projection and camera matrices are fed into two distinct but equal rendering pipelines, one for each eye. It would be more convenient to only have configure the plugins for stereo once and use it to render both the left and right image, but VolumeShop does not yet support this.

Interaction Gestures

By using the interaction device position and orientation, hand gestures can be implemented. This thesis implemented only one hand gesture which is activated as long as a button on the interactor is pressed and causes the data to follow the user's movement as described in Section 3.3.1. The requirement is that the interaction component should preserve the translation and orientation of the gesture.

To accomplish this gesture, the compensated interaction device matrix is stored in the interaction component. When the matrix is updated, the difference between the previous

matrix M_{n-1} and the new matrix M_n is computed by $\Delta M_n = M_n \cdot M_{n-1}$. This difference matrix is applied to the model matrix D_{n-1} after it has been translated to M_{n-1}'s origin. Finally, the matrix is translated back by the same translation to obtain:

$$D_n = T_{n-1}^{-1} \cdot \Delta M_n \cdot T_{n-1} \cdot D_{n-1} \quad \text{where } T_i = \mathrm{Tr}(M_i) \cdot \mathrm{Tr}(D_i)^{-1} \tag{6.1}$$

Equation 6.1 allows the user to translate and rotate the object by moving the interactor in the same manner. This concludes the summary of the immersive visualization pipeline, the next section will describe the multimodal visualization pipeline.

6.1.3 Multimodal Visualization

The multimodal visualization pipeline is used in this thesis to visualize ultrasound time series combined with MRI as described in Section 3.4. This pipeline is realized by presenting three views to the user, one for the ultrasound slice, one for a cross section of the MRI and one which presents the combined visualization

The first problem is to spatially relate the ultrasound slices to each other. Thus pipeline annotates the ultrasound slices with position and orientation to be able to spatially relate them to each other and to the MRI volume. The motion tracking component was used to log the motion tracking of the ultrasound probe during the acquisition sessions.

Since the ultrasound slices and the accompanying motion tracking data is acquired on different machines which lacked a suitable synchronization mechanism, they needed to be synchronized manually after the acquisition. This is done in the implementation by inspecting the ultrasound slices in two views. The user needs to spot a motion in the slice and the corresponding gesture in the tracking data and to specify a time offset between the two. The frequency of the slices also needs to be specified and is usually known as it is a configuration setting on ultrasound acquisition devices. Once the offset and frequency is properly defined, the ultrasound is synchronized with the motion tracking and one may continue to specify the registration.

Registration is done by specifying three corresponding points in the two data sets. These points are specified by the user in the ultrasound view and the cross-section view. The three points are used to define a frame of reference of both modalities. The frames of reference for the ultrasound U and for the MRI M are then used to compute the registration matrix R which basically maps the points in the ultrasound slice to the points in the MRI volume and is computed as follows:

$$R = M \cdot U^{-1} \tag{6.2}$$

Using the registered compensated motion tracking matrix for the slice supplied by the compensation component, the slices are transformed into the reference frame of the MRI

volume and are rendered using a custom slice renderer. The MRI volume is rendered using the built-in direct volume rendering component.

This section concludes the summary of the two pipelines described and implemented in this thesis. For implementation details of the components see Chapter 4 and for results see Chapter 5.

6.2 Conclusion

The cornerstone of this thesis; the incorporation of tracking support into VolumeShop has had its share of difficulties. The first of these difficulties was to integrate OpenTracker into the motion tracking component. The choice of motion tracking middle-ware proved to be sub-optimal and has caused much delay in the implementation of this thesis, Section 6.2.1 will describe this in detail. After the tracking support was successfully integrated, the implementation of the pipelines went much smoother.

The immersive visualization pipeline was implemented without difficulties and proved to give satisfactory results. User feedback confirmed that the pipeline performs satisfactory. New users managed to interact with the immersive environment without any prior training confirming that the interaction is very intuitive.

The choice of using homogenous transformation matrices to represent all transformation has simplified the implementation due to the elimination of conversions to and from other representation within the pipelines.

Using homogenous matrices for the computation of the registration matrix has proven simple to implement and has yielded satisfactory results. The formulas are very simple, basically consisting of only one matrix-inversion and one matrix-multiplication. This approach, while being simple, has its drawbacks. If the modalities are not acquired simultaneously, the patient will most certainly move between the acquisitions. This causes the different modalities will contain slightly deformed data compared to each other.

Using a rigid transformation for the registration transformation makes it impossible to register such nonlinear deformations. This becomes apparent after specifying the landmarks which then may result in a non-orthogonal registration matrix which will result in a sheared ultrasound slice in the combined view. More advanced registration techniques could have been utilized to enable a more correct registration of the modalities. For the particular data sets used in this thesis, the linear registration transformation produced satisfactory results. The results of this pipeline were discussed in Section 5.3.

6.2.1 OpenTracker

As noted in Section 2.2.2, OpenTracker was the motion tracking middle-ware of choice. OpenTracker is a great middle-ware solution for tracking, but it has caused some problems. After overcoming these problems, it has performed admirably.

One of the decision factors for choosing OpenTracker over VRPN were its support for the Wiimote. It turned out that this support was questionable at best as it did not provide positional or orientational data, only the position on the screen it pointed at.

The Flock of Birds module for OpenTracker also left something to desired. It first became operable after the author of this thesis fixed a couple of bugs in the initialization procedure. This was possible due to OpenTracker being open source software.

Furthermore, the OpenTracker came with a stand-alone executable which just starts OpenTracker up given the configuration file as a parameter. Configuring it with a terminal sink node, the tracking information is displayed in a terminal window. The stand-alone executable were used for testing and determining the proper configuration for the Flock of Birds but somehow never worked properly. After some time, the trials stand-alone executable were abandoned and a test program were developed which linked directly with the OpenTracker library. For reasons still unknown, this worked with the same configuration the stand-alone executable failed with.

In retrospect, it can be argued that VRPN would have been the better choice and could have caused less problems. However, the choice was reasonable at the time taken the available information of the two into account.

6.2.2 VolumeShop

VolumeShop is a very good framework for developing amongst other visualization techniques; direct volume rendering techniques. VolumeShop is not publicly released as of the time of writing and its lack of maturity shows in some areas.

The `Matrix` class uses homogenous matrices but the `Vector` class uses regular 3D coordinates which are treated as vertices when multiplying with matrices; that is the w coordinate is 1. This causes problems where one would like to perform matrix-vector multiplication and want the vector to act as a vector ($w = 0$). If M and N are homogenous matrices and $M = [x_M\ y_M\ z_M\ w_M]$, then:

$$N \cdot M \neq [N \cdot x_M \;\; N \cdot y_M \;\; N \cdot z_M \;\; N \cdot w_M] \tag{6.3}$$

which should definitively be equal as it is the definition of matrix multiplication. The matrix-matrix multiplication is correctly performed as it handles the x_M, y_M and z_M as vectors

and w_M as a vertex. The matrix-vector multiplication is not correctly performed since the VolumeShop has no way to distinguish between vectors and vertices as they both are represented by the `Vector` class.

The conclusion regarding VolumeShop is that it is a very good framework for visualization purposes but can be tricky to work with if one is unfamiliar with its quirks. A development documentation would significantly ease the implementation of prototypes in this otherwise very flexible framework.

6.3 Future work

Since this thesis was mainly focused on implementing the basic functionality to support research immersive environments, the implemented pipelines served as proof of concept.

The immersive visualization pipeline proved to perform its task in a satisfactory manner. Future work in this area can focus on the interaction aspect. More interaction concepts can be implemented such as a laser pointer enabling the selection of objects on the screen with the tracker and a wide array of gestures such as flicks of the wrist in a specific direction to invoke commands.

The multimodal visualization pipeline can be improved in several areas such as the registration technique, the time synchronization and the visualization itself:

- The registration could for example implement non-rigid registration transformation to enable the registration of deformed data.
- The time synchronization could also utilize a landmarks method where points in time where a correspondence is known is marked and the offset and frame-rate computed by the computer based on these landmarks.
- The visualization could employ multiple transfer functions, for example, one applied to the part of the volume in front of the ultrasound plane and a second applied to the part behind the ultrasound plane. The transfer function in front would then in general be less opaque than the one behind to show the structures in the volume at the slice. The slice could then be integrated into the volume by making it semi-transparent so that the structures behind become visible.
- The multimodal visualization pipeline was designed with real-time visualization in mind. Therefore the existing components are capable of handling real-time input of ultrasound slices. The completion of the real-time acquisition of ultrasound images only needs the implementation of a plugin which interfaces a frame grabber. This should be fairly easy and would realize one of the goals of this thesis but which were foiled by the lack of appropriate hardware, e.g. a frame grabber.

Acknowledgements

I would like to thank our clinical partners at Haukeland University Hospital for their help on the acquisition of the medical data sets and for lending us tracking equipment. I would also like to thank my supervisor Ivan Viola, and Daniel Patel for their help on making this thesis even remotely readable.

Bibliography

[1] Anisfield, N. Ascension technology puts spotlight on DC field magnetic motion tracking. *HP Chronicle 17*, 9 (August 2000).

[2] Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., and Cruz-Neira, C. VR juggler: A virtual platform for virtual reality application development. In *VR* (2001), p. 89.

[3] Bloomenthal, J., and Rokne, J. Homogenous coordinates. *The Visual Computer 11*, 1 (January 1994), 15–26.

[4] Bruckner, S., Grimm, S., Kanitsar, A., and Gröller, M. E. Illustrative context-preserving volume rendering. In *EuroVis* (May 2005), pp. 69–76.

[5] Bruckner, S., and Gröller, M. E. VolumeShop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization* (october 2005), pp. 671–678.

[6] Bruckner, S., and Gröller, M. E. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum 26*, 3 (Sept. 2007), 715–724.

[7] Burns, M., Klawe, J., Rusinkiewicz, S., Finkelstein, A., and DeCarlo, D. Line drawings from volume data. *ACM Trans. Graph. 24*, 3 (2005), 512–518.

[8] CIRC. Elasticity QA phantom. http://www.cirsinc.com/pdfs/049.pdf.

[9] Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In *SIGGRAPH* (1993), pp. 135–142.

[10] Fröhlich, B., Hochstrate, J., Hoffmann, J., Kl□ger, K., Blach, R., Bues, M., and Stefani, O. Implementing multi-viewer stereo displays. In *WSCG conference proceedings* (2005).

[11] Gavigan, P., and Pearce, T. Implementation of v-cave display system using the object-oriented graphics rendering engine. *CCECE* (2008), 1309–1312.

[12] Guthart, G., and Salisbury, J.K., J. The intuitivetm telesurgery system: overview and application. *ICRA 1* (2000), 618–621 vol.1.

[13] Hauser, H. Generalizing focus+context visualization. *Scientific Visualization: The Visual Extraction of Knowledge from Data* (2003), 305–327.

[14] Jr., J. J. L. A discussion of cybersickness in virtual environments. *ACM SIGCHI Bulletin 32*, 1 (January 2000), 47–56.

[15] Kitamura, Y., Konishi, T., Yamamoto, S., and Kishino, F. Interactive stereoscopic display for three or more users. In *SIGGRAPH* (2001), pp. 231–240.

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

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

[18] Maintz, J. B. A., and Viergever, M. A. A survey of medical image registration. *Medical Image Analysis 2*, 1 (1998), 1–36.

[19] Nixon, M. A., McCallum, B. C., Fright, W. R., and Price, N. B. The effects of metals and interfering fields on electromagnetic trackers. *Presence: Teleoper. Virtual Environ. 7*, 2 (1998), 204–218.

[20] Pluim, J., Maintz, J., and Viergever, M. Mutual-information-based registration of medical images: a survey. *Medical Imaging, IEEE Transactions on 22*, 8 (Aug. 2003), 986–1004.

[21] Reitmayr, G., and Schmalstieg, D. Opentracker-an open software architecture for reconfigurable tracking based on XML. *Virtual Reality, Proceedings. IEEE* (March 2001), 285–286.

[22] Russell M. Taylor, I., Hudson, T. C., Seeger, A., Weber, H., Juliano, J., and Helser, A. T. Vrpn: a device-independent, network-transparent VR peripheral system. In *VRST* (2001), pp. 55–61.

[23] Sandin, D. J., Margolis, T., Ge, J., Girado, J., Peterka, T., and DeFanti, T. A. The varriertm autostereoscopic virtual reality display. In *SIGGRAPH* (2005), pp. 894–903.

[24] Shreiner, D., Ed. *OpenGL Reference Manual - Blue Book*, 4 ed. Addison-Wesley Professional, 2004.

[25] Wein, W., Roper, B., and Navab, N. Integrating diagnostic b-mode ultrasonography into ct-based radiation treatment planning. *Medical Imaging, IEEE Transactions on 26*, 6 (June 2007), 866–879.

[26] Welch, G., Bishop, G., Vicci, L., Brumback, S., Keller, K., and Colucci, D. High-performance wide-area optical tracking: The HiBall tracking system. *Presence: Teleoperators and Virtual Environments 10*, 1 (2001), 1–21.

[27] Westermann, R., and Ertl, T. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH* (1998), pp. 169–177.