2009-02-17

Interactive Training System for Medical Ultrasound

Christian John Banker
Worcester Polytechnic Institute

INTERACTIVE TRAINING SYSTEM FOR MEDICAL ULTRASOUND

by

Christian J. Banker

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Electrical and Computer Engineering
by

January 2009

APPROVED:

Dr. Peder C. Pedersen, Major Advisor

Dr. Edward A. Clancy

Dr. Thomas L. Szabo

Abstract

Ultrasound is an effective imaging modality because it is safe, unobtrusive and portable. However, it is also very operator-dependent and significant skill is required to capture quality images and properly detect abnormalities. Training is an important part of ultrasound, but the limited availability of training courses presents a significant hindrance to the use of ultrasound being used in additional settings.

The goal of this work was to design and implement an interactive training system to help train and evaluate sonographers. The Interactive Training System for Medical Ultrasound is an inexpensive, software-based training system in which the trainee scans a lifelike manikin with a sham transducer containing a 6 degree of freedom tracking sensor. The observed ultrasound image is generated from a pre-stored 3D image volume and is controlled interactively by the sham transducer's position and orientation. Based on the selected 3D volume, the manikin may represent normal anatomy, exhibit a specific trauma or present a given physical condition.

The training system provides a realistic scanning experience by providing an interactive real-time display with adjustable image parameters such as scan depth, gain, and time gain compensation. A representative hardware interface has been developed including a lifelike manikin and convincing sham transducers, along with a touch screen user interface. Methods of capturing 3D ultrasound image volumes and stitching together multiple volumes have been evaluated. System performance was analyzed and an initial clinical evaluation was performed.

This thesis presents a complete prototype training system with advanced simulation and learning assessment features. The ultrasound training system can provide cost-effective and convenient training of physicians and sonographers. This system is an innovative approach to training and is a powerful tool for training sonographers in recognizing a wide variety of medical conditions.

Acknowledgements

I would like to acknowledge Dr. Peder C. Pedersen for his guidance, input, and financial support. His support and guidance truly played a major role in my success. I would like to thank the Telemedicine and Advanced Technology Research Center (TATRC), for providing the grant under which this research was performed.

Jason Bryan of the Ohio Supercomputer Center was a tremendous help, providing his *gear* library, along with support and assistance.

For providing the *Stradwin* and *Stradx* software, along with support, I would like to acknowledge Dr. Richard Prager, Dr. Andrew Gee, Dr. Graham Treece and the rest of the Medical Imaging Group at the University of Cambridge in England.

The transducer shells used to construct the sham transducer were provided free of charge by Sound Technology, Inc.

Patrick Morrison and Tom Angelotti of the WPI ECE shop were a valuable resource for finding and ordering parts and machining the calibration fixture.

I would also like to thank Dr. David Polan and Dr. Dana Resop of the University of Massachusetts Memorial Medical Center for their advice and evaluation of the system and Dr. Joyoni Dey for providing her 3D registration code and assistance with using it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ultrasound is a useful medical imaging modality because it is portable, minimally invasive and relatively inexpensive. It does not use ionizing radiation like Computed Tomography (CT) and is therefore regarded as safe imaging modality. However, the quality of ultrasound images is very operator dependent, requiring significant skill and experience to perform scans and identify conditions. Thus, extensive training and certification is required for sonographers. This has generally proven to be a significant bottleneck and has prevented ultrasound from becoming more widely used in additional settings. To simplify the training process and help train more sonographers, computer-based training systems have been developed.

1.1 Other Ultrasound Simulation and Training Systems

Other ultrasound simulation systems have previously been developed or are under development, but none have yet come into widespread use. A system developed at the University of Leeds [53] is focused on needle guidance and uses simulated ultrasound data generated from CT scans. Data are generated by manually segmenting 3D

CT data and applying selected ultrasound textures to these regions. Processing to emulate ultrasound-specific artifacts is then applied to the image to produce speckle, shadowing and radial blurring. A study was performed to evaluate whether a group of 10 novice subjects were able to improve their needle insertion performance between two temporally separated training sessions. It was shown that every novice tested was able to show statistically significant improvement in at least 6 of the 10 studied performance metrics [21].

VOLUS [47] focuses on 3D surface mapping of the heart. It uses data captured from human patients, but no attempt is made to combine multiple 3D volumes. As of yet, no clinical trials have been published.

The UltraSim system [2] has a full-size mock ultrasound imaging system and uses mechanically-controlled acquisition of ultrasound data. Work has been performed on registering multiple 3D data files through a 2D registration approach. UltraSim focuses on obstetrics, gynecology and internal medicine [22].

SONOSim3D [12] is designed to train for a variety of parts of the anatomy. This system uses the computer mouse to move the current slice and does not provide any hands-on interactive control. The data volumes are captured as 3D scans of human subjects using mechanically-controlled acquisition that rotates the transducer in place; only single volumes are used and no stitching of data is performed.

SonoTrainer [22] is focused on obstetrics, gynecology, and internal medicine and uses 3D ultrasound data. It uses a manikin and sham transducer; however, the tracking system is only three-dimensional and only one abdominal quadrant can be scanned at a time [46]. The makers of SonoTrainer surveyed a group of physicians, asking them subjective questions about the training experience and data quality; 80% of those surveyed judged the image quality as the highest rating of "good" and 90% rated the training effect as being "good" [22]. Training effectiveness was also evaluated by testing sonographers' ability to detect abnormalities on the simulator and in real patients using an ultrasound system. The results of this test showed that

the system was effective because sonographers performed almost the same on the simulation system as they did scanning a patient.

1.2 Interactive Training System for Medical Ultrasound

The Interactive Training System for Medical Ultrasound is an interactive, PC-based ultrasound simulation tool. The system dynamically displays views of stored data, based on user-controlled input from a 6 degree of freedom (DoF) position sensor. As the user moves a sham transducer across the surface of a manikin, the computer display is updated in real time to show a corresponding ultrasound B-mode image. This sham transducer contains only a position and orientation sensor and no actual transducer elements. By loading different data sets, trainees can become skilled at identifying various pathologies or traumas, without having to train on human subjects. Because this system is software-based, rather than using different phantoms for each disorder, an inexpensive generic manikin can be used for any pathology. This system has the benefits of being simple, inexpensive, and flexible, and has the potential to be a very useful training tool.

The primary components of the system design are the training system, the acquisition of composite 3D data volumes, and learning assessment features to assist with the learning process. The training system consists of software that generates scan planes based on the position and orientation input from a 6 DoF tracking system. Interactive scanning features have been developed to provide a realistic scanning environment, permitting users to select different probe geometries and adjust a variety of image parameters, such as gain, time gain compensation, and scan depth. These features have been designed to emulate their equivalents in a true ultrasound system to provide nearly all of the functionality a sonographer would expect. All of these features are accessible through a graphical user interface that provides simple, easy to

use controls. Additionally, a realistic hardware interface has been designed, including an anatomically correct manikin and convincing sham transducers to provide a truly realistic scanning experience. Methods of acquiring simulation data and stitching together multiple 3D data volumes were developed and evaluated to provide a means of generating simulation data. Learning outcomes assessment tools help trainees improve their skills and provide a means of assessment. These tools include the ability to capture still images, and to record scan paths and play them back on a surface model of the manikin. The system can also evaluate a trainee's ability to identify anatomical features and pathologies by requiring him or her to select the features using the touch screen and then determining if the features were properly selected.

1.3 Motivation

Part of the ongoing research being conducted in the Ultrasound Research Laboratory at Worcester Polytechnic Institute is the development of rugged, portable ultrasound systems for first responders and battlefield medicine. These systems open up doors to expanded use of ultrasound by permitting it to be used in situations where it previously could not. With this increased use, however, comes a need to efficiently train more sonographers. The goal of this training system is to streamline the training process and provide effective, low-cost training opportunities. By reducing the need for real ultrasound hardware and human subjects, training can be accomplished more quickly and at a lower cost. Additionally, training does not necessarily need to take place in a clinical environment, providing greater flexibility.

1.4 Thesis Outline

Chapter 2 discusses the interactive generation of scan planes including software selection, volume rendering and interactive control. A number of software programs

and toolkits were evaluated to choose an appropriate framework for development of this system. Basic volume rendering was achieved and then the capability of generating 2D scan planes was implemented. Interactive control was developed first using an inexpensive 6 DoF input device and then with more effective magnetic tracking systems.

Chapter 3 expands upon the basic scan plane display described in Chapter 2 with additional interactive features, as well as graphical and hardware user interfaces. A number of interactive features, such as gain control, scan depth and time gain compensation are described, along with the details of their implementation. The graphical user interface that controls these features is also presented in detail. The hardware interface is discussed, including selection of components and the design of realistic sham transducers.

Chapter 4 details the data acquisition process, including 3D image volume capture and stitching together of 3D volumes. Methods of capturing data and performing calibration are evaluated and discussed. Ways of stitching together multiple 3D data volumes to form a composite 3D volume are also examined.

In Chapter 5, additional learning assessment tools that have been implemented are discussed. These features help ensure that trainees learn from the program by providing helpful tools and assessment of learning outcomes.

Chapter 6 presents the results of this project as well as data from a clinical evaluation.

Chapter 7 provides conclusions on the overall system and its effectiveness. Opportunities for future work are also presented and discussed.

Chapter 2

Interactive Generation of Scan Planes

2.1 Introduction

The purpose of this training system is to provide an interactive environment in which the user can perform simulated ultrasound scans in a realistic manner. Figure 2.1 shows a simplified block diagram of the training system. The user scans a lifelike torso and head manikin containing an embedded Ascension Technology Corporation *trakSTAR* [4] transmitter using a sham transducer containing the 6 DoF tracking sensor. Based on the sensor's position and orientation, a 2D slice is generated from a 3D image volume in real time and displayed on the computer screen. A touch screen is provided for additional interactive features. Based on the selected 3D volume, the manikin may represent normal anatomy, exhibit a specific trauma or present a given physical condition.

This chapter discusses the design of the basic training system and specifically details the process of interactively generating 2D scan planes from a 3D image volume. The first step in implementing interactive scan plane generation was to select the base software that would be used as a development starting point, whether it be a complete

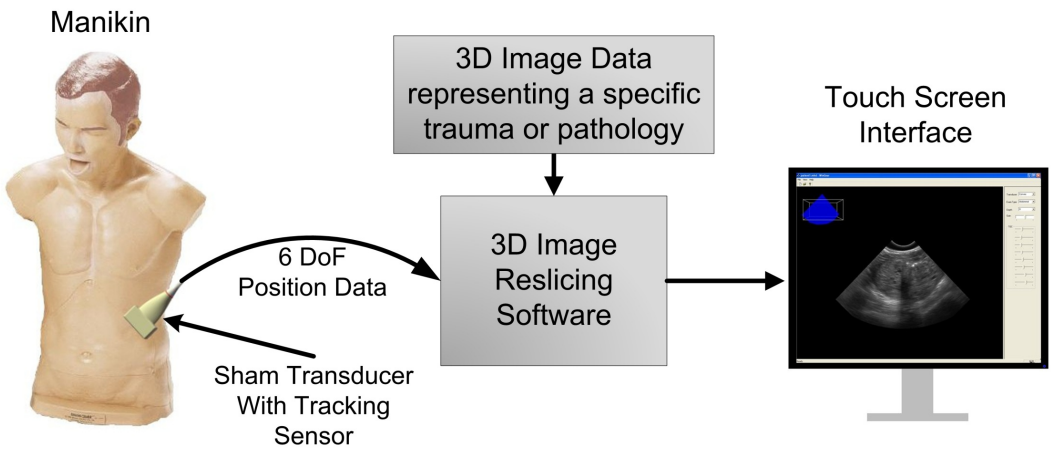


Figure 2.1: Training System Block Diagram

software package or open source libraries. The next challenge was implementing basic rendering of 3D data volumes and creating a display that closely mimics a 2D scan plane. The final step was creating a truly interactive environment by integrating a 6 DoF input system. The following sections thoroughly document this development process and present the design challenges that were faced.

2.2 Software Selection

The first step in designing the image generating software for the training system was the selection of a means of developing it. This could be in the form of a complete software package, a development toolkit or the option to develop a custom software program. Clearly a more complete solution would speed development, but it is also important that the design not be restricted by the limitations of the selected development tools.

In its completed state, the software system must fulfill the following requirements:

- Load 3D volumes

- Create 2D scan images in real time
- Accept input from a 6 DoF position sensor
- Use this input to determine the position and orientation of the 2D slice
- Permit a touch screen interface to be added

Figure 2.2 shows a block diagram of the basic functional requirements for the training system and how they are related. The 6 DoF input is used to determine the transformations for the 2D reslice. These transformations are applied to the 3D data volume to generate a 2D scan image, which is displayed on the touch screen.

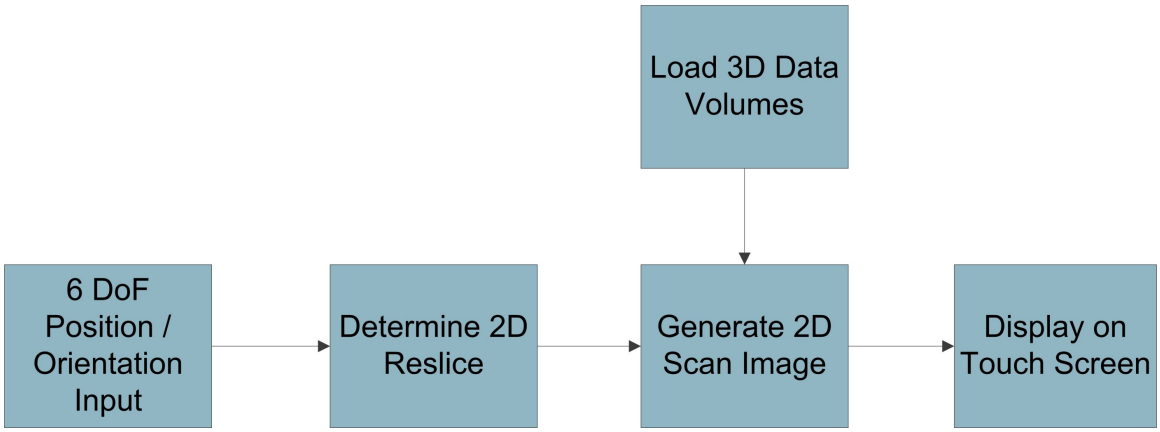


Figure 2.2: Block Diagram of Basic Training System Functional Requirements

The image generation software must be developed to be easy to use and to closely emulate the user experience of a real ultrasound scanner. It must also be capable of running at a frame rate high enough for smooth motion and should lend itself well to the addition of training and evaluation tools.

Starting with some of the desired system capabilities can speed development and potentially provide advanced features. Although the selected starting point does not necessarily need to fulfill all of the goals of the system, it must be possible to implement them through further development.

There are 3 basic types of development platforms that could be used for this system:

- Custom-designed software
- Open source library or toolkit
- Complete software package

Designing the completely custom software is a difficult and time-consuming venture, but would result in an entirely customized system. If possible, it is easier to begin with some type of framework to avoid spending time on basic functions.

A good compromise between a completely custom design and an off-the-shelf software solution is to design custom software using an open source library or toolkit. There are a number of libraries available that focus on 3D rendering and medical imaging. Using one of these toolkits could provide some of the basic functions for the system and would limit required development to the design of new features.

A complete software package is a great solution if it performs most or all of the required functions, but it must permit modifications to be made to fit the needs of this project. Closed source software tends to be highly developed and well supported, but generally comes at a cost. Use of any closed source software would require that either the source code or a set of development tools be available to add features to the software for the needs of this project. This is generally not the best option for a research project because of cost and intellectual property concerns. Open source software is publicly available and source code is openly provided. An open source system that implements useful features could provide a strong starting point with few limitations.

A number of different options were studied and evaluated, which are described in sections 2.2.1 and 2.2.2. The most relevant features are the capability to render 3D volumes and provide the capability to perform arbitrary generation of 2D slices from a 3D volume.

2.2.1 Complete Packages

A number of complete software packages were evaluated to determine if any were a suitable solution. The following sections describe some of the programs that were taken under consideration and put through more in-depth evaluations.

Mayo Clinic's *Analyze*

The Mayo Clinic's *Analyze* software [3] is a powerful, full-featured medical imaging package designed primarily for MRI, PET, and CT use. It is capable of performing advanced visualization, registration and segmentation functions.

A full-featured demonstration version of this software was evaluated and the oblique slicing tool was very effective for viewing arbitrary slices of volumes. It is fast and simple and has the ability to accept a matrix specifying the orientation for the slice. However, it does not have any built-in capability for mapping the rotation to any input other than clicking buttons on the screen.

Although the oblique slicing tool works well, only a small portion of the program's capabilities would be put to use in this application. The numerous, powerful features make *Analyze* a rather expensive piece of software, which costs nearly $5000 for a single node-locked license, and an additional $3000 for the developer's add-on that would be necessary to build upon the software. With an unlimited budget, this could be a good option, but realistically, it is not worth the price to use only a small part of the feature set.

VolSuite

VolSuite [8] is an open source volume viewing and editing application developed by Jason Bryan of the Ohio Supercomputer Center. It is designed as a framework for development that includes an application for volume viewing and analysis with the capability to add custom modules for additional features. It is no longer under active

development, but the developer is still willing to provide support to some degree.

VolSuite is a very useful application for working with 3D medical imaging data. It provides features for viewing, analyzing and modifying 3D volumes. *VolSuite* is capable of working with a variety of different data formats and can import raw data and convert between all of the different formats.

VolSuite includes a powerful tool for arbitrary reslicing. It allows the user to click and drag the volume to change the slice orientation and has a number of options for changing how the slice is displayed.

When contacted about using *VolSuite* for this application, the author was very helpful and agreed that it could be modified to serve the needs of this project. However, he recommended using a newer library that he is currently developing, called *gear*, which is similar to the underlying structure of *VolSuite*, but uses a more efficient volume rendering framework. This library is discussed further in Section 2.2.2.

2.2.2 Toolkits and Libraries

The following sections describe some of the toolkits and libraries that were evaluated. These generally do not include complete executables and are primarily geared toward developers. Some of these libraries include extensive examples, but none have full-featured applications.

Visualization Toolkit (VTK)

The *Visualization Toolkit* (VTK) [18] is an open source toolkit for 3D graphics and image processing. It provides many features for designing graphical programs. VTK is widely used in the medical imaging community for various projects and is often integrated with the *Insight Toolkit* (ITK), which provides image processing functions without a graphical interface.

VTK is a powerful and well-documented toolkit, but does not directly imple-

ment any type of 2D reslicing capabilities. It would certainly be possible to develop these functions, but would take quite a bit of extra work. VTK would provide a solid foundation for graphical rendering, but would require additional development to implement some of the other features.

Medical Imaging Interaction Toolkit (MITK)

The *Medical Imaging Interaction Toolkit* (MITK) [23] is an open source project that combines VTK and the *Insight Toolkit* (ITK), along with some other features geared toward medical imaging. This would be a beneficial extension of VTK, because it combines the visualization facilities of VTK with the image processing algorithms that ITK provides, plus some extra features specific to medical imaging.

While MITK adds additional useful features beyond those of VTK, it still does not have built-in capabilities for dynamic reslicing. As is the case for VTK, much additional development work would be required.

gear

gear is an open source C++ library developed by Jason Bryan of the Ohio Supercomputer Center. This library is designed for visualization of medical images and performs functions similar to those implemented in *VolSuite*. This was recommended over *VolSuite* because it uses a recent, more efficient rendering framework [9]. In contrast to *VolSuite*, *gear* is only a library and does not include a full-featured application. It is in an advanced state, but still under active development. Although *gear* is currently not documented especially well, the author was willing to be of assistance and responded quickly through email. Pre-compiled versions of most of the dependencies were provided, along with access to the development subversion repository, where updates and bug fixes are frequently posted.

gear is an extension of OpenGL [26] graphical rendering. OpenGL [26] is a commonly used and well-supported open source graphics library that is used for a mul-

titude of computer graphics and 3D rendering applications. Further OpenGL background is provided in Section 2.3.1. The *gear* library adds in classes and functions that are aimed toward the rendering and manipulation of 3D image volumes. It is very similar to using OpenGL, but makes development for volume visualization much simpler and provides added functionality.

Included with the source code for *gear* are a helpful set of example applications. These are all very simple examples, each exhibiting only a couple of the features of *gear*, but they are quite useful. Most notably, there is an example called 'glv_VolumeRender_test5' that implements basic volume rendering capabilities. This example uses *gear's* 'glv' volume rendering library along with the OpenGL graphics libraries to render a 3D volume and allow the orientation to be changed by dragging it with the mouse. Using this example as a starting point, it would not be very difficult to develop a program to perform the desired volume rendering capabilities.

Although there are not any examples of performing 2D reslicing, this is a feature of *VolSuite* and Jason was confident that it would not be too difficult to implement using *gear*.

The following is a summary of the important features of *gear*:

- Framework for performing rendering of 3D image volumes
- Designed for medical image data
- Well-supported by the author
- Uses updated, efficient rendering techniques
- Provides useful examples

Because *gear* helps perform some of the desired functions, uses current rendering techniques and is well supported, it was chosen for the development of the software for this project. *gear* provided a strong starting point for development by simplifying

basic volume rendering. By designing the system from this set of libraries, there is minimal reliance on "black box" code that is not understood well, as most of the system is custom developed. Additionally, features are not compromises made to work with existing code; everything has been designed with the goals of this project in mind. The sections that follow provide details of the development of this system.

2.3 Volume Rendering

The most basic required function for this system is the ability to render, or display, a 3D volume. It must be able to read in ultrasound data stored as a volume and correctly draw the volume on the screen.

Based on the evaluation of different software options described in Section 2.2, the *gear* library was selected as a starting point for development. Starting with this library, basic functions were implemented and additional features were added. The following sections describe the implementation of these features.

2.3.1 Three-D Graphics Background

The *gear* library is based on OpenGL [26], which is a popular open source graphics library. OpenGL is used for a variety of graphics applications including scientific modeling and game development. While there are other graphics libraries available, the scope of the section will be constrained to 3D graphics background as it applies to OpenGL.

Before discussing the details of 3D graphics, it is necessary to introduce some terminology specific to 3D graphics applications. A series of *vertices* can be used to specify points, lines and polygons, known as *primitives*. *Primitives* are typically combined to form 3D objects, known as *models*. The process of generating an image on the computer screen from a model is known as *rendering*. This rendered image is drawn on the screen as a series of pixels.

Figure 2.3 shows a block diagram depiction of the typical OpenGL rendering pipeline. The following descriptions, presented at a very basic level, are based on information from the OpenGL Programming Guide [41]. Geometric data (vertices, lines, and objects) are handled separately from pixel data (images and bitmaps). Data of both types are stored in display lists for future use. Geometric data are put through evaluators which break everything down into vertices. These vertices are then acted on with per-vertex operations, such as applying transformation matrices, and are formed into geometric primitives. Depth-based clipping and perspective are also applied at this point. Pixel data are processed separately and either stored to texture memory or passed to the rasterization step. Rasterization is performed on both geometric and pixel data, and is a process by which the data are converted to screen pixels and loaded into the frame buffer for display on the screen.

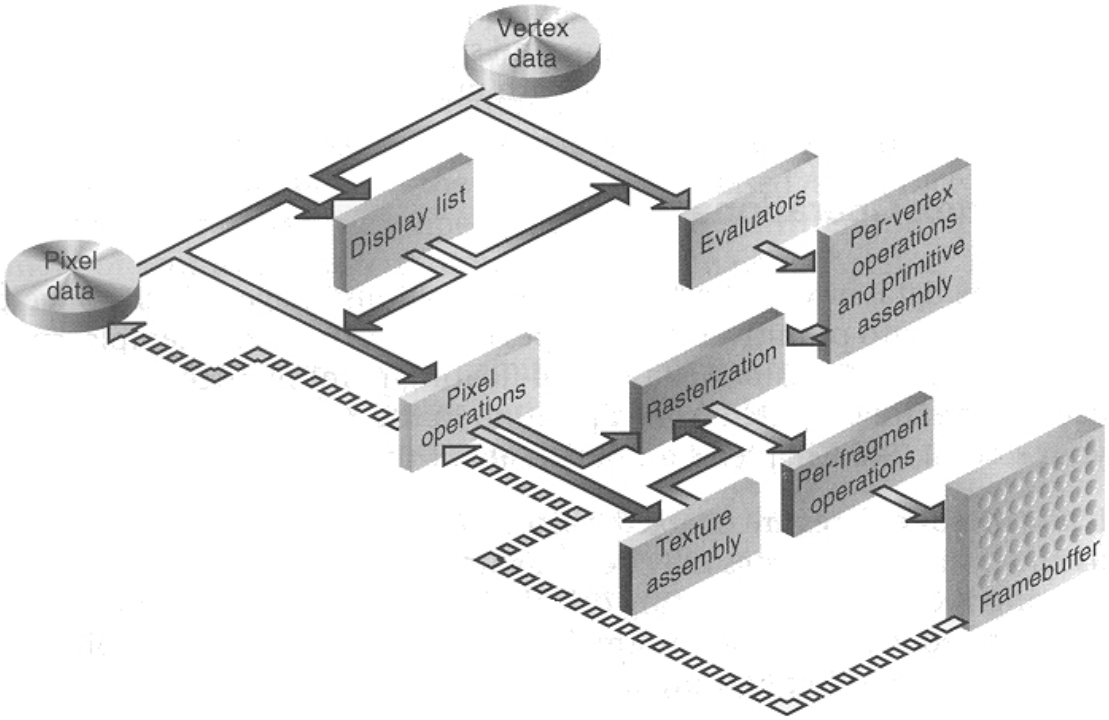


Figure 2.3: OpenGL Rendering Pipeline [41]

The main function within most OpenGL applications is the display callback. This callback function is run every time the display needs to be updated. Therefore, anything that is drawn to the screen is usually executed from this function. It is here that models are typically modified and updated. Transformations are also generally applied in this function.

OpenGL is based heavily on matrices and uses these matrices to specify transformations and constraints on the vertices to be rendered. The two primary matrices are the *modelview* matrix and the *projection* matrix. The *modelview* matrix holds any viewing and modeling transformations that have been specified and is used to modify vertices [41]. When transformations are applied, they are multiplied onto the *modelview* matrix, which is in turn applied to all objects on the screen. OpenGL has a matrix stack, which is used to store multiple versions of the *modelview* matrix, of which only the top matrix is used. This is useful when it is necessary to prevent a matrix from being applied or modified. *glPushMatrix()* can be used to push the current *modelview* matrix onto the stack and *glPopMatrix()* removes a matrix from the stack. To avoid applying the current *modelview* matrix to a given object, it can be surrounded with *glPushMatrix()* and *glPopMatrix()*, which will prevent the transformation from being applied to that specific object. Similarly, transformations specified within those two commands will operate on a new matrix and not affect the matrix that was pushed down on the stack. The *projection* matrix defines a viewable volume, which determines the range of objects that will be displayed [41].

Transformations are applied as matrices that are multiplied onto the *modelview* matrix and applied to the graphical objects. These transformations can be used to translate, rotate, or scale objects. Transformations are always applied before an object is drawn, so that they are on the *modelview* matrix when it is applied to the object at the time of rendering. Different versions of the transformation commands exist for working with a variety of data types including integer, float, and double, and are specified by the last letter in the function name. For example *glTranslated(Tx,Ty,Tz)*

and *glTranslatef(Tx,Ty,Tz)* are respectively double and float versions of the same function.

glTranslated(Tx,Ty,Tz) generates a translation with double-precision parameters, which has the matrix form shown in 2.1:

$$\left[\text{New Modelview Matrix}\right] = \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \left[\text{Current Modelview Matrix}\right] \quad (2.1)$$

glScaled(Sx,Sy,Sz) produces a scaling transformation, which has the matrix form shown in 2.2:

$$\left[\text{New Modelview Matrix}\right] = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \left[\text{Current Modelview Matrix}\right] \quad (2.2)$$

The formulation of rotation matrices, produced using *glRotated(angle,x,y,z)* is significantly more complex, as it is generalized to perform rotation of angle a about any arbitrary axis specified by x, y, and z. However, in the majority of cases, rotations are only performed about one of the Cartesian axes at a time. Any arbitrary rotation can be comprised of individual rotations in each of three orthogonal x, y, and z axes. The matrix forms of these transformations take the simple forms shown below:

(2.3) shows rotation about the x axis (1,0,0):

$$\left[\text{New Modelview Matrix}\right] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) & 0 \\ 0 & \sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \left[\text{Current Modelview Matrix}\right] \tag{2.3}$$

(2.4) shows rotation about the y axis (0,1,0):

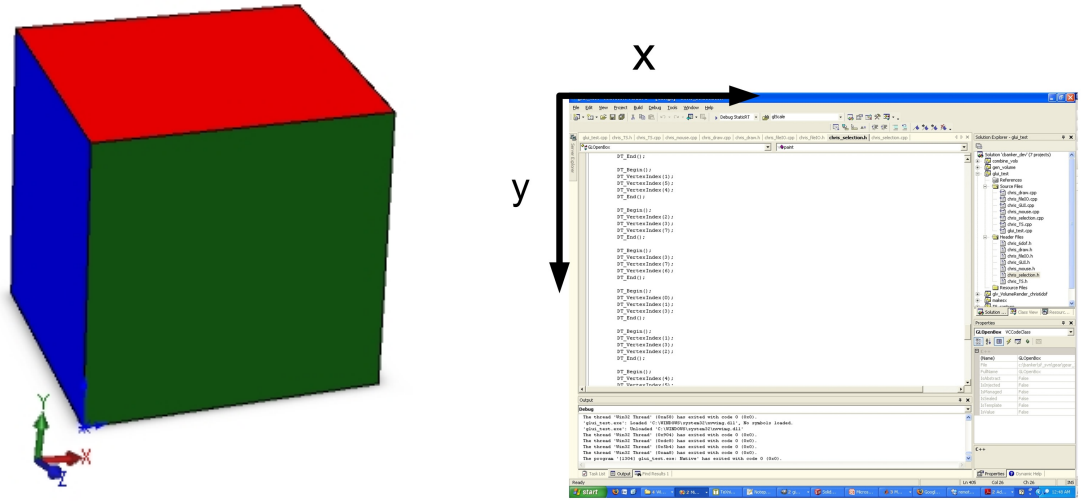
$$\left[\text{New Modelview Matrix}\right] = \begin{bmatrix} \cos(a) & 0 & \sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \left[\text{Current Modelview Matrix}\right] \tag{2.4}$$

(2.5) shows rotation about the z axis (0,0,1):

$$\left[\text{New Modelview Matrix}\right] = \begin{bmatrix} \cos(a) & -\sin(a) & 0 & 0 \\ \sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \left[\text{Current Modelview Matrix}\right] \tag{2.5}$$

OpenGL uses a "right-hand" coordinate system, which is different from the typical coordinate system used for the 2D display on computers. The terms "right-hand" and "left-hand" refer to the left- and right-hand rules that can be used to define a coordinate system's orthogonal axes. Figure 2.4 compares the two different coordinate systems. Figure 2.4(a) shows a right-hand coordinate system like that used in OpenGL. Figure 2.4(b) shows a Windows screen capture with added coordinate axes. It can be seen that the y-axis is inverted between the two different coordinate

systems, which in turn also means that the *z-axis* is inverted. In practical terms, this means that the coordinates of the window and mouse differ from those used for OpenGL rendering, which must be taken into consideration.



(a) OpenGL "Right-hand" Coordinate System

(b) Windows "Left-hand" Coordinate System

Figure 2.4: Comparison of OpenGL and Windows Coordinate Systems

OpenGL is designed solely as a rendering language and purposely does not implement any functions for working with windows or reading events from input devices, so as to be independent of any operating system or window manager [41]. Therefore, OpenGL is frequently used with the *OpenGL Utility Toolkit*, better known as GLUT. GLUT handles platform-dependent operations such as window management, display, and input devices. GLUT is used by setting up some basic parameters and then registering a series of callbacks. These callbacks are references to functions, which will be called when a certain event occurs. For example, when the mouse is clicked, the mouse callback function is called. Similarly there are callbacks for the keyboard, mouse motion, timers, and many other basic functions. After setting up all the callbacks, the main function enters an infinite loop, where the program waits

for callbacks and executes any graphical rendering that it has been programmed to perform.

2.3.2 Basic 3D Volume Display

At the start of software development, some initial design considerations were taken into account. These included selection of a development environment, linking method and source control. These software design considerations are discussed in Appendix A.

The first step to creating this system was to implement basic 3D volume display functionality with the ability to change the orientation of the volume. This was created by modifying a volume rendering example to work for the purposes of this application. A few example projects were provided with the *gear* source code, including 'glv_VolumeRender_test5', which is a volume rendering example made to work specifically with stored 3D image volumes. This example sets up the basic OpenGL graphics interface, including using GLUT to create windows and initiate mouse and keyboard callbacks.

Minimal modification was required to make this example work for the needs of this application at a basic level. Ultrasound data sets were loaded and could be properly displayed and reoriented. Figure 2.5 shows a screen capture of the system performing basic rendering of a 3D ultrasound volume.

The image material in Figure 2.5 was made available to us courtesy of Dr. Aaron Fenster of the Robarts Institute in Canada without any identifying attributes [1]. This image material is presented later in this chapter in Figures 2.7 and 2.8. This image volume is a scan of a prostate gland, which was scaled for use with the training system. While the size and position of the image data are unrealistic, this provided

[1]The Use of Human Subjects at The University of Western Ontario - Ethics Approval Notice #12682E, "Comparison of 3D transrectal ultrasound (TRUS) to conventional 2D TRUS in the measurement of prostate volume". The Principle Investigator for this approval was Dr. C. Romagnoli (Radiologist).

some initial data for testing of the training system.

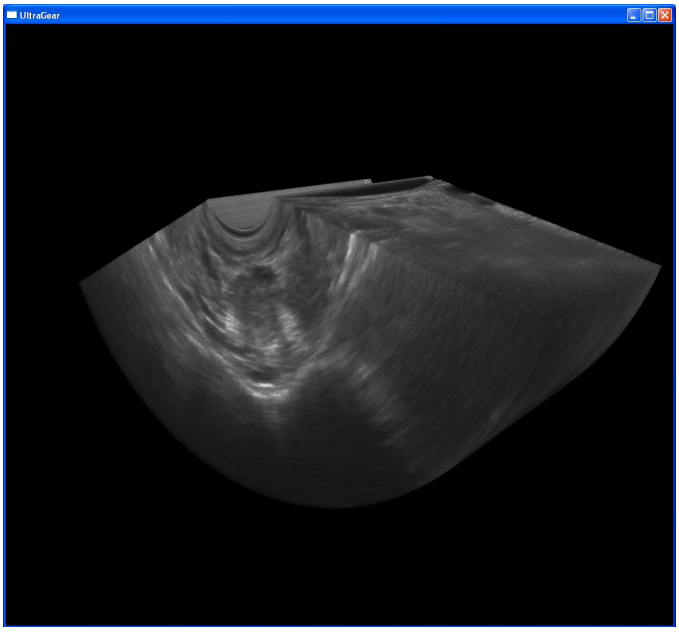


Figure 2.5: Screen Capture of 3D Rendering

One issue that arose was that the rendering speed was far too slow, updating at a rate of less than once per second when run on a desktop PC with a 2.0 GHz dual-core Xeon processor, 4GB of memory and a NVIDIA Quadro FX 550 graphics card. This performance is clearly unacceptable for a system that should provide smooth, immediate updates.

The author of *gear* was consulted and he recognized the problem and recommended a change to the code to improve the frame rate [9]. The problem arises from the fact that, by default, *gear* tessellates the volume into a enough texture bricks to exactly represent the volume, which ends up being a very large number of bricks. The number of bricks that are generated is a function of the size of the volume in voxels and how it is shaped. For large volumes or those with dimensions that are far from cubic, the large number of bricks formed can cause rendering to run slowly. This slowdown occurs as a result of the fact that the renderer must perform a clipping

operation for each texture brick boundary. Most of the ultrasound volumes that will be used with this system are both large and non-cubic, which can be problematic.

To fix this issue, it was suggested that the following line of code be added to set the maximum number of bricks formed by the texture shader to 4:

shader->setTesselateFunc(TextureBrickSet::tesselateFewestBricks, (void*)4);

This simple change involved telling *gear* to render at most four texture bricks, which saves on processing with no discernible difference in quality. Figure 2.6 shows a comparison of the rendered volume with and without this fix applied. It can be seen that there is little or no visible difference.

(a) Training System Software Rendering With Tessellation Fix

(b) Training System Software Rendering Without Fix

Figure 2.6: Comparison of Volume Rendering With and Without Tessellation Fix

The only drawback to this fix that the author warned about is that slightly more memory is required for the smaller number of larger texture bricks. In the two examples shown in Figure 2.6, the version without the fix requires 192 MB of RAM, while the version with the fix requires 214 MB, a difference of less than 11.5%. This amount of memory is well within the amount of memory typically found in current computer systems.

This modification made an immense difference in performance, increasing the frame rate to a range of approximately 25-35 frames per second (FPS) on the same

system. The immense improvement in frame rate far outweighs the small increase in memory usage. This frame rate is similar to that of a typical TV signal, broadcast at a rate of 30 FPS, which is generally enough for low frequency movement based on physical user inputs to appear smooth on the computer monitor. These frame rate limitations are only constrained by processing power, so a faster processor and graphics card could certainly achieve higher frame rates if desired.

2.3.3 Arbitrary Reslice Capabilities

The software must be capable of generating a 2D slice from the 3D image volume in real time. To accomplish this, the depth display range is set to a very thin range, so that a 2D slice with negligible thickness is displayed. The specific range that is used is -1.0 to -0.95, which are unitless values represented in OpenGL coordinates. This range is located at the center of the volume and is thin enough to look like a single slice without looking transparent. Additionally, an orthographic view is used, rather than a perspective view. In orthographic mode, no perspective transformations are applied, which prevents the image from looking stretched or deformed at oblique views.

The arbitrary reslice is generated in reference to the position and orientation of the sham transducer. Through the use of a 6 DoF tracking system, position and orientation values reflecting the current location and pose of the sham transducer are generated. Figure 2.7 shows the relationship between the position and orientation of the sham transducer, the transformed volume and the 2D scan image.

By translating and rotating the volume being displayed, the user's view is changed appropriately. Moving the volume instead of the slice being displayed keeps the view constant, which emulates what would be seen on an ultrasound scanner. Since the entire volume is being moved rather than the view, the movements must be the opposite of the input, as shown in Figure 2.7. For example, an input translation to the right should result in the volume being shifted to the left to correctly emulate an

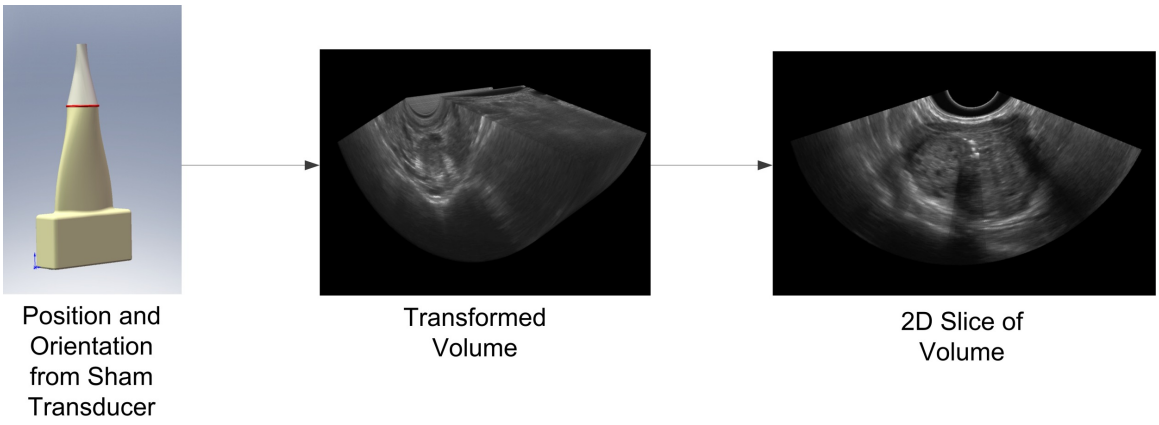


Figure 2.7: Relation Between Sham Transducer and Generated 2D Slice

ultrasound system. Figure 2.8 shows a screen capture of the system performing 2D reslicing.

2.4 Interactive Control

Interactive control is the key feature of this system, as it provides changes to the displayed image in accordance with the transducer movement. For initial testing, a 3DConnexion *SpaceNavigator* 6 DoF input device was used. This provided an immediate means of using the system with 6 DoF input, without the need to purchase an expensive magnetic tracking system. Further along in the design, an Ascension Technology Corporation *Flock of Birds* system [16] was purchased and integrated into the system and later an Ascension Technology Corporation *trakSTAR* [4] was used for its smaller size.

Regardless of the input device, the general method of interactive control remains the same. This general control method is shown in Figure 2.9. Coordinates are continuously read for all six axes of the input device or tracking system. This position and orientation information is scaled appropriately and converted to produce a transformation matrix that can be applied to the volume being rendered. Through this

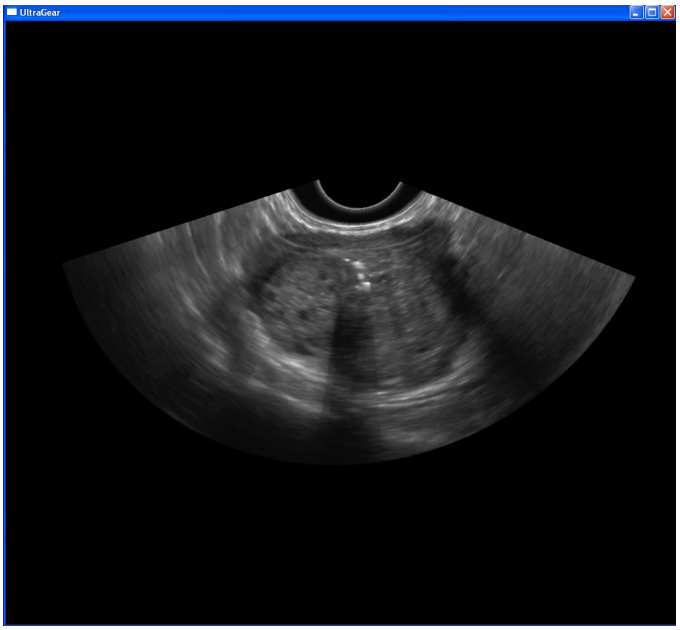


Figure 2.8: Screen Capture of 2D Reslicing

process, the movement of the input device controls the orientation of the scan image.

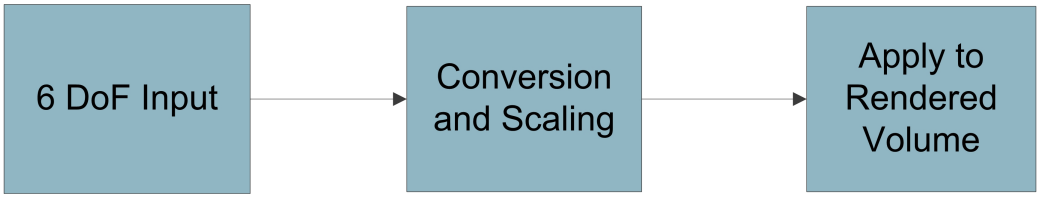


Figure 2.9: Generalized Interactive Control Block Diagram

2.4.1 Control Using SpaceNavigator 6 DoF Input Device

3DConnexion's *SpaceNavigator* [1] is an innovative new input device that provides a full six degrees of freedom. It is in the form of a "hockey puck" shaped device, as shown in Figure 2.10, with a cap that can be translated and rotated in all 6 DoF. The device connects to the computer through a USB port and includes two programmable buttons on its sides. At only $60, it is very economical and has become widely used

for CAD and 3D drawing applications. A support community exists for this device on the 3DConnexion website, and a software development kit is provided for those who wish to design their own drivers for currently unsupported applications.

Figure 2.10: 3DConnexion *SpaceNavigator* Input Device [1]

One difficulty with using the *SpaceNavigator* is that it requires application-specific drivers to be able to work with any program. Although a software development kit is provided, it is still a significant task to design custom drivers for it. A custom driver would be the preferable way of implementing this device, but it was only used for initial testing, so less time consuming options were more desirable.

To avoid the need for an application-specific driver, a user-submitted driver available on the development discussion boards was used to generate a joystick input stream from the *SpaceNavigator*. This driver, called *RBC9-SpaceNav*, permits the *SpaceNavigator* to act as a standard mouse, joystick, or keyboard and provides options to program how each of the axes and buttons are interpreted [34]. Implementing a joystick input in the simulation software is significantly easier than designing a cus-

tom driver.

Figure 2.11 shows a block diagram of how the *SpaceNavigator* is used to interactively control the generation of 2D slices. In the *RBC9-SpaceNav* driver settings, the device is configured to act as a joystick with the 6 axes of the device configured as the 6 joystick axes. The buttons are configured as joystick buttons 1 and 2.

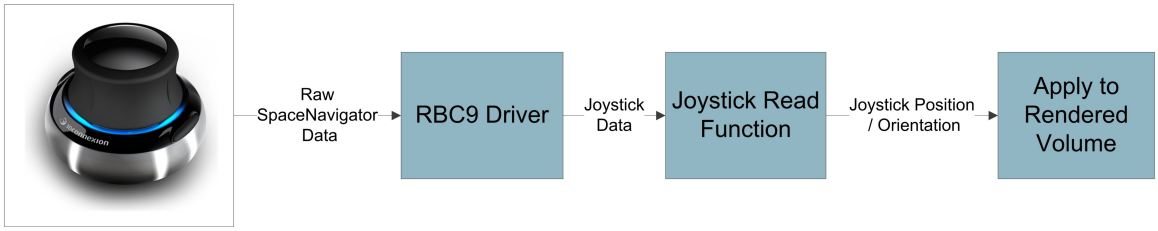


Figure 2.11: Block Diagram of *SpaceNavigator* Communication and Control

The joystick read capability is implemented using a polling method. An OpenGL timer is set to call a joystick read function once every 50 ms. This function, called *joyGetPosEx*, is an extended joystick read function within the Windows device library. It returns the values for all six joystick axes, along with the button states.

The *gear* library links input devices to the displayed volume using an object called a 'TrackballManipulator'. This object takes input motions from a device such as a mouse or joystick and uses them to determine a transformation matrix to apply to the volume being displayed. This method is primarily designed for use with a mouse where there are only two axes, and different pairs of axes are selected by clicking a different mouse button while dragging. However, multiple trackball manipulators can be used and linked to separate pairs of axes.

Figure 2.12 shows a diagram of how trackball manipulators are used to translate 6 DoF inputs to OpenGL transformations. By using three different manipulators simultaneously, all six axes can be represented. A trackball object is set up for each of the x, y, and z axes, each of which handle the axis' translation and the rotation

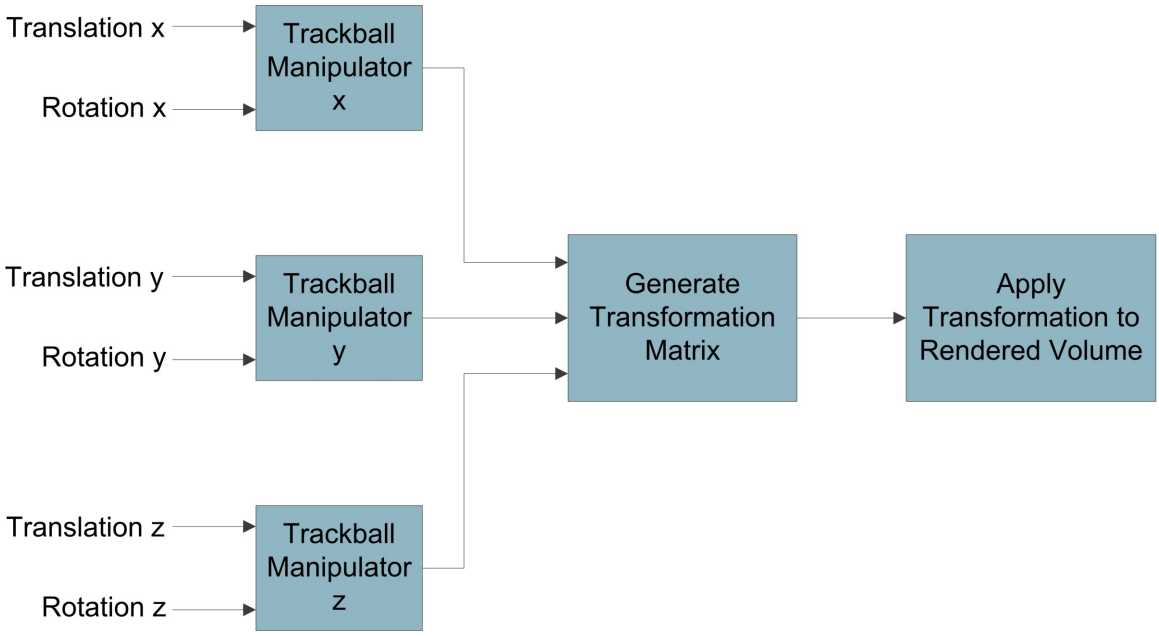


Figure 2.12: Diagram of 'TrackballManipulator' Operation

about that axis. The trackball manipulator class has a member function that can generate a matrix from the most recently received data. In the main display function, the transformation matrices generated by the 'TrackballManipulator' are applied to the volume to appropriately follow the movement of the sensor.

To avoid causing the volume to suddenly move when the program is first started, a flag is used to indicate whether or not it is the first run of the joystick function. On the first run, the initial position is read and used as a reference, which is called 'grab' for the trackball object. There is also a 'release' action for the trackball that performs the opposite function of 'grab'. On subsequent calls, the current position is updated and the volume is moved accordingly.

The two buttons on the *SpaceNavigator* were implemented to improve navigation. Initially, different views could only be achieved while the *SpaceNavigator* was held in place. To improve upon this, the left button is used to 'lock' the position at a given view, setting that view as the new center point. The right button is used as

a reset button to reset back to the default view. This use of the buttons permits easier navigation of a volume. For example, a user can rotate to a side view, 'lock' the view, and then easily scan through the volume sideways, without having to hold the rotation.

While the *SpaceNavigator* was adequate for initial testing, as an input device it can only provide relative movement and does not have the ability to track absolute position and orientation. Although the *SpaceNavigator* is not the final input device, it was an important tool for design and testing and provides the option to navigate ultrasound volumes without the need for a tracking system.

2.4.2 Control Using Ascension Technology Corporation *Flock of Birds* 6 DoF Tracking System

The software must be able to correctly display slices of the 3D volume based on the position of the sham transducer, which requires absolute position and orientation information in reference to a fixed point. To do this, it is necessary to use a 6 DoF tracking system rather than an input device.

To provide this absolute position information, an Ascension Technology Corporation *Flock of Birds* system [16] was used. The *Flock of Birds* is a DC magnetic 6 DoF position and rotation tracking system. Figure 2.13 shows the *Flock of Birds* system.

This system uses a pulsed DC magnetic field signal, which is more immune to the effects of nearby metal than systems based on AC magnetic signals. Further comparison of AC and DC magnetic tracking systems is presented in Appendix E. Figure 2.14 shows a functional block diagram of the *Flock of Birds* system. It has a magnetic transmitter that remains in a fixed location and emits pulsed magnetic signals from three orthogonal transmitter coils [6]. A small, 25.4 x 25.4 x 20.3 mm receiver detects these signals using three receiving coils. Some signal processing is performed on the sensor signals and then the system's embedded computer processes

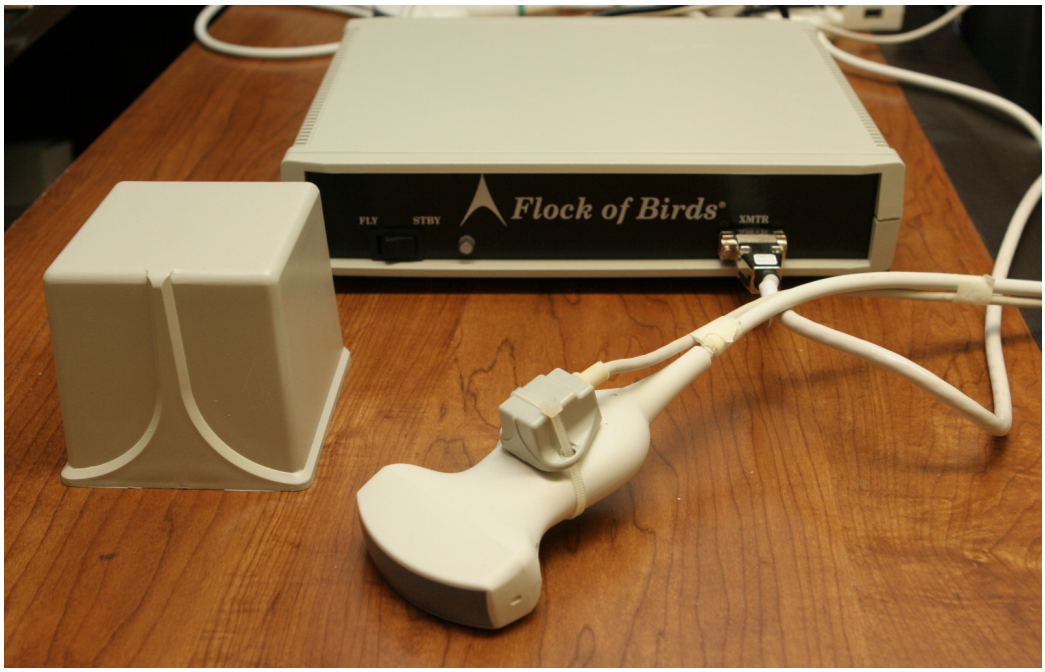


Figure 2.13: Ascension Technology Corporation *Flock of Birds* System [16]

the signals to determine the position and orientation of the sensor in relation to the transmitter, which is output to the computer as a serial stream.

Figure 2.15 shows a signal diagram for a typical measurement cycle performed by the *Flock of Birds*. At the start of the measurement cycle, the sensors take a reading of Earth's ambient magnetic field so that it may be subtracted out of the measured signals received from the transmitter [6]. During each measurement cycle, the transmitter pulses each of its three coils, one at a time. For each of these pulses, the signals at the three receiver coils are read by the control unit. The signals are not sampled immediately during each time segment to allow time for magnetic flux in nearby metals to settle [6]. The received signals are processed by the control unit, which uses a microprocessor to perform the position and orientation calculations. The output of the system is a stream of serial data containing the position and orientation coordinates. Further details on the specifications of the *Flock of Birds* can be found in the datasheet in Appendix B.

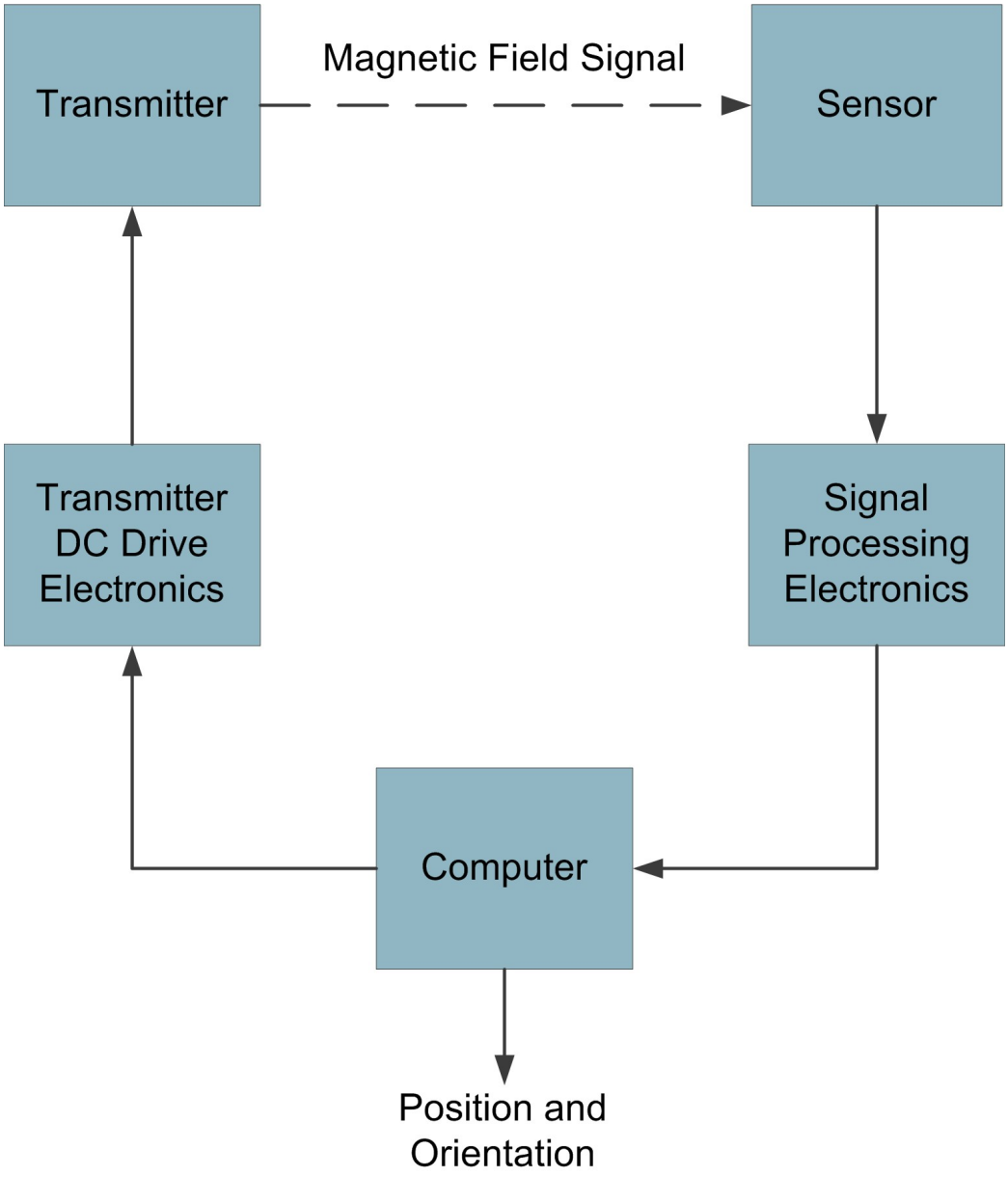


Figure 2.14: Ascension Technology Corporation *Flock of Birds* System Functional Block Diagram [6]

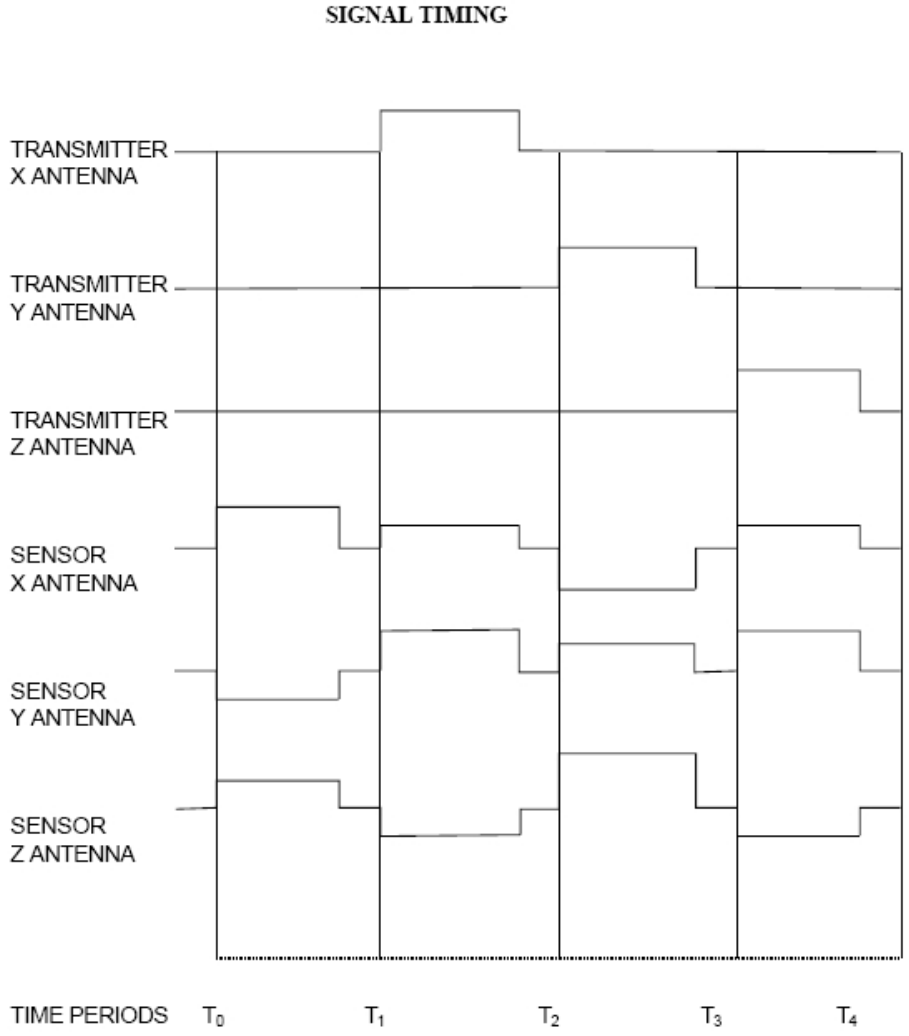


Figure 2.15: Ascension Technology Corporation *Flock of Birds* Signal Diagram [6]

Using the typical mid-range transmitter, the *Flock of Birds* is designed to operate in a ± 0.9 m range with static translational and rotational RMS accuracies of 1.8 mm and 0.5° within this range, respectively [16]. Precision for the *Flock of Birds* is specified as 0.5 mm and 0.1° at a distance of 0.3 m between the transmitter and tracking sensor [16]. Discussion of precision versus accuracy is presented in Appendix D.

Figure 2.16 shows a block diagram describing how the *Flock of Birds* is used to control the reslicing.

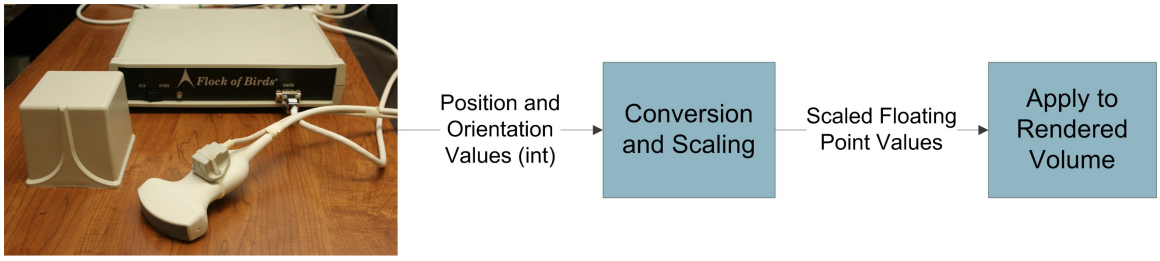


Figure 2.16: Block Diagram of *Flock of Birds* Communication and Control

The *Flock of Birds* was implemented by setting up two-way serial communication between the computer and the tracking system. To match the *Flock of Birds'* default communication configuration, the serial port is opened at 115,200 baud using 8 bit data words, no parity and 1 stop bit.

The *Flock of Birds* uses a simple query-and-response style of communication. ASCII characters are sent to the tracking system and values are immediately sent back. Values are read using the 'position/angles' mode, which consists of three translations and three rotation values, each expressed as two bytes. The rotation values are expressed in Euler angles. These messages can be requested by transmitting an ASCII 'B'. In response, the *Flock of Birds* sends back the 12 data bytes. After transmitting the request, the software waits to receive the 12 byte response from the tracking system, which is stored into a buffer for processing.

As shown in Figure 2.16, these are unscaled integer values which require some additional processing. To convert the received data into usable information, the two bytes for each value must be combined to form 16-bit integers and then scaled to floating point values representing inches and degrees. After the conversion process, the values are ready to be used to transform the volume. Equations (2.6) to (2.11) show how the two data bytes are combined into usable values for positions and angles, respectively. The array *data* is the 12-byte array of values obtained from the *Flock of Birds*. The position and orientation values are provided as an 8-bit high byte and a 7-bit low byte. The first bit of the low byte is 1 for the first value and 0 for each byte thereafter. Therefore, the first byte must be masked to remove the one. The bytes are appropriately shifted and then added together to obtain a full-scale value with a range of -32768 to 32767.

$$x = (data[1] * 128 + data[0] - 128) * 4 \quad (2.6)$$

$$y = (data[3] * 128 + data[2]) * 4 \quad (2.7)$$

$$z = (data[5] * 128 + data[4]) * 4 \quad (2.8)$$

$$rz = (data[7] * 128 + data[6]) * 4 \quad (2.9)$$

$$ry = (data[9] * 128 + data[8]) * 4 \quad (2.10)$$

$$rx = (data[11] * 128 + data[10]) * 4 \quad (2.11)$$

To obtain proper values for the graphical transformations, these full-scale values must be converted into usable formats. A floating point value is obtained by converting the value to floating point and then dividing by 32768 to create a scaling factor in the ± 1.0 range. For position, this number is multiplied by 36 to get a result in inches and for angles it is multiplied by 180 for a result in degrees.

Transformations are performed using *gear*'s 'TrackballManipulator' in a similar manner to the *SpaceNavigator* implementation, described in Section 2.4.1. The track-

ball object is configured in the same manner and its axes are linked to the values provided by the *Flock of Birds*.

It is necessary to calibrate the location of the sham transducer at startup and when changing probes. To do this, the probe is held vertical, centered on the belly button of the manikin. The position can be calibrated either by clicking the 'Re-center Probe' button on the GUI or by pressing the space bar while no other buttons are highlighted. This triggers a function that resets the zero location for the position system to the point at which the probe is being held.

The command that is used for this is called *boresight* and is activated by sending an ASCII 'u' to the *Flock of Birds*. This command tells the system to consider the current position to be position (0,0,0) and from then until the end of the training session or the next reset, this is used as the reference point.

2.4.3 Control Using Ascension Technology Corporation *trakSTAR* 6 DoF Tracking System

The *Flock of Birds* provides effective, accurate 6 DoF tracking, but the sensor is too large to be adequately concealed in a standard transducer shell and the transmitter is also rather large and heavy. To remedy these concerns, an Ascension Technology Corporation *trakSTAR* [4] system was purchased. This system is a recent design from Ascension that improves upon their previous *MiniBIRD* system.

The *trakSTAR* system provides similar functionality to that of the *Flock of Birds*, but permits the use of small 8 mm sensors and the compact short-range transmitter, while still providing an adequate tracking range of ±46 cm. Figure 2.17 shows the components of the *trakSTAR* system along with a comparison of the different transmitters. The short-range transmitter measures 6.3 x 4.6 x 5.2 cm. The 8 mm sensor is the largest of the three shown in Figure 2.17(b) and measures only 8 x 8 x 20 mm. The smaller size of the transmitter and sensors permit them to be easily

embedded into a manikin and sham transducers, to remove awareness of the tracking system. Full specifications for the *trakSTAR* system can be seen in Appendix C, which contains the complete datasheet.

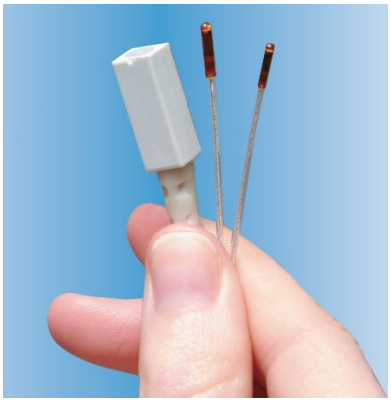
The *trakSTAR* provides slightly better accuracy than the *Flock of Birds* with static translational and rotation RMS accuracies of 1.4 mm and 0.5° within the 46 cm tracking range [4]. The precision of the *trakSTAR* system is the same as the *Flock of Birds* at 0.5 mm and 0.1° [4]. Discussion of precision versus accuracy is presented in Appendix D. The *trakSTAR* works on the same principle of operation as the *Flock of Birds*, which was described in Section 2.4.2.

In addition to permitting the use of smaller components, the *trakSTAR* uses improved connectors, allows up to four sensors to be tracked simultaneously and provides an updated USB interface. Although it is backwards compatible with the *Flock of Birds* when connected through the serial port, using the USB connection permits the use of the Ascension Technology Corporation 3D Guidance API (Application Programming Interface). This API includes functions to control the tracking system and read data from it in a simple and effective manner. The API provides modular, high-level access to the functions of the device and greatly simplifies the code. This avoids the need to manually process values, as floating point data can be read directly.

Switching to the *trakSTAR* was relatively straightforward because communication is quite similar to the *Flock of Birds*, as shown in Figure 2.18. Initially, functionality was verified by using the serial interface and the original *Flock of Birds* code. After basic testing, the software was modified to utilize the USB interface and the Ascension API. For each *Flock of Birds* function, an equivalent function was written that takes advantage of the API. These new functions were created as a separate module, which makes switching back to a *Flock of Birds* as simple as swapping one file.

Initially, the *trakSTAR* was implemented using the same 'TrackballManipulator' approach as the *SpaceNavigator* and *Flock of Birds*; however, as development progressed, this interface started to become a hindrance. Without direct access to the

(a) *trakSTAR* Control Unit



(b) *trakSTAR* Sensors

(c) Ascension Transmitter Comparison: Mid-range (left) and Short-range (right)

Figure 2.17: Ascension Technology Corporation *trakSTAR* System

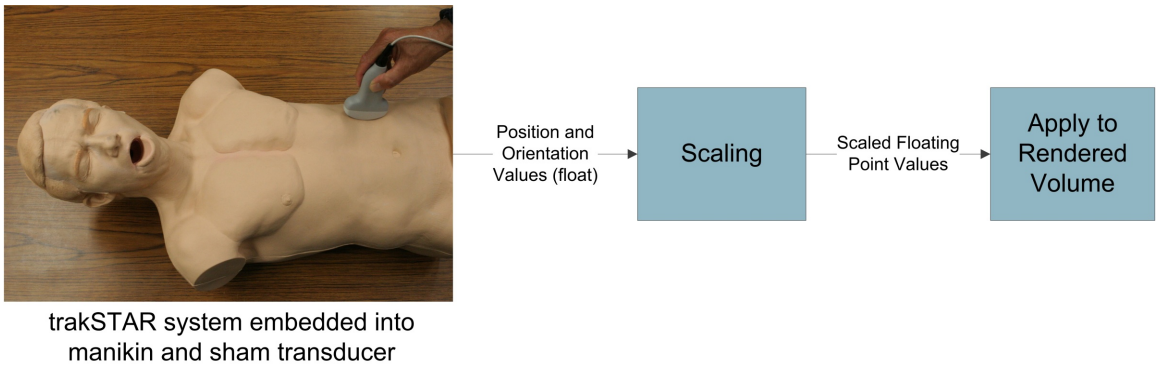


Figure 2.18: Block Diagram of *trakSTAR* Communication and Control

position and orientation values and complete control over the transformations, it was difficult to implement features efficiently. As none of the 'grab' and 'release' functions are needed for the tracking systems as they were for the *SpaceNavigator*, the trackball method is unnecessary and simply extra overhead. To rectify this, an improved method of data transfer was designed.

Figure 2.19 depicts the improved read and transformation process used for the *trakSTAR*. Instead of passing values to the 'TrackballManipulator' to generate a transformation matrix, the values are taken directly into a read function and stored into an array for use by the transformation function. The *trakSTAR* read function reads the position and orientation values from the *trakSTAR* as double-precision floating point values using the Ascension API.

The transformation function is called every time the displayed scan image is updated to apply the appropriate transformation. This function takes the array of values stored by the read function and performs the necessary transformation with the functions *glTranslated* and *glRotated*, which perform translations and rotation, respectively. For reusability, the transformation function also takes an array of unitless scaling factors corresponding to each of the translations and rotations. These scaling factors are multiplied by the output values of the *trakSTAR*. The primary purpose of these scaling factors is to invert axes, so they are generally set to 1.0 or

-1.0, but could be set to any value if there were a need to scale the inputs. These scaling factors are used for the navigational display discussed in Section 3.4.6, which uses the same transformation function as the main volume, but with all of the axes inverted to properly follow the movement of the transducer.

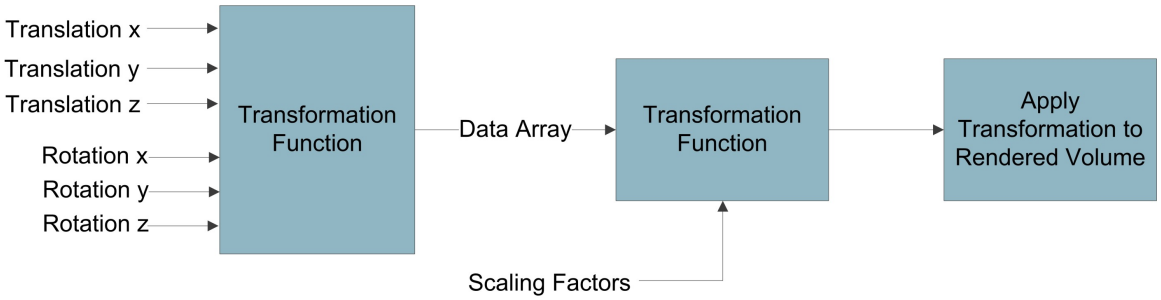


Figure 2.19: Block Diagram of *trakSTAR* Read Function

Like the *Flock of Birds*, the *trakSTAR* system also needs to be calibrated to the surface of the manikin. The action taken by the user is exactly the same, but the underlying process is slightly different. This stems from the fact that Ascension Technology Corporation did not incorporate the *boresight* command, described in Section 2.4.2, into their API. Therefore, this action must be performed within the code. A set of three variables are used to keep track of the x, y, and z coordinates of the center point, which are subsequently subtracted from each set of new position values.

The *trakSTAR* transmitter and receiver have the same coordinate system when they have their axes aligned, as shown in Figure 2.20. Such an alignment would produce angles of (0°, 0°, 0°).

Because the receivers are aligned vertically in the sham transducers, as shown in Figure 2.21, it is necessary to realign the angles. Therefore, a sham transducer held perpendicularly will always have rotations of (0°, -90°, 0°). On initialization, this alignment is set using the 'angle_align' command.

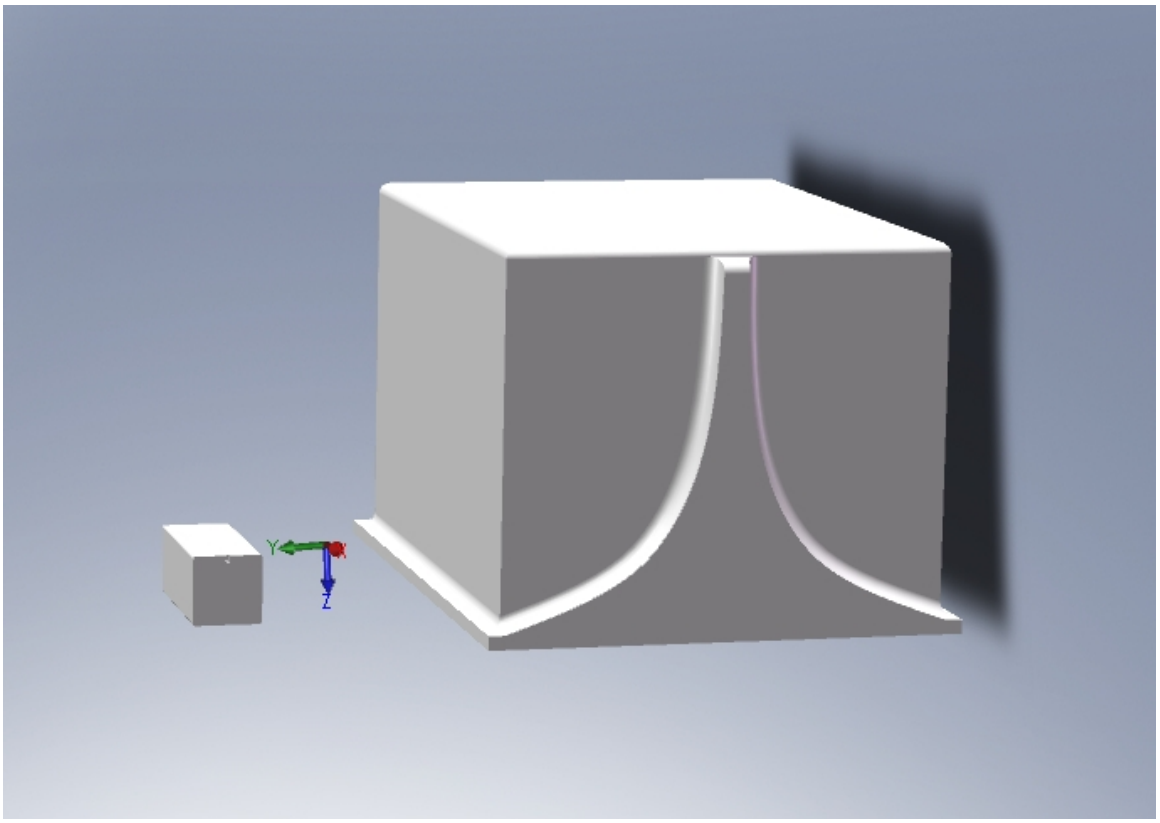


Figure 2.20: *trakSTAR* Transmitter and Receiver with Their Coordinate Axes Aligned

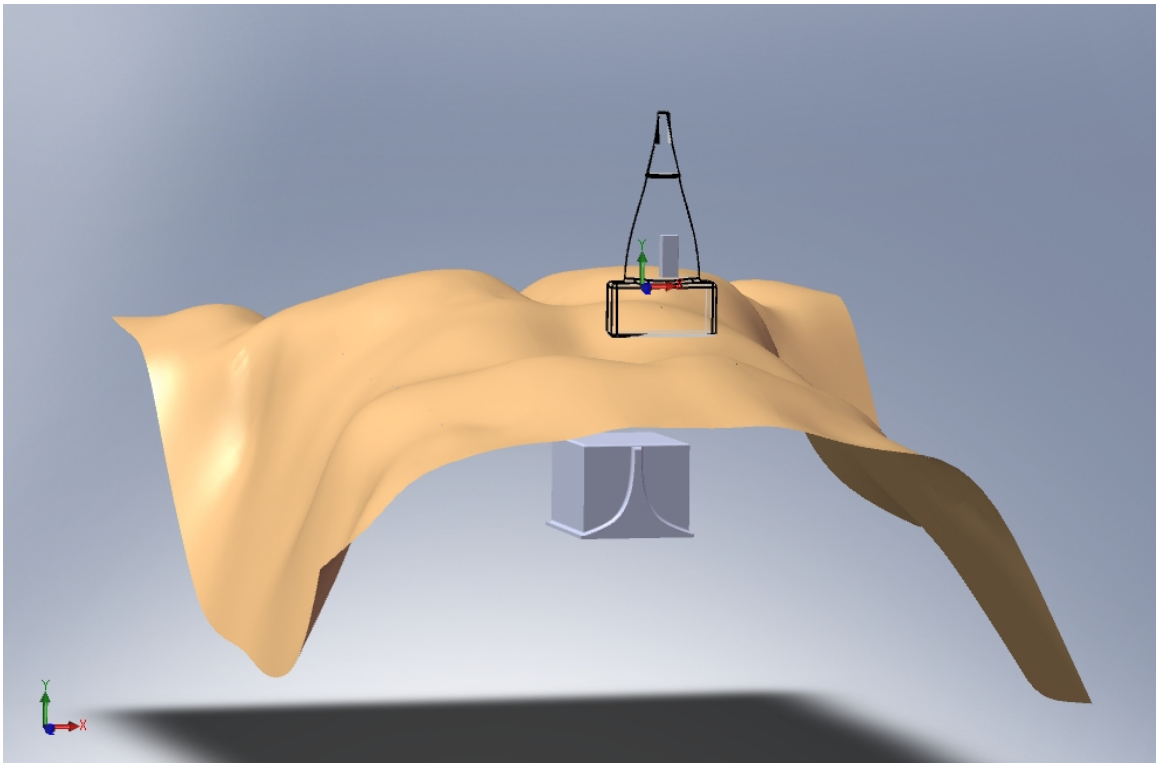


Figure 2.21: *trakSTAR* Transmitter and Receiver as Used With Manikin and Sham Transducer

2.4.4 *trakSTAR* Noise Reduction

The *trakSTAR* system is a very compact system with small, lightweight transmitters and receivers. However, the size reduction comes at the cost of its accurate tracking range. Even though the device is being operated within the rated range of 18", the precision was initially much lower than the *Flock of Birds* system, because the *Flock of Birds* was being operated within a relatively smaller region in the center its operation range. In the training system this noise manifests itself as random Gaussian movement and jitter of the image while the sham transducer is held still. To remediate this, some tests were designed to track down the sources of noise and reduce or eliminate them.

Mounting Transmitter Closer to Scanning Surface

The tracking system has the least noise when the transmitter and receiver are in close proximity. Therefore, it is beneficial to position the transmitter as close to the surface of the manikin as possible. This was done by mounting the transmitter on a rigid strut within the manikin, placing it very close to the scanning surface. This strut configuration is shown in Figure 2.22 and described in further detail in Section 3.2.1. This showed a significant improvement over the transmitter being mounted directly to a board beneath the manikin, but was still unable to match the performance of the *Flock of Birds* system.

Selection of Measurement Rate

The DC magnetic tracking used by Ascension Technology Corporation's magnetic tracking systems works by sending a DC pulse and then waiting for a period of time before reading the sensor's coils. This permits time for any magnetic flux to settle in nearby metallic objects, which is the reason DC magnetic tracking is relatively immune to nearby metal. The measurement rate is an adjustable system parameter,

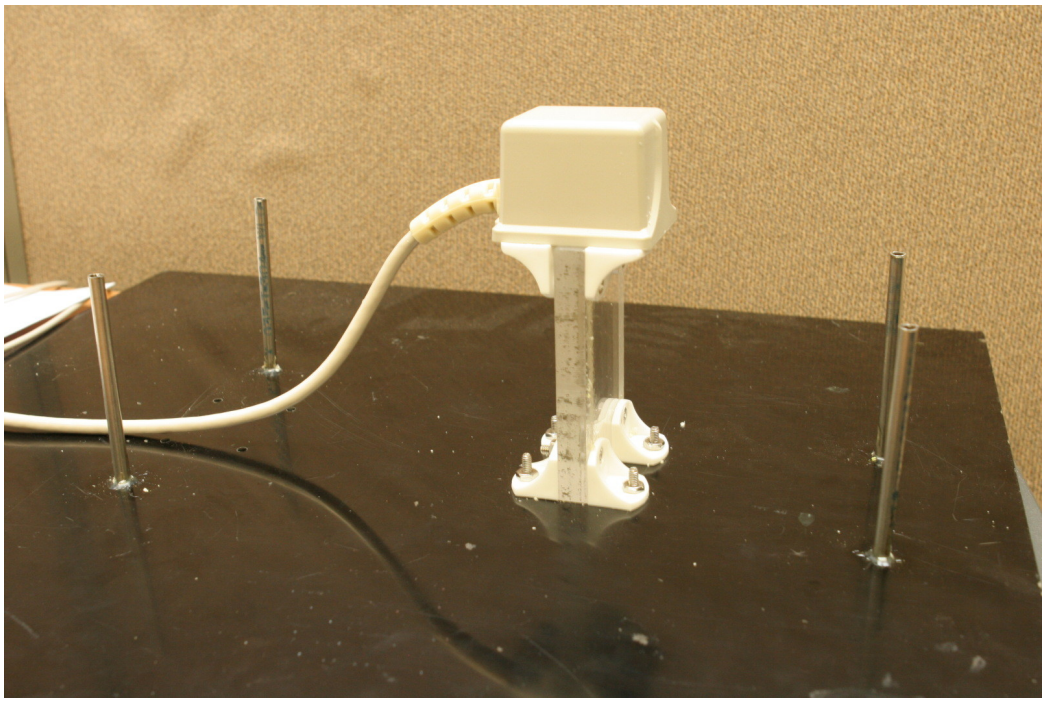


Figure 2.22: Strut For Mounting *trakSTAR* Transmitter Close to Scanning Surface

specified in Hz, that determines how long this wait time is.

Selection of an appropriate measurement rate can significantly improve system precision, by avoiding environmental interference. To facilitate selection of an appropriate measurement rate, a C++ program called *ts_noisex* was written to determine an optimal measurement rate. This program is described in further detail in Appendix K. The noise measurements are based on the *trakSTAR*'s 'quality' parameter. The 'quality' parameter is a measure of the degree to which the position and orientation values are in error and is indicative of the amount of noise. This parameter has a range of 0 to 32768, with the optimal value being 0. By comparing the noise at different measurement rates, it is possible to select an optimal rate.

Figure 2.23 shows a plot generated using data from this utility. It can be seen that there are two ranges where the 'quality' parameter is especially high, and then smaller variations of noise throughout the range. The high 'quality' value at low

measurement rates is due to the system's built-in high-pass filters, which remove noise near the 50-60 Hz AC wall power frequencies. The spike around 120 Hz is due to the second harmonic of the 60 Hz AC power used in the United States. Based on the captured data, the measurement rate is set at 115.25 Hz, which was determined to be an optimal rate for this system in its current environment. This measurement rate is defined as a parameter called 'MEAS_RATE' in the software and is set by issuing a command to the *trakSTAR* system.

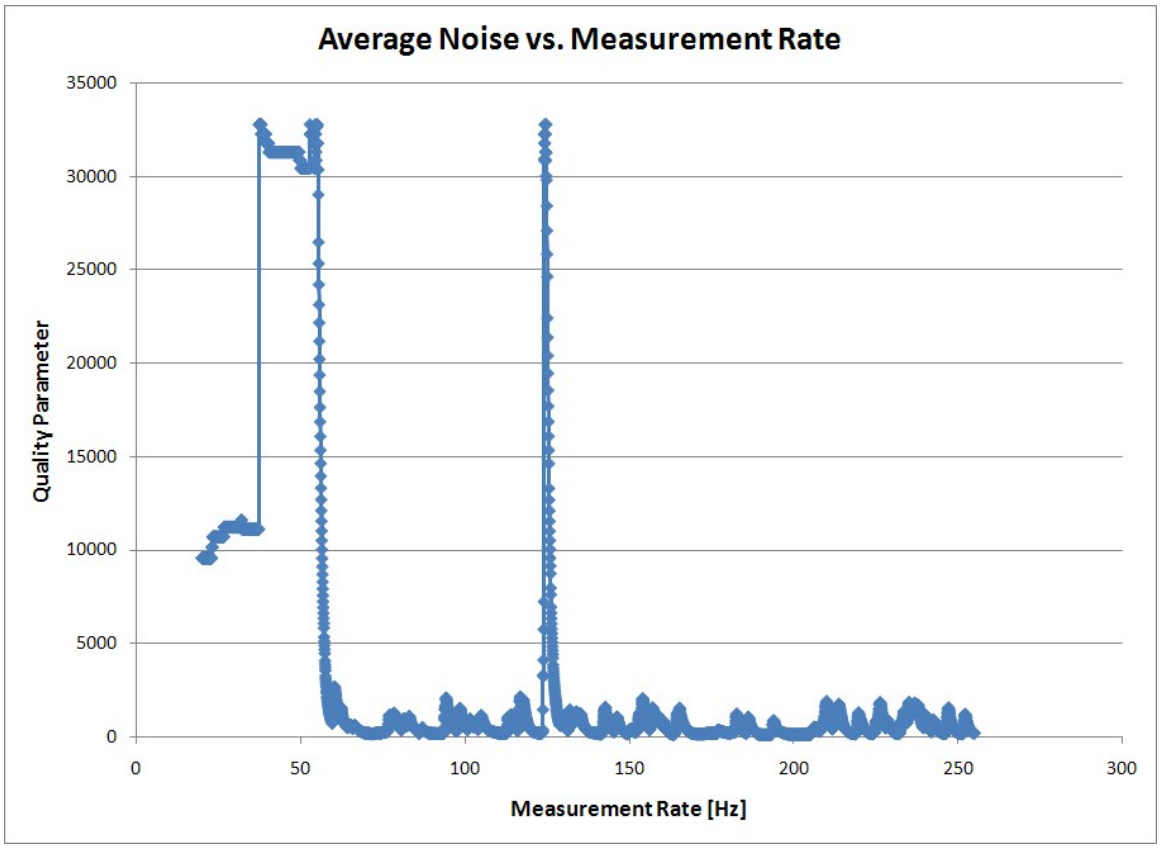


Figure 2.23: *trakSTAR* Noise Plot Showing the Quality Parameter as a Function of Measurement Rate

The measurement rate is sensitive to the presence of metal in the operating environment and should be adjusted accordingly. The described tests were performed in an environment relatively free of nearby metallic structures. For optimal performance,

noise measurements should be performed to determine an appropriate measurement rate each time the system is moved to a new environment.

Switching to Ferrite-Core Transmitter

An additional noise-reducing improvement to the *trakSTAR* system is the ferrite-core version short-range transmitter. Compared with the original air-core short-range transmitter, the ferrite-core transmitter is the same size and adds slightly more weight in exchange for improved precision. This was an ideal solution for this application, where weight is of little concern. Ascension Technology Corporation exchanged the transmitters free of charge.

A comparison was performed between the two transmitters under equivalent conditions to determine how beneficial the ferrite-core version was. To capture data for analysis of noise, a C++ program called *ts_capture* was written. This program, described in Appendix L, records a series of data points from the *trakSTAR* system. Tests run using this utility provided quantitative data as to how much of an improvement this transmitter made. These data sets were captured for the two different transmitters using the same transmitter and sensor locations. Figure 2.24 and 2.25 show the noise comparisons for positions and orientations, respectively.

The ferrite-core transmitter in combination with selecting an optimized measurement rate and mounting the transmitter near the scanning surface was finally enough to decrease the noise to an acceptable level. With these improvements, performance of the *trakSTAR* system is now on par with the *Flock of Birds* and provides excellent performance in a small package. In its final configuration, the *trakSTAR* system has variances of less 0.002 mm and 0.002° for all axes in position and orientation, respectively.

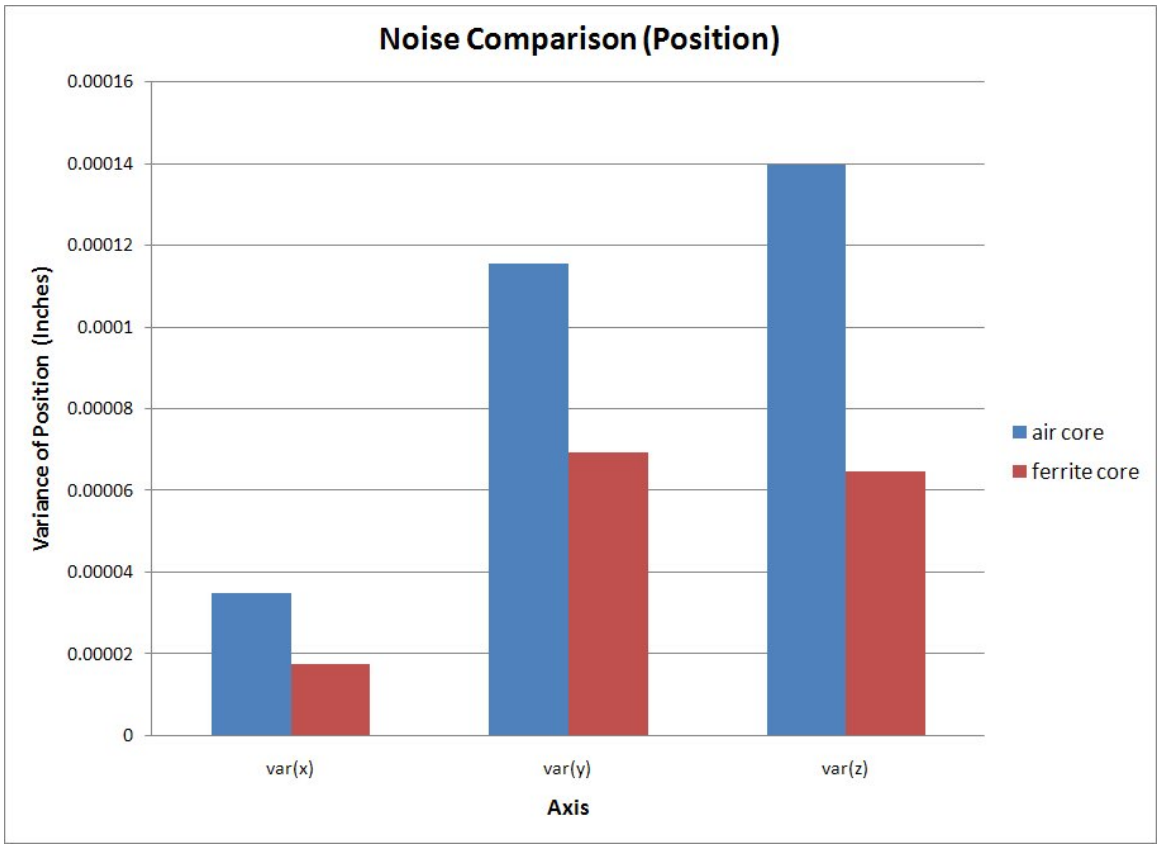


Figure 2.24: Comparison of Position Noise for *trakSTAR* Between Air-core and Ferrite-core Transmitters

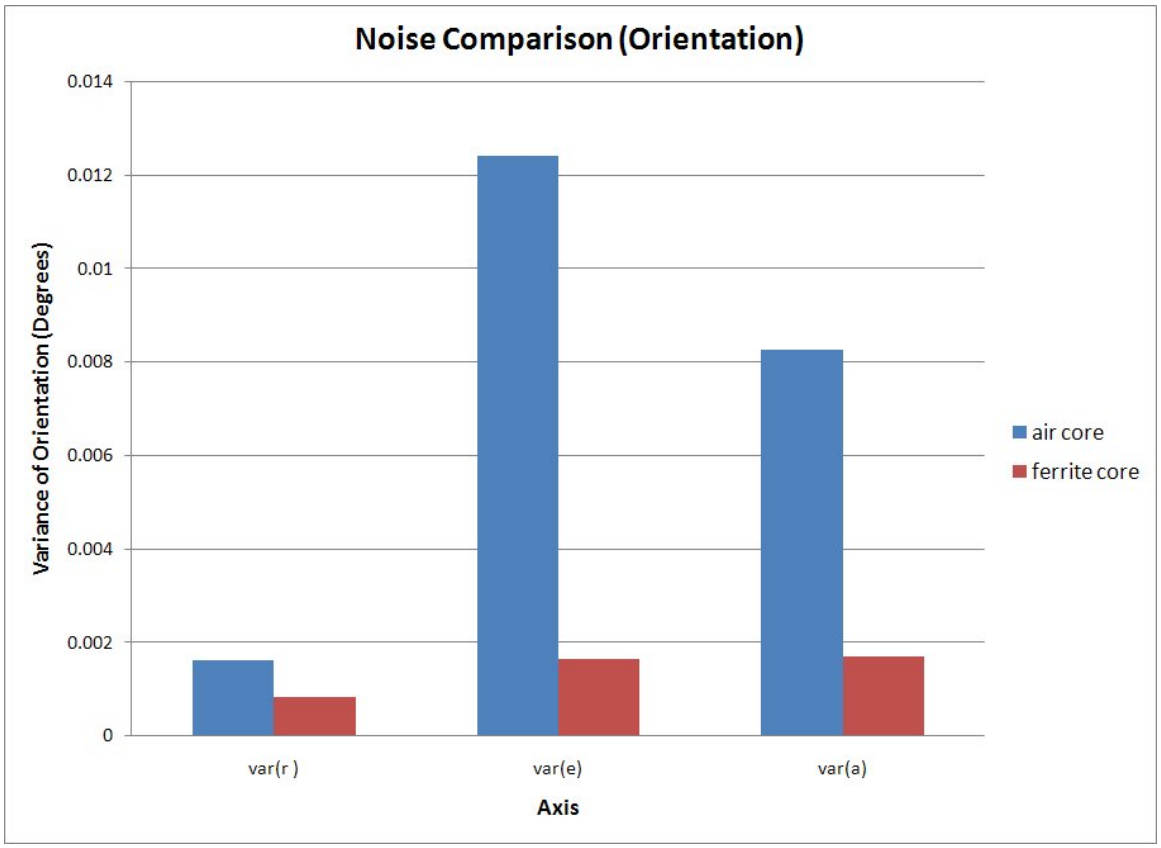


Figure 2.25: Comparison of Orientation Noise for *trakSTAR* Between Air-core and Ferrite-core Transmitters

2.5 Conclusions

A system for interactive generation of scan planes has been developed, which generates 2D scan planes based on input from the user. The toolkit *gear* was selected as a software starting point and used for development of the basic graphical display system. 3D image volumes can be loaded and rendered, and clipping is applied to generate a thin 'slice' of the data. Orientation of the scan plane can be interactively controlled using a 6 DoF input device or tracking system, such as the 3DConnexion *SpaceNavigator*, the Ascension Technology Corporation *Flock of Birds*, or the Ascension Technology Corporation *trakSTAR*.

This portion of the system provides the basic required functionality and a strong basis for additional design. Because custom software has been developed using a library, complete flexibility is afforded for future development. In the following chapters, further extensions upon this basic design will be discussed.

Chapter 3

Interactive Scanning Features and User Interfaces

3.1 Introduction

The basic scan plane display and interactive control described in Chapter 2 create an interactive display system, but do not implement many of the other features typically used by sonographers. There is no graphical user interface and no hardware interface for scanning. In this chapter the implementation of additional scanning features and user interfaces are discussed. The hardware user interface is the physical hardware, including the manikin and sham transducers, which provide a physically realistic training experience. The graphical user interface section describes the design of the graphical interface to the program, including menus and buttons, which control various parameters and interface with features. The interactive scanning features include useful controls such as probe geometry selection, scan depth setting and gain control.

3.2 Hardware User Interface

It is important that the system's physical hardware interface create a sense of realism. The hardware interface consists of the manikin with embedded tracking transmitter and the sham transducers, along with a touch screen monitor. Figure 3.1 shows a photo of the entire hardware interface as it would be used in a training session. This hardware should provide a realistic interface to adequately simulate scanning a patient. The following sections describe the individual components of the hardware interface.

Figure 3.1: Photo of Entire Hardware Interface

3.2.1 Manikin with Embedded Tracking Transmitter

The manikin provides a realistic scanning surface, closely resembling a human body and also serves to conceal the tracking transmitter from view. The criterion for selecting a manikin were that it must have a realistic feel and anatomically correct features, be compressible, and preferably not be excessively expensive. Of these requirements, compressibility is especially important for ultrasound training because

probe pressure is determined by the user.

A number of CPR, trauma and choking training dummies were considered including offerings from Laerdal and Darley. It was determined that the manikins most likely to have realistic compressibility would be Heimlich Maneuver training dummies, as they are clearly designed to be compressed. One such model is "Choking Charlie" from Laerdal [19]. This is a lifelike torso manikin that was cast from a human subject, with a compressible abdominal region for Heimlich Maneuver training and simulation. It is designed to expel a bolus from its airway when the Heimlich Maneuver is properly performed. This appeared to be an ideal option and Laerdal was willing to send a representative over with one for a hands-on evaluation. This evaluation proved that the manikin would adequately suit the needs of this system and one was purchased. Figure 3.2 shows the "Choking Charlie" manikin.

Figure 3.2: Laerdal "Choking Charlie" Manikin [19]

The manikin is mounted over the *trakSTAR* transmitter so as to completely conceal it. A cavity is cut into the manikin to accommodate the tracking transmitter. Conveniently, the manikin had a solid insert glued into its back as part of the breathing apparatus, which when removed, provided a perfect space for the tracking transmitter. The removed section and resulting cavity are shown in Figure 3.3.

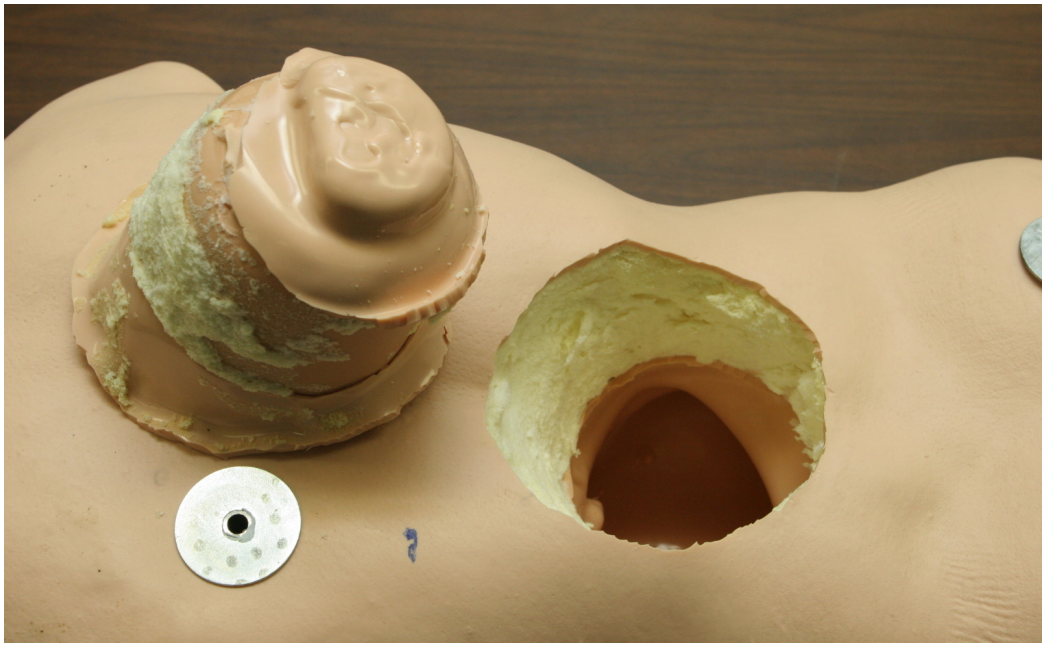


Figure 3.3: Cavity Cut into the Back Side of the Manikin to Accommodate Tracking Transmitter

The transmitter is mounted to a composite board for stability. A rigid strut is used to elevate the transmitter, placing it closer to the scanning surface for improved accuracy as described in Section 2.4.4. Installed in the board are four smooth rods for mounting the manikin. The board with strut and rods is shown in Figure 3.4. The manikin has four metal rods of slightly larger diameter installed into it, which fit over the rods on the board. Figure 3.5 shows the back of the manikin with the installed mounting rods. This mounting system allows the manikin to easily be removed and repositioned in a repeatable manner, while still ensuring that it is held firmly in place

during use.

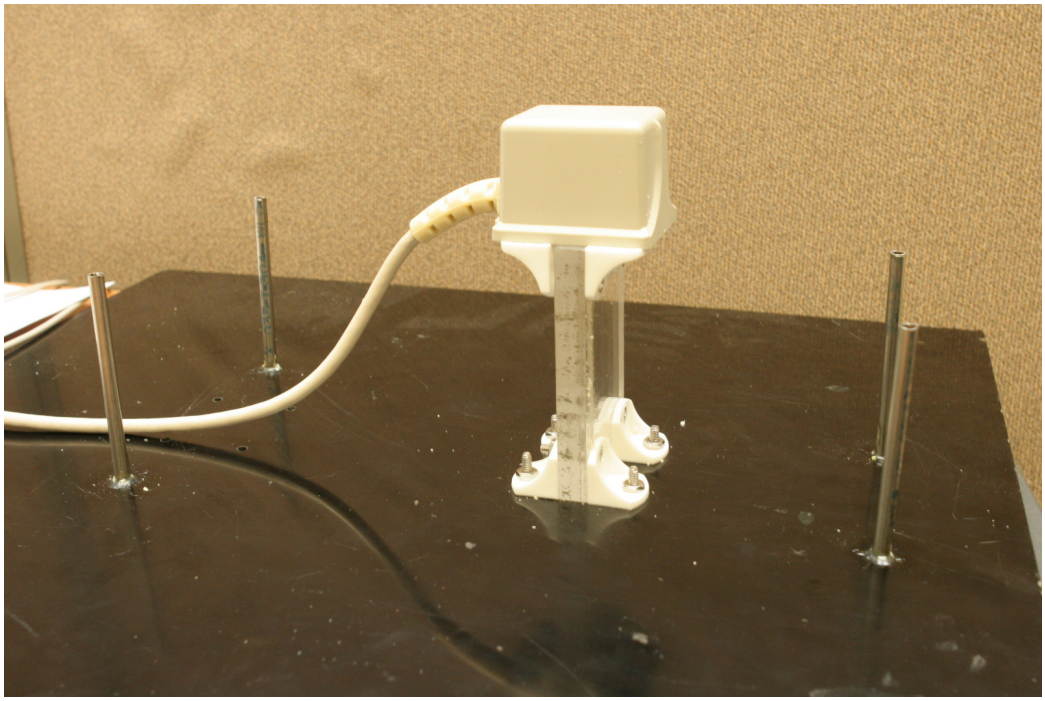


Figure 3.4: Manikin Mounting Board with Rods and Transmitter Strut

The transmitter strut is specially designed to avoid introducing any significant interference that would hinder the accuracy of the *trakSTAR* system. The strut itself is cut from a piece of clear acrylic and the mounting brackets are made of nylon. The nuts and bolts that hold everything together are 300 series stainless steel, which causes little to no interference with the *trakSTAR* system. The mounting rods attached to both the board and the phantom are also made from 300 series stainless steel and the mounting board is made from a non-metallic composite material. In tests, no added interference is seen when using the transmitter mounted on the strut, and in fact, the performance is greatly improved by the transmitter positioning as presented in Section 2.4.4.

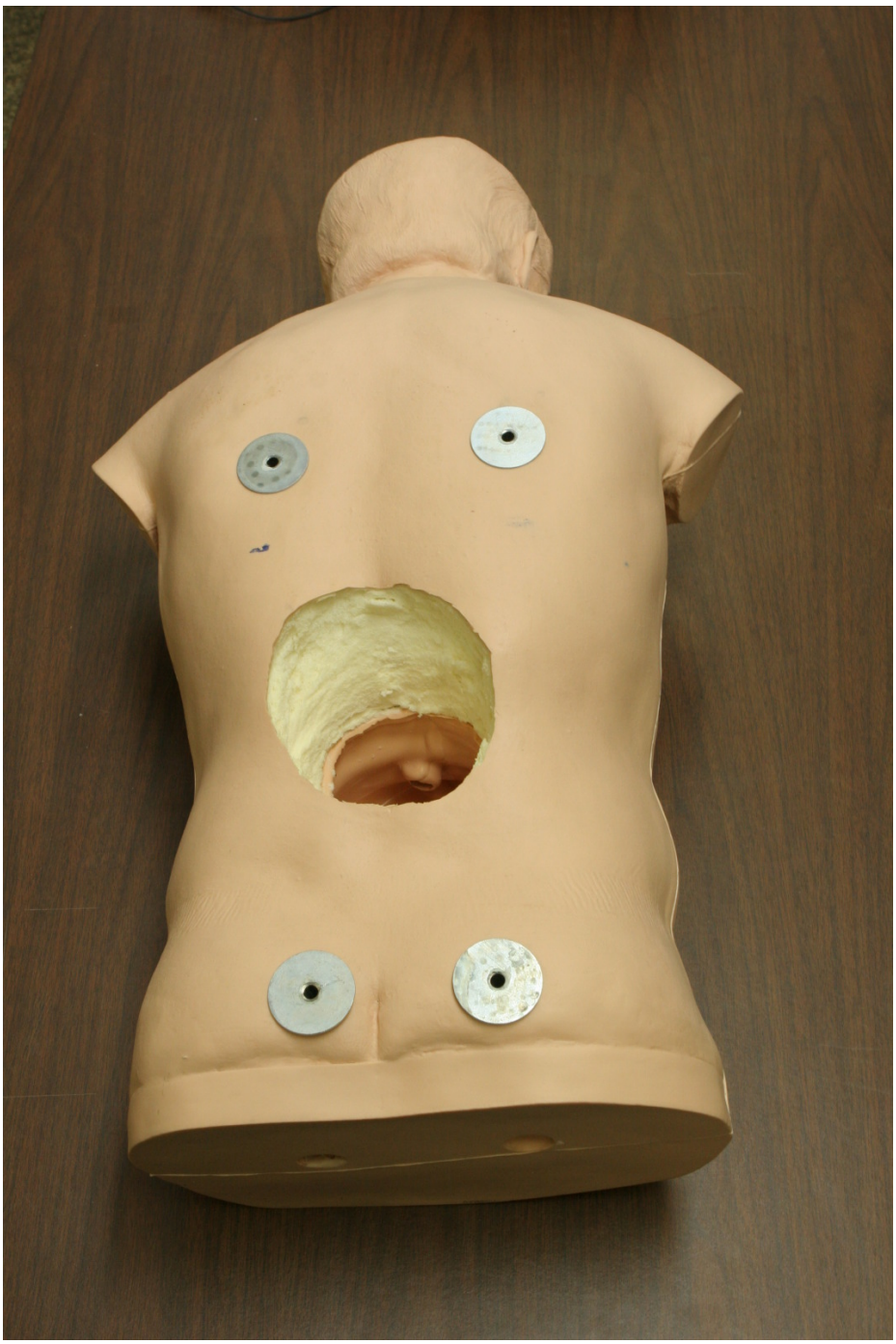


Figure 3.5: Back of Manikin Showing Mounting Rods

3.2.2 Sham Transducers

The sham transducers are made to mimic real transducers, but instead of ultrasound piezoelectric elements the sham transducers contain 6 DoF position sensors. The sham transducers must adequately conceal the tracking sensors and have a realistic look and feel.

Empty transducer shells were provided by Sound Technology [42], a manufacturer of ultrasound transducers. They provided both linear and convex array versions, which are shown in Figure 3.6. Two separate sensors were purchased for the *trakSTAR* system to construct both linear and convex array versions of the sham transducer.

Figure 3.6: Empty Transducer Shells for Sham Transducers

Figure 3.7 shows both sham transducers in their finished state. The transducer shells were filled with plumber's putty to hold the sensors in place and to add weight, as shown in Figures 3.8 and 3.9. Silicone alligator clip covers were used as strain reliefs for the cables. The contact surface was cut from a sheet of silicone rubber and held in place by plumber's putty. To attain an authentic feel, the weights and balance were compared to real transducers and adjusted appropriately. The end result is a pair of transducers that very closely resemble the look and feel of real ultrasound

transducers.

Figure 3.7: Sham Transducers

3.2.3 Touch Screen Interface

To provide a higher level of interactivity and permit additional training features, a touch screen was added to the system. The primary task for integrating a touch screen was selection of an appropriate model.

There are three primary touch screen technologies: resistive, capacitive and surface acoustic wave (SAW). Resistive technology uses two resistive metallic layers to detect where a touch occurs. The voltage drop that is generated by the resistance of the two layers can be used to calculate the position of the touch [15]. Capacitive touch screens transfer a small amount of charge to the user's finger, which is measured by sensors at the corners of the display to extract position [15]. SAW uses ultrasound waves that propagate across the glass surface of the screen. Transmit and receive transducer

Figure 3.8: Inside of Linear Sham Transducer

Figure 3.9: Inside of Convex Sham Transducer

pairs are used for both axes of the monitor and modifications to the received wave can be used to determine where a touch occurs [15]. These technologies vary primarily in their light transmission and how a touch can be triggered, which are the reasons a certain technology is typically selected for a given application. Table 3.1 shows a tabular comparison of these technologies.

Table 3.1: Touch Screen Technology Comparison [15]

Parameter	**Resistive**	**Capacitive**	**Surface Acoustic Wave**
Light Transmission	∼75%	∼90%	∼90%
Contact Object	Anything	Only skin or conductive materials	Anything

For this design, maximum light transmission is desirable because of the need to see small details within the data. Furthermore, ultrasound images can often be fairly dark, so the best possible light transmission is desirable. Because this system will be used in a clinical environment, it is likely that it will be used by trainees wearing gloves, thus there should be no requirement for direct skin contact. Based on the requirements and the information shown in Table 3.1, SAW is the clear choice for touch screen technology, as it provides high light transmission and does not require direct skin contact.

A 17" SAW touch screen by Planar Systems, Inc. was selected, which is shown in Figure 3.10. Planar [27] is a well-established company that has been in the display business since 1983. They have a strong market share and are known for producing quality touch screens. The mid-size 17" model wave selected as a compromise between screen size and cost, and represents a typical display size used for ultrasound systems. Table 3.2 shows the manufacturer's specifications for the selected monitor.

The touch screen acts as a monitor and a USB mouse, providing both input and output for the system. Since the screen replaces the monitor and mouse, no additional implementation effort is necessary to use the training system with the touch screen.

Table 3.2: Planar Touch Screen Specifications [27]

Product Name	PT1705MU
Planar Part Number	997-4409-00
Viewable Size	17 inch diagonal
Touchscreen Type	SAW (Surface Acoustic Wave)
Touchscreen Interface	USB
Transmissivity	88% (Min)
Contrast Ratio (Typical)	700:1
Viewing Angle (Typical)	160°H,V (specified at CR¿10:1)
Response Time (Typical)	6 ms (2 ms rise, 4 ms fall)
Brightness (w/touchscreen)	270 cd/m^2
Brightness (w/o touchscreen)	300 cd/m^2
Display Resolution	1280 x 1024
Tilt Range	-5°to 90°
Pixel Pitch	0.264 mm
Refresh Rate	56 to 75 Hz
Dimensions (W x H x D)	15.8” x 14.6” x 8.9” (400.0 mm x 370.7 mm x 225.0 mm)
Dimensions without Stand	15.8” x 13.3” x 2.7” (400.0 mm x 337.8 mm x 68.6 mm)
Display Weight	With Stand: 17.2 lbs (7.8 kg) (net)
Video Inputs	Analog
Audio Output	2 speakers, 1W/ch
Compatibility	Windows Vista, XP, 2000, 98
External Connections	D-sub 15-pin, DC power connector, Type B socket USB, Stereo jack
Power Supply	Internal AC power supply,12V DC power connector
Power Requirements	100-240 VAC
Power Consumption	50W
VESA Compatible/Location	Built-in 75mm and 100mm VESA, back
Service and Support	3-Year Customer First Warranty
Options / Features	Anti-glare coating, Multi-language support, On screen display
Product Approvals	UL/c-UL , FCC-Class B, TUV-T mark , CE, CB, RoHS

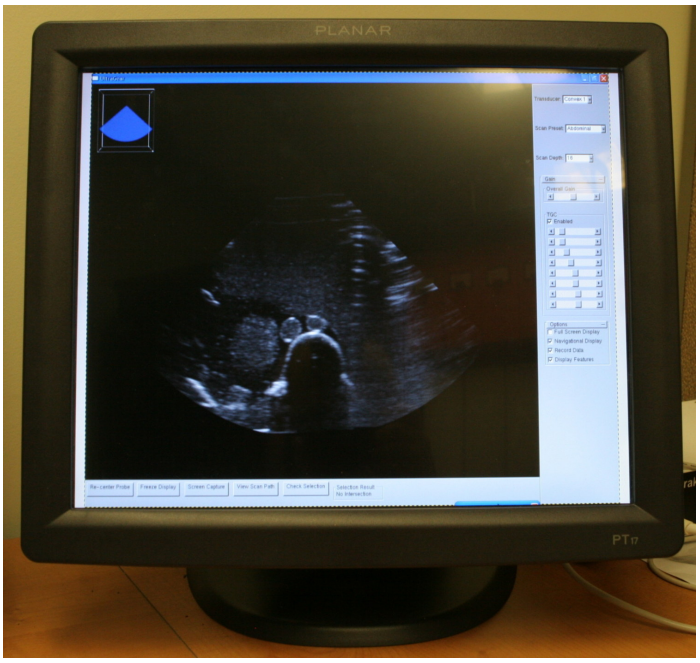


Figure 3.10: Planar PT1705MU Touchscreen Monitor

3.3 Graphical User Interface Design

Having an effective user interface is an important part of any software application and this is no exception. In addition to basic usability features, the system must also emulate a user interface typical of ultrasound scanners to provide a realistic training experience. The display should look somewhat similar to that of the popular models of ultrasound scanners and should have similar features. This section describes the design of the graphical user interface (GUI) for the training system.

3.3.1 GUI Implementation

A common method of developing a GUI is to first develop the GUI and then add the desired functions into it, but with OpenGL applications there is another option: to implement the GUI within the OpenGL application. Because OpenGL performs graphical rendering, it is logical that it should be possible to design a GUI with this

environment. This is certainly possible and has been implemented in a number of open source libraries.

One such library is called GLUI [33], which works with the *OpenGL Utility Toolkit* (GLUT) and is designed to be simple yet powerful. It integrates easily into existing OpenGL applications and provides a number of options for clickable controls. Because it is simply drawing more OpenGL objects to the screen, a GUI can be integrated into a pre-existing application with minimal modification. Since the GUI is drawn using OpenGL, it is also inherently platform-independent, which is not always true for GUIs.

3.3.2 GLUI Details

GLUI uses live variables, which are global variables that are modified by the controls. Each control has live variables associated with it, which are instantaneously modified whenever the control is changed by the user. GLUI also provides update functions which can be used to set the controls based on changes to the live variables by the software. These update functions can be used to ensure that the controls are always synchronized with the live variables.

For most of the controls, GLUI also offers the option to use a callback function. This is a function that is instantly called when the control is modified. For example, a callback function can be triggered when a button is clicked. The callback function can be used to change values in the program or to perform a specific function.

Figure 3.11 shows a flowchart of the operation of GLUI. When any value-based or numerical control in the GUI is changed, the associated live variable is changed to the current value of the control. Next, if the control has a callback function specified, this function is called. The callback checks the callback identifier to determine which control issues the callback, and then the appropriate actions for the control are performed. After this process has completed, normal program operation is resumed.

GLUI includes a number of different object types for creating controls on the

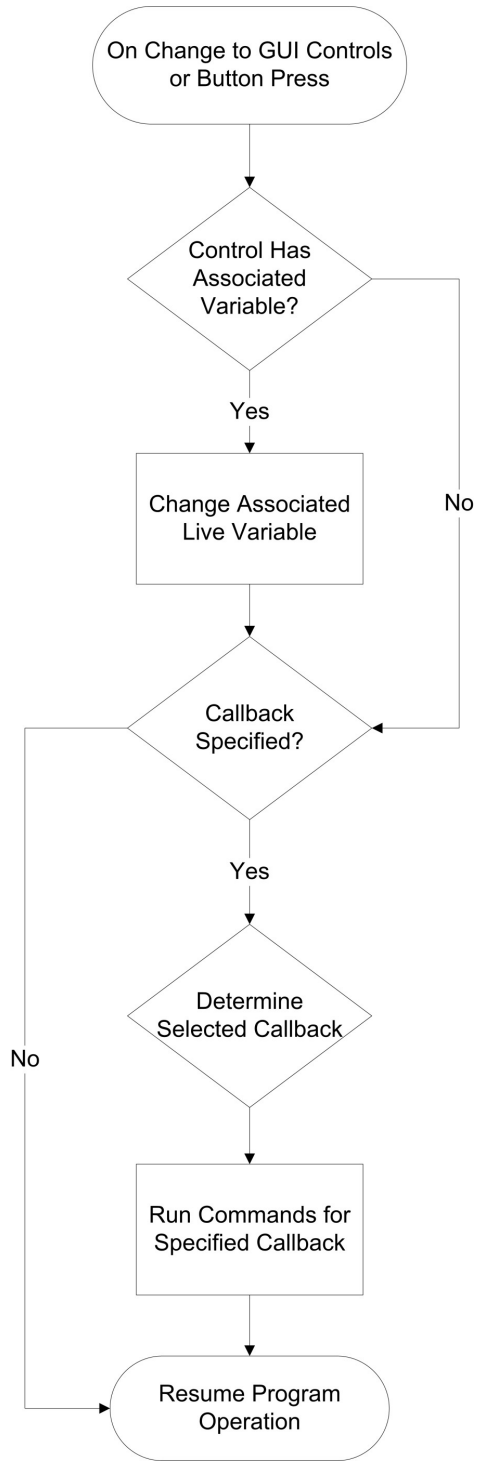


Figure 3.11: GLUI Software Flowchart

screen. These include dropdown menus, sliders, checkboxes, and buttons. GLUI also provides for creating groups of objects surrounded by a box and roll-out groups, which can be compressed to a single item when they are not needed. Many of the object types have adjustable parameters that allow adjustment of how the control is drawn or how it acts. For example, the size of buttons can be modified to make them larger or smaller.

3.3.3 GUI Design

The GUI for the simulation program has been modeled after the GUI of the Terason software [45]. Since Terason is a full-featured PC-based ultrasound system that requires no external hardware control interface, it is an ideal interface to emulate. All of the user interface features are implemented in software and accessible through the GUI.

Figure 3.12 shows the GUI for the simulation system. Additionally, Terason's GUI is in Figure 3.13 for reference. The buttons and controls are designed to be large enough that they can be used with the touch screen display.

Figure 3.14 shows a block diagram of all of the features that are controlled by the GUI. The GUI includes a sidebar with options for selection of transducer type, preset exam settings, and probe depth. Transducer type allows the user to select between the available probes, which will automatically begin using a different physical sham transducer. Preset exam settings automatically set the gain, TGC, and depth to preset values, as described in Section 3.4.5. Probe depth allows the user to select between a number of different scan depths using a dropdown menu. The overall system gain is controlled by a single slider. The time gain compensation can be adjusted using a series of sliders to control the gain based on the depth of features.

The options control set contains a few options that are not as commonly used. The 'Navigational Display' checkbox controls whether or not the navigational display is drawn. The 'Record Data' option controls the data recording, permitting the user

Figure 3.12: Simulation System GUI

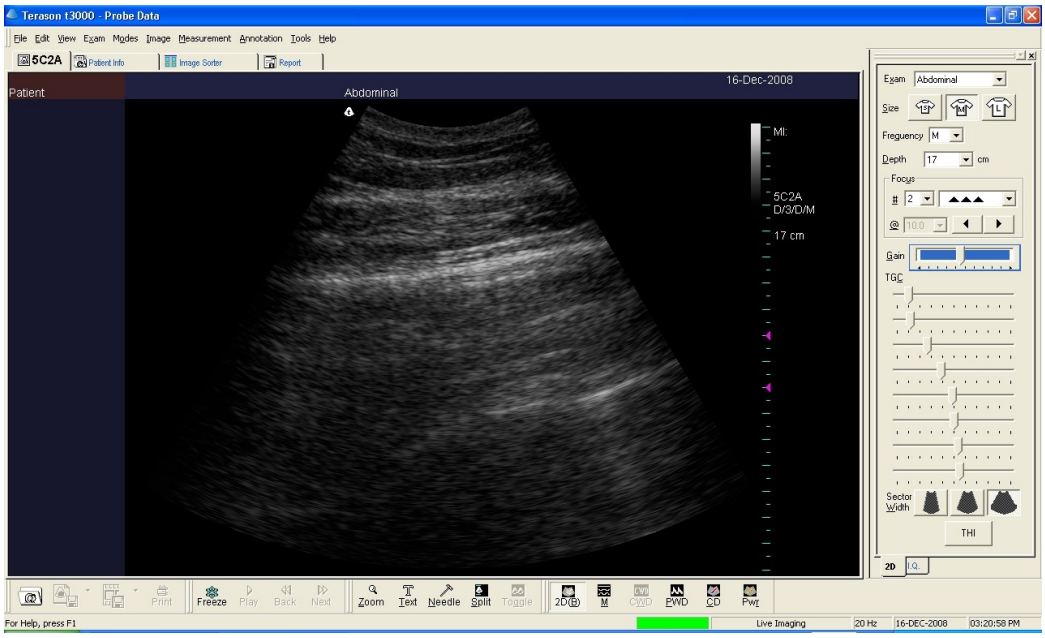


Figure 3.13: Terason's GUI

to start and stop recording at any time. 'Display Features' determines whether the regions of interest included in the data set are drawn on the screen or kept hidden. In most cases the regions of interest will be hidden and left for the trainee to properly identify, as described in Section 5.2. The 'Fullscreen' checkbox controls whether the system runs in the normal windowed mode or if it is expanded to cover the entire screen. In fullscreen mode, the title bar and the operating system's task bar are hidden, to provide a cleaner interface and remove awareness of the underlying operating system.

There is also a bottom menu bar, which contains easily accessible buttons for some important training features. 'Re-center Probe' recalibrates the probe center position. This will generally need to be done at startup and upon switching sham transducers. 'Freeze Display' holds the current view, which is useful for capturing images and selecting regions of interest. To simplify the user interface and avoid any confusion, the same button is used for freezing and un-freezing the display. When the display is

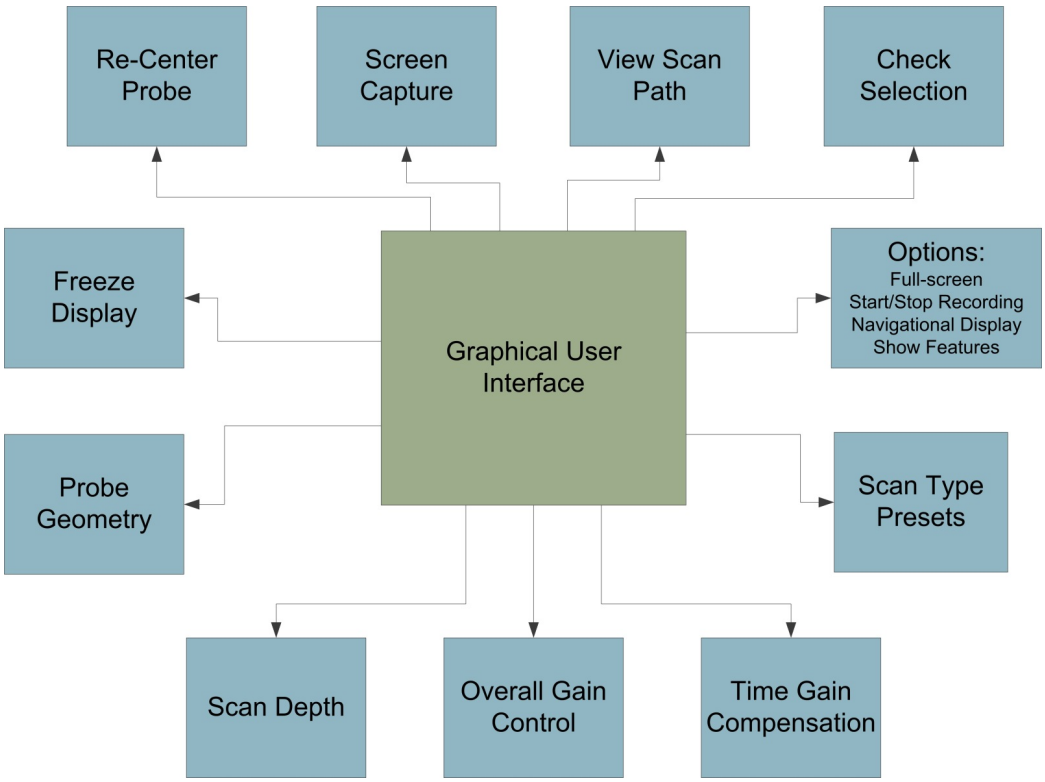


Figure 3.14: GUI Block Diagram

frozen, the button text is changed to 'Unfreeze' and is changed back after the display is unfrozen. As the name would suggest, 'Screen Capture' captures a still image of the screen in the form of a JPEG image. Still image capture is explained further in Section 5.4. The 'View Scan Path' button can be used to launch a MATLAB script that displays the recorded scan path as described in Section 5.3. The 'Check Selection' button is part of the region of interest selection training feature, described in Section 5.2. When this button is pressed, the selection evaluation function is called to determine whether the user's selection is correct or not. The result of this check is displayed in the textbox next to the button.

3.4 Interactive Simulation Features

In this section, additional interactive features implemented into the training system are discussed. These features are not required for the most basic functions, but are useful features that improve the system and make it more realistic. Modeled after features and capabilities implemented in commercial ultrasound systems, these features are designed to create a training system that resembles a commercial ultrasound system as closely as possible.

Figure 3.15 shows a block diagram of the various interactive simulation features that will be discussed in this section. All of these features can be activated from the graphical user interface (GUI) shown in Figure 3.16. In addition, scan depth, gain, and time gain compensation can be modified automatically to preset values by selecting a scan preset using the GUI.

Probe geometry selection permits users to work with different probe geometries depending on their current task. Scan depth setting, gain control, and time gain compensation provide the capability to adjust the image for optimal viewing of specific anatomical features. The navigational display provides a reference point as to where the current scan plane resides in the overall volume. The freeze display feature allows

the user to lock the display at a given point for further analysis.

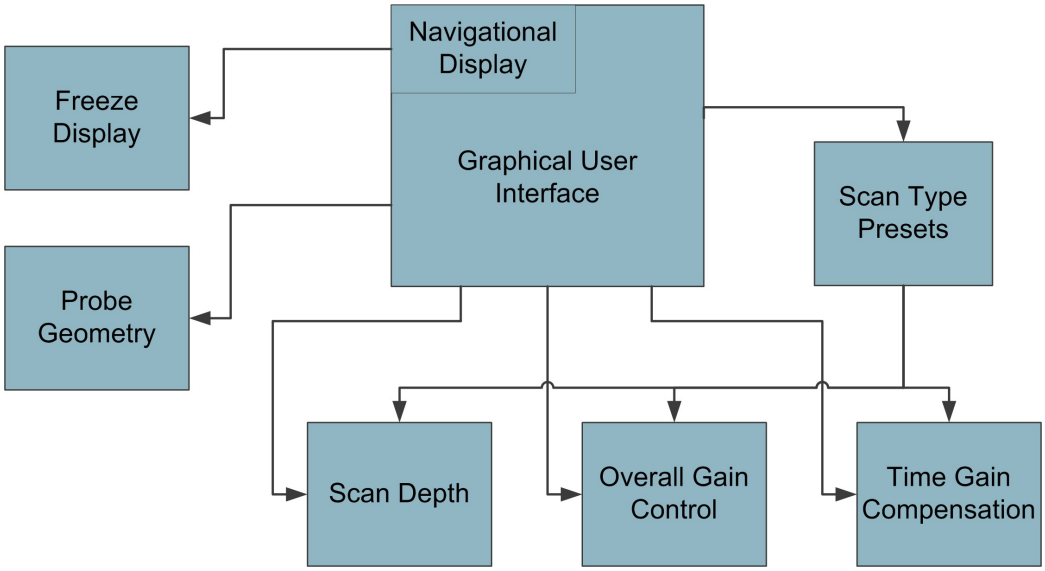


Figure 3.15: Diagram of Interactive Simulation Features

3.4.1 Probe Geometry Selection

Ultrasound transducers exist in a number of different geometries including convex array, linear array, and phased array, and can be strategically selected to provide a better view for a given situation. This feature provides the option to select probe types relevant to the imaging application.

Changing the current probe geometry alters both the image seen on the screen as well as the physical sham transducer that is used. The user may select a different probe in software and then begin using the corresponding sham transducer. Changing the probe type and corresponding image geometry alters the shape of the stencil that determines the visible portion of the current image plane. Selection of the initial probe shape is determined automatically by the sham transducer that is attached to port 1 of the *trakSTAR* system. Based on the serial number of the attached tracking sensor, the software determines which sham transducer it is and applies the appropriate

Figure 3.16: Graphical User Interface Screen Capture

probe shape. The detection algorithm has been designed such that the software can determine which probe is connected to each port of the *trakSTAR* system and account for that accordingly. Thus, selecting a convex probe in software will always switch to the convex sham transducer, regardless of whether it is plugged into port 1 or 2.

By applying a mask to the image being viewed, it can be altered to take on the appearance of the image geometry of the specific transducer. This allows users to experience scanning with different probe shapes and extends the usefulness of this training system. This masking is accomplished in OpenGL using a 'Stencil Buffer'. The stencil buffer acts very much like a stencil one might use to paint letters onto a sign. A black and white mask is defined which specifies the regions to be drawn or to be blocked. A comparison function is used to determine which pixels to draw and which to ignore. In this implementation, white pixels are drawn, while black ones are ignored. By appropriately drawing and applying the stencil, the envelope of the display can be made to take on any shape. Different stencils are generated based on the selected probe geometry, to accurately portray the viewing area of the selected probe.

Currently, no additional signal processing is performed to modify the image volume to match a given geometry. Therefore, it is not possible to make a data set captured with one transducer geometry look exactly like it was taken with another. For example, data captured with a convex array will always exhibit radial shadowing and speckle indicative of the probe geometry, which will not look quite correct for a linear array geometry. However, this feature does provide the opportunity for the user to work with the viewing area corresponding to the chosen transducer, which is still quite beneficial. An alternate solution to this issue would be to capture both linear and convex array versions of each data set and load a different data volume when transducers are changed.

Figures 3.17 and 3.18 show convex and linear transducer mask shapes applied to an ultrasound volume being viewed.

Figure 3.17: Convex Array Transducer Shape

Figure 3.18: Linear Array Transducer Shape

3.4.2 Scan Depth Setting

Scan depth is an essential feature for medical ultrasound and is thus made adjustable. To allow the depth to be modified, a 'Depth Setting' menu was created. This adjustment appropriately scales the image to show the selected depth. Selecting a different depth from the menu performs a transformation to the image being viewed that modifies the magnification of the volume and the area being viewed.

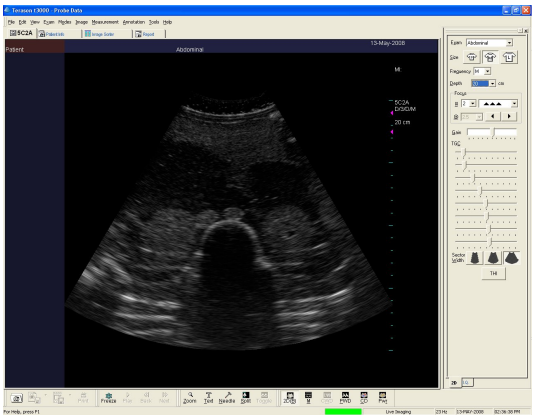
Modifying the depth affects the viewed image differently for linear and convex probes. As shown by the Terason screen captures in Figure 3.19, when a linear array transducer is chosen, the aspect ratio of the scan is also altered. For convex array geometries, as scan depth is increased, the aspect ratio of the scan remains the same because the array angle is maintained regardless of depth.

Figures 3.19 and 3.20 show a comparison between changing the depth in Terason and in the simulation software. In both cases, linear and convex probes are shown.

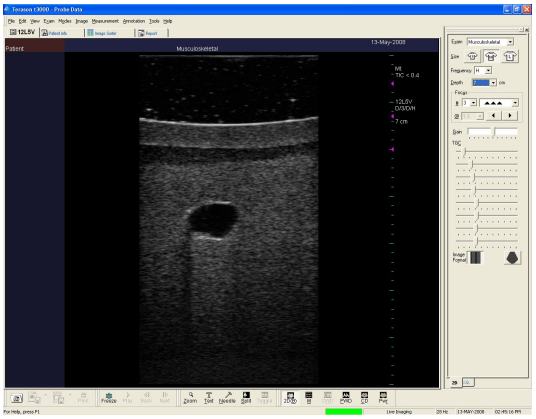
It should be noted that one inherent difference between the training system and a real ultrasound scanner is that the maximum depth for a linear probe is the same as the convex probe in the simulation system. This is due to the fact that the data are the same regardless of which probe is being used. The depth range can of course be limited in software if it is desired that the linear probe only allow a certain range of depths.

3.4.3 Overall Gain Control

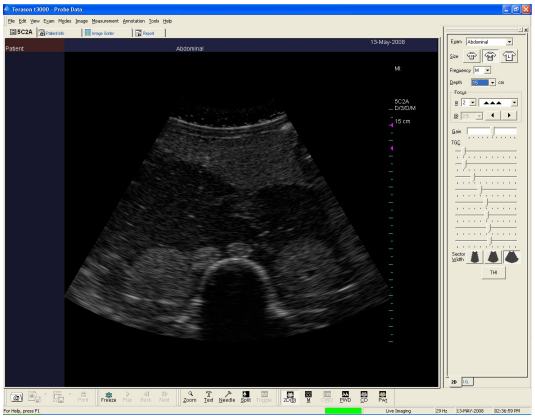
Adjustment of the signal gain is an important part of ultrasound imaging. Altering the overall gain on an ultrasound system modifies the receive amplifier gain and has the effect of making the image appear darker or brighter. This is necessary because some dark features may not be visible if the image is too dark and likewise, details in lighter images may be obscured by an image that is too bright. Sonographers will typically adjust the gain to an appropriate value before scanning and will sometimes



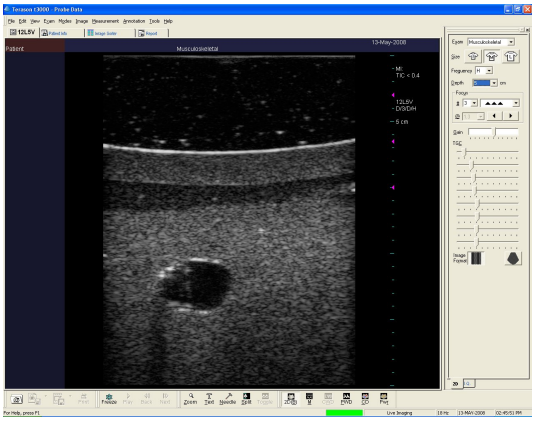
(a) Convex 20 cm



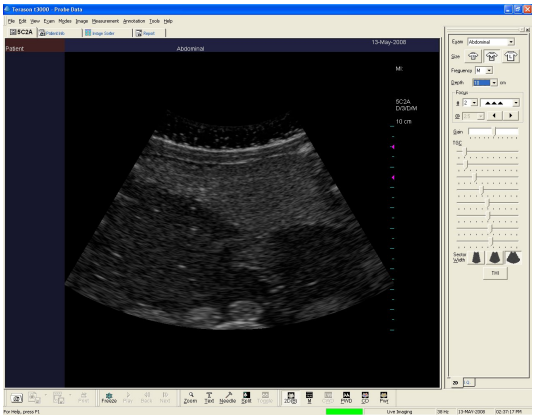
(b) Linear 7 cm



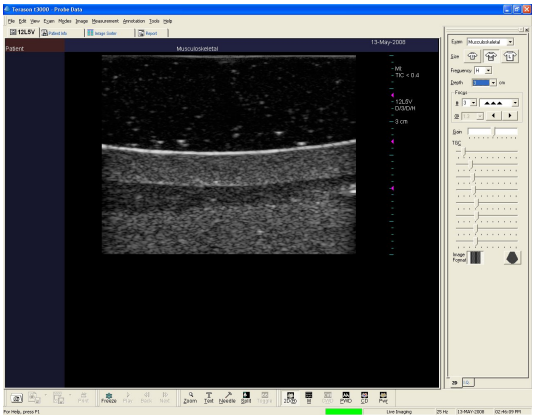
(c) Convex 15 cm



(d) Linear 5 cm



(e) Convex 10 cm



(f) Linear 3 cm

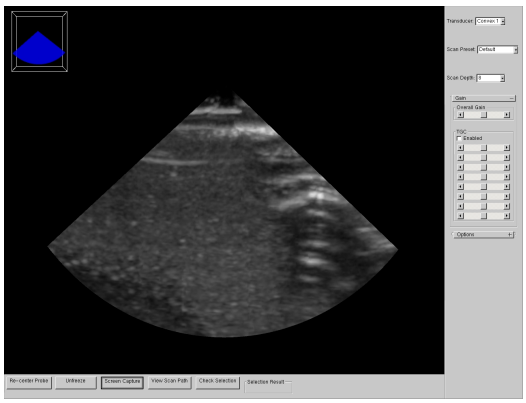
Figure 3.19: Terason Convex and Linear Probes at Various Depths

(a) Convex 16 cm

(b) Linear 16 cm

(c) Convex 12 cm

(d) Linear 12 cm



(e) Convex 8 cm

(f) Linear 8 cm

Figure 3.20: Simulation System Convex and Linear Probes at Various Depths

modify the gain during a scan to achieve a better view of specific features.

Figure 3.21 shows a block diagram of the gain adjustment process. The gain parameter is used to generate a blending function, which is then applied as an overlay on the displayed scan image.

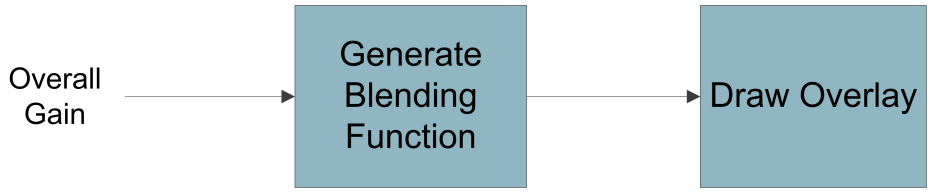


Figure 3.21: Block Diagram of Overall Gain Adjustment

The overall gain control was implemented by applying a semi-opaque mask to the image being displayed. By adjusting this mask through a range of values between opaque black to transparent to opaque white, the entire range of gains can be achieved. The gain adjustment is implemented in OpenGL using OpenGL's blending capabilities. The mask is created and then blended with the underlying image to produce a proper gain-adjusted final image.

The variable *gain_overall* is used for the overall gain. This variable can take a continuous range of values between 0 and 2, with less than 1 being darker than the original image and greater than 1 lighter than the original. For *gain_overall* values greater than 1, the image is blended with a greyscale color represented by (*gain_overall*-1, *gain_overall*-1, *gain_overall*-1), where the three values are red, green, and blue, respectively. The numerical value of 1 is subtracted from each of the *gain_overall* values to put them in an appropriate range of 0 to 1, which spans the range from transparent to opaque white. When the gain is less than or equal to 1, the image is blended with a greyscale color represented by (*gain_overall*, *gain_overall*, *gain_overall*), with the range spanning from opaque black to transparent.

Figure 3.22 shows approximately the same scan plane captured at three different gain levels. For comparison, Terason ultrasound captures using three similar gain

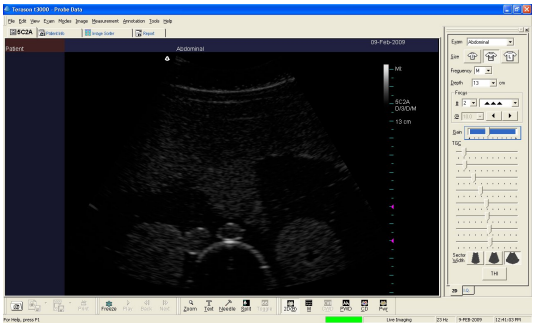
levels are also shown. It can be seen that the implemented gain control has similar effects on the simulated image as changing the gain for the Terason system does on the observed ultrasound scan images.

3.4.4 Time Gain Compensation (TGC)

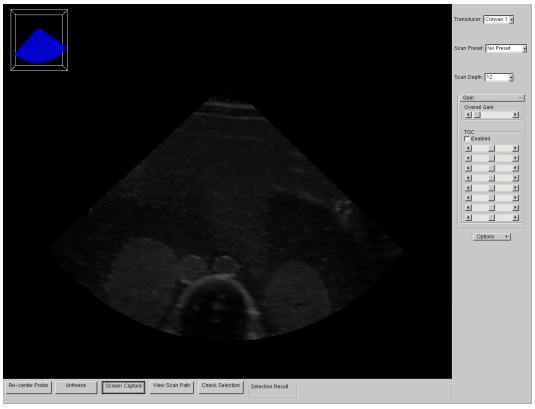
Time gain compensation (TGC) is a common feature on ultrasound systems. This control modifies the signal gain based on the relative time of the received signal. Receive time corresponds to depth in the displayed image, so this is effectively a depth-based gain control and is sometimes referred to as Depth Gain Control. TGC is essential for adjusting for decreased signal strength deeper within the subject. It can also be used to improve visibility of certain features. TGC is frequently adjusted by sonographers before and during a scan, in order to better view features within the body. Most ultrasound systems have eight TGC adjustments and this number has become a *de facto* standard for medical ultrasound [2].

Eight TGC slider adjustments are provided in the GUI to set the TGC values, which correspond to equally-spaced depths on the scan image. Adjusting a given slider instantly modifies the gain for that portion of the image.

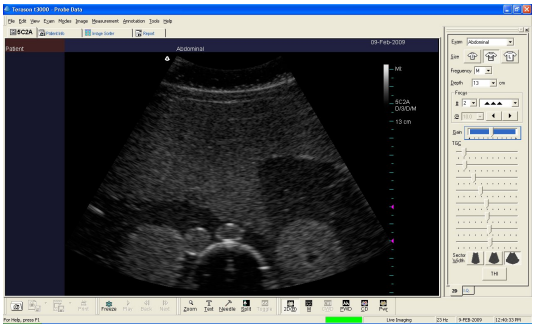
If only eight distinct gain values were used, the gain would look discontinuous. To solve this without having an excessive number of TGC adjustments, linear interpolation is performed between the eight adjustment points to create a smooth gradation. This interpolation is calculated as a weighted average of the two nearest points. Equation (3.1) shows how the gain for a given depth is calculated. i is the loop variable that increments through the different TGC control values and j is the loop variable that goes through the intermediate steps between the controlled gain values. tgc[i] is the value of TGC slider i. As the depth changes between two consecutive TGC values, the weighting of a given value is increased as the depth approaches that value. Table 3.3 shows an example of the linear interpolation between two consecutive TGC parameters of 0.5 and 1.0, with a MAX_STEPS value of 20.



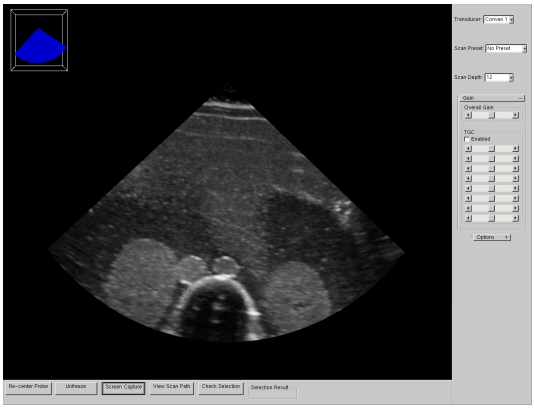
(a) Terason Low Gain



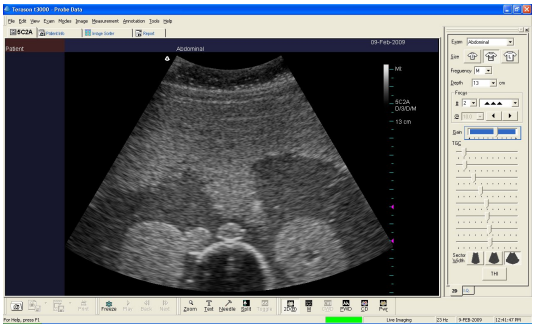
(b) Simulation Low Gain



(c) Terason Medium Gain



(d) Simulation Medium Gain



(e) Terason High Gain

(f) Simulation High Gain

Figure 3.22: Comparison of Gain Adjustment between Terason and the Simulation System

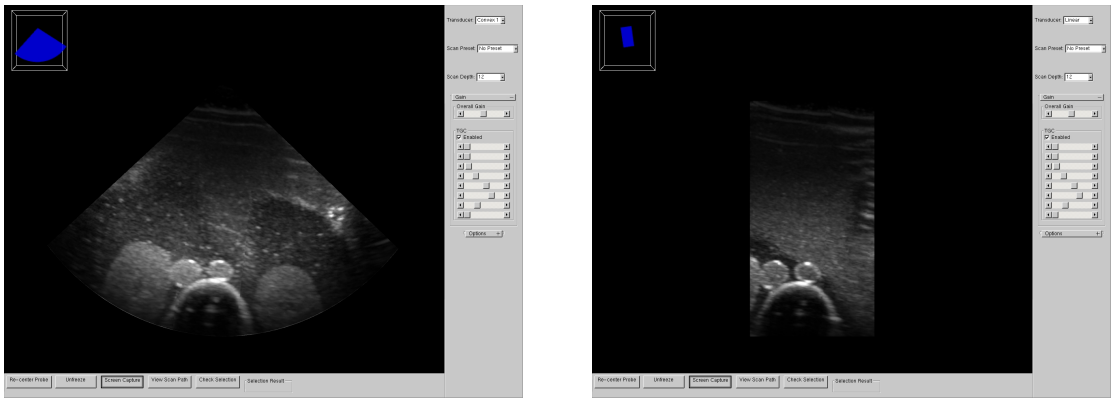
$$\text{gain} = (\text{MAX\_STEPS} - j) * \text{tgc}[i] + j * \text{tgc}[i+1] \tag{3.1}$$

tgc[i]	tgc[i+1]	j	gain
0.5	1	0	0.5
0.5	1	1	0.525
0.5	1	2	0.55
0.5	1	3	0.575
0.5	1	4	0.6
0.5	1	5	0.625
0.5	1	6	0.65
0.5	1	7	0.675
0.5	1	8	0.7
0.5	1	9	0.725
0.5	1	10	0.75
0.5	1	11	0.775
0.5	1	12	0.8
0.5	1	13	0.825
0.5	1	14	0.85
0.5	1	15	0.875
0.5	1	16	0.9
0.5	1	17	0.925
0.5	1	18	0.95
0.5	1	19	0.975

Table 3.3: Example of TGC Linear Interpolation

For the TGC to look realistic, its effect must differ between linear and convex probe geometries. Physically, TGC is a time-based gain, and the gain therefore varies with the distance from each transducer element. This means that for a convex array, the gain changes with the radial distance from the transducer, while for a linear array, the gain varies with the linear depth of the image. Figure 3.23 shows a comparison of linear and convex geometries with TGC applied.

TGC is implemented in a manner very similar to the overall gain control described in Section 3.4.3, except that the gain is applied in steps to create a gradient corresponding to the selected gain values. Each infinitesimal gain step is generated



(a) Convex TGC Example (b) Linear TGC Example

Figure 3.23: Comparison of TGC for Linear and Convex Probe Geometries

according to the block diagram shown in Figure 3.24. For the given depth, the current and next TGC parameters are read, as well as the overall gain. These are used to calculate the gain as described in Equation (3.2), which is similar to the linear interpolation described Equation (3.1), except that the overall gain is also included.

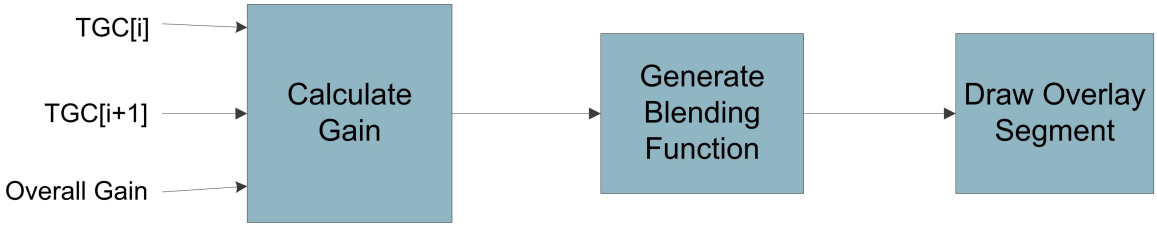


Figure 3.24: Block Diagram of TGC Generation For a Single Step

$$\text{gain} = \text{gain\_overall} * ((\text{MAX\_STEPS} - j) * \text{tgc}[i] + j * \text{tgc}[i+1]) \qquad (3.2)$$

A blending function is then generated in the same manner as the overall gain. TGC parameters, which are stored in an array called *tgc* can take a floating point value between 0 and 2. For *tgc[i]* values greater than one, the image is blended with a greyscale color represented by ($tgc[i]$-1, $tgc[i]$-1, $tgc[i]$-1), where the three values

are red, green, and blue, respectively. The value of 1.0 is subtracted from each of the gain values to put them in an appropriate range of 0 to 1, which spans the range from transparent to opaque white. When the gain is less than or equal to 1, the image is blended with a greyscale color represented by ($tgc[i]$, $tgc[i]$, $tgc[i]$), with the range spanning from opaque black to transparent.

The overlay segment is calculated as a fractional portion of the entire scan image area. The slice location for given values of i and j are given in (3.3) and (3.4) for linear and convex arrays, respectively. The total number of steps is $7 * \text{MAX\_STEPS}$ and the current step is $i * \text{MAX\_STEPS} + j$, so the current segment location is calculated as the fractional distance through the scan plane determined by the ratio of the current step to the total number of steps. The shift for the linear array value is due to the fact that the segment is drawn as a rectangle between curr_slice_linear and curr_slice_linear+slice_width_linear, while the convex array is drawn as an arc that is translated up by 1.1 in the y-axis. Equations (3.5) and (3.6) show the segment width, which is the fractional portion of the entire scan image area that is represented by one step.

$$\text{curr\_slice\_linear} = 0.9 - (i * \text{MAX\_STEPS} + j) * (2.4/(7 * \text{MAX\_STEPS})) \quad (3.3)$$

$$\text{curr\_slice\_convex} = (i * \text{MAX\_STEPS} + j) * (2.6/(7 * \text{MAX\_STEPS})) \quad (3.4)$$

$$\text{slice\_width\_linear} = 2.4/(7 * \text{MAX\_STEPS}) \quad (3.5)$$

$$\text{slice\_width\_convex} = 2.6/(7 * \text{MAX\_STEPS}) \quad (3.6)$$

Figure 3.25 shows a flowchart of the software for the TGC. The software first checks

if TGC is enabled by the checkbox in the GUI. If it is disabled, TGC is skipped entirely to save on processing. Next, the software checks whether the current transducer is a linear or convex array and uses the appropriate processing code. The processing for linear and convex arrays is essentially the same, except that the equations are slightly different to account for how the gain is applied as described earlier in this section. Next a pair of nested loops are used to generate the TGC overlay. The outer loop increments through the 8 different TGC parameter values, while the inner loop increments through the intermediate steps between each value. At each intermediate step, the gain is calculated using linear interpolation and then applied to a segment of the image.

Figure 3.26 shows a few examples of different TGC parameters being applied to a scan image. Figure 3.26(a) shows the scan image without any TGC settings applied. Figure 3.26(b) gives an example of a TGC curve that has increased gain near the surface and reduced gain in the middle depth range. In Figure 3.26(c), the gain is reduced near the surface and increased in the deeper region. Figure 3.26(d) shows a TGC curve that targets a specific region near the center of the scan.

3.4.5 Scan Type Presets

Scan type presets are a common feature for ultrasound systems, which provide quick pre-defined settings for a variety of common scan types. While these are not perfect for every scan of a given type, they are generally a good starting point for further adjustment. A sonographer performing an abdominal exam might start with the abdominal preset and then make some minor adjustments to the gain and TGC to suit the specific patient and scan. Presets commonly affect the scan depth, focus, gain, and TGC values.

Figure 3.27 shows a block diagram of how the scan type presets work. When a preset is selected, it automatically changes the scan depth, gain, and TGC to the preset values for the given scan type. Because this training system does not currently

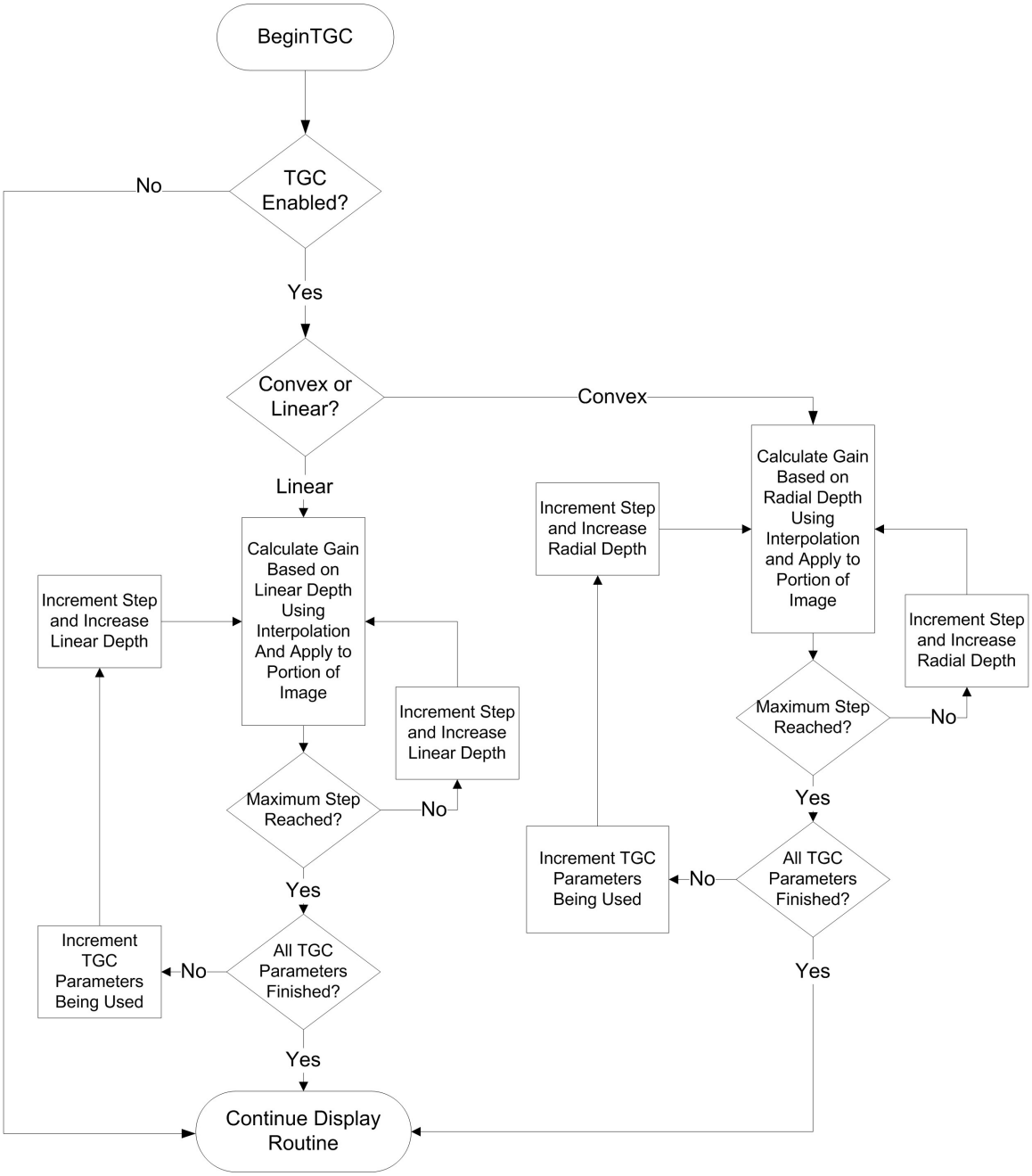
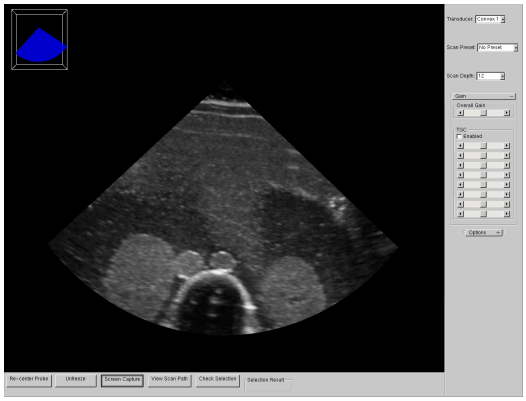
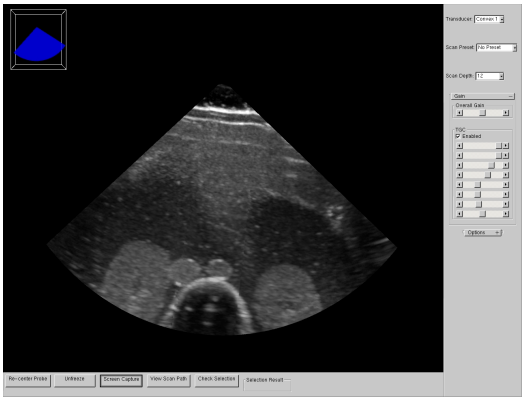


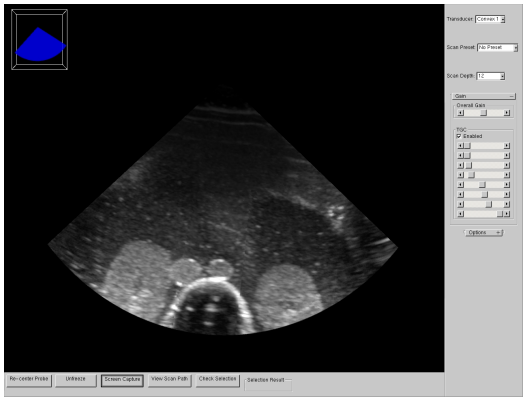
Figure 3.25: TGC Software Flowchart



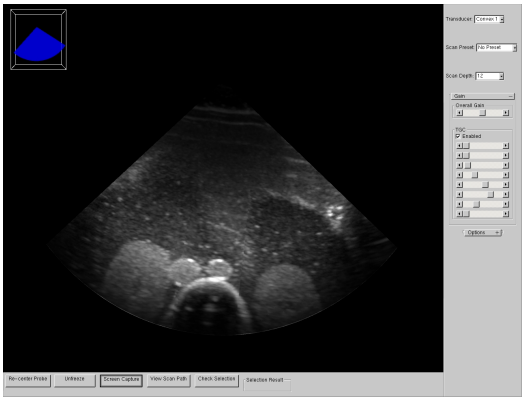
(a) No TGC Applied



(b) TGC Example 1



(c) TGC Example 2



(d) TGC Example 3

Figure 3.26: Time Gain Compensation Example Screen Captures

have any capability of adjusting focus, the focus remains the same as that of the selected image data volume.

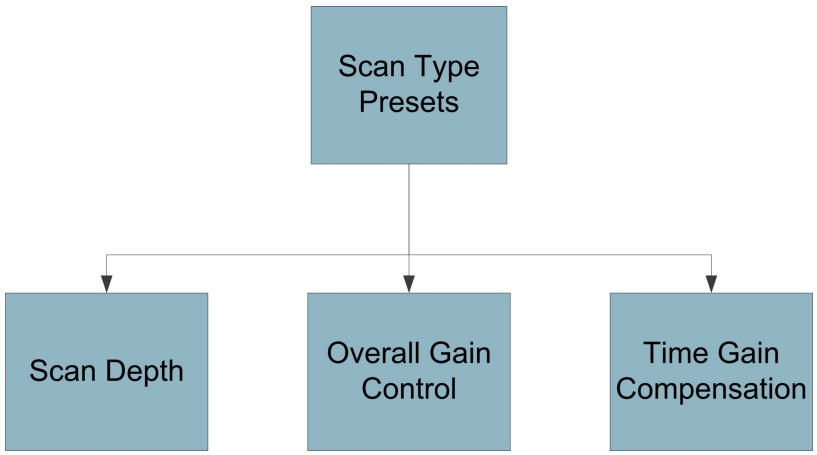


Figure 3.27: Scan Type Presets Block Diagram

Figure 3.28 shows a flowchart for the software implementation of the scan type presets. The function is part of the GUI callback function that is described in Section 3.3.2, and is called immediately upon a change to the current preset. First a conditional expression checks which preset was selected. Next, the TGC, gain, and depth values are set by modifying their respective variables. After these variables have been set, it is necessary to call the GUI update function to ensure that these changes are reflected in the on-screen parameters of the GUI. Finally, the depth is updated by calling the depth update callback, which propagates the depth value change to the rest of the program. After these operations have been completed, the program resumes at the point it left off.

Presets have been implemented for four selected scan types: No preset, Abdominal, Gynecological, and Prostate. "No preset" is simply a blank preset that allows the user to revert back to a default state. Table 3.4 shows the parameters associated with each of the implemented presets. These values are based on approximations of the preset values used in the Terason software for the equivalent scan types. Examples

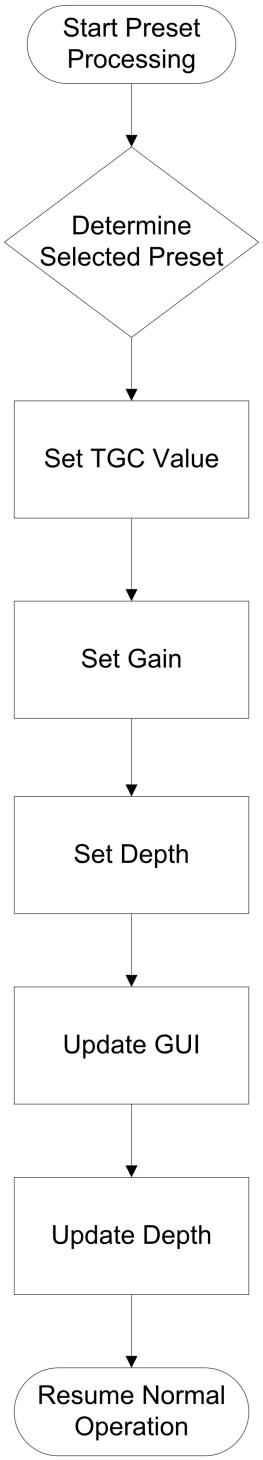


Figure 3.28: Scan Type Presets Software Flowchart

of these presets being applied are shown in Figure 3.29.

While only four different presets have been implemented at this point, additional presets can be added simply by defining their characteristics in the code.

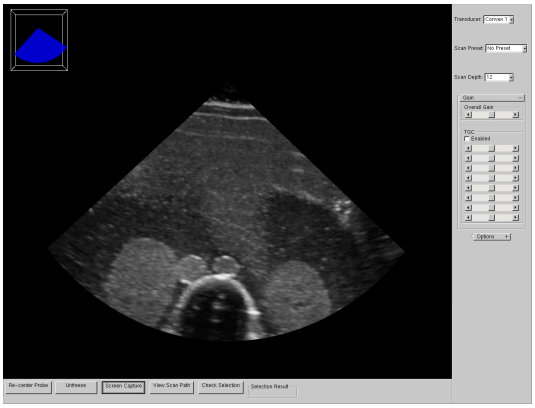
Table 3.4: Parameter Values for Implemented Scan Type Presets

Preset	**Depth**	**Gain**	**TGC[0-7]**							
No Preset	12 cm	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Abdominal	16 cm	1.0	0.4	0.4	0.6	0.8	1.0	1.0	1.1	1.1
Gynecological	12 cm	0.85	0.6	0.4	0.65	0.9	1.0	1.1	1.1	1.05
Prostate	12 cm	0.9	0.5	0.55	0.6	0.8	0.95	1.05	1.1	1.05

3.4.6 Navigational Display

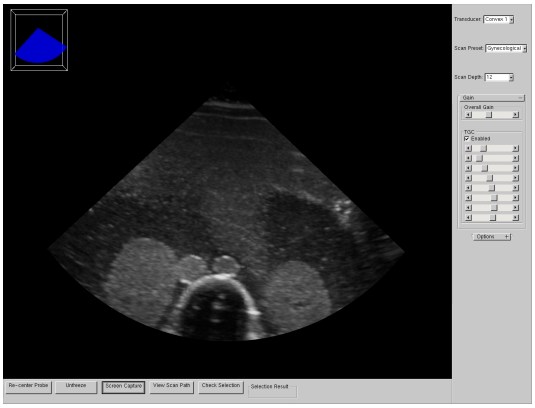
A navigational display was added to show the orientation of the current view in reference to the overall volume. This display shows a wireframe outline of the volume being viewed, along with a translucent object in the shape of the slice being viewed. This provides a reference showing where the currently viewed slice resides within the overall volume. The display is automatically adjusted to account for different probe shapes and depths as they are modified for the main display. The same stencil shapes that are generated for the stencil buffer of the main display are used to determine the shape to be drawn to this display. Figure 3.30 shows this display, which is located in the upper left corner of the display window.

While this might not be essential in the final system, it is very useful for development and testing purposes. It is especially important when working with the *SpaceNavigator* input, because it is not readily apparent where the 2D slice through the volume is located. Additionally, it is useful for debugging purposes, to ensure that the proper view is being displayed for a given sensor position.

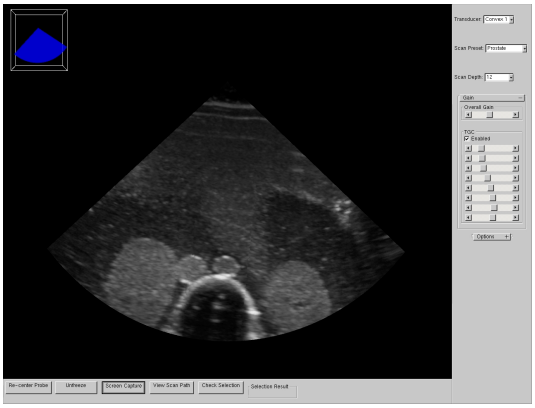


(a) No Preset

(b) Abdominal Preset



(c) Gynecological Preset



(d) Prostate Preset

Figure 3.29: Screen Captures of Different Scan Presets

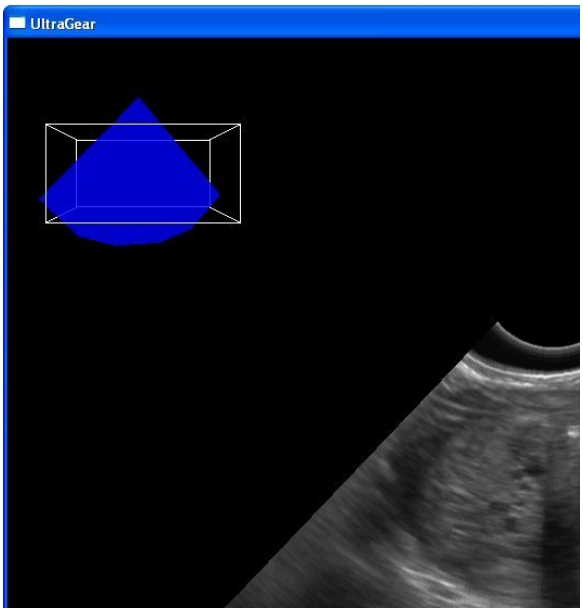


Figure 3.30: Navigational Display

3.4.7 Freeze Display Functionality

The capability to freeze the display at any time is a useful feature that is included in most ultrasound systems. The current image can be instantaneously frozen to permit image captures, analysis, and measurements. For the simulation system, this is especially useful for some of the training facilities, such as selection of regions of interest, which will be described in Chapter 5. This feature is not to be confused with still image capture, which stores an image of the current view and is described in Section 5.4.

Implementation of this feature was fairly straightforward. When the user presses the 'Freeze Display' button, a flag called *freeze* is set, which tells the *trakSTAR* read function not to update the values. A value of true means that the screen is frozen, while a value of false indicates normal operation. Without any incoming position values being used, it is as if the transducer is being held still and no screen movement occurs.

When the display is frozen, the button is changed to say 'Unfreeze Display'. When clicked, the flag is set to false and normal operation resumes. The button text is also changed back to 'Freeze Display'.

3.5 Conclusions

The additional simulation features and user interfaces described in this section create a far more realistic training environment. The scanning features provide important adjustments such as probe geometry selection, depth setting, and gain control. These features are nearly universal in commercial systems and have come to be expected by users. The navigational display and ability to freeze the display are additional useful features that help to enhance the overall experience.

The graphical user interface provides a simple, intuitive interface to all of these features and the learning assessment features that will be described in Chapter 5. Without the GUI, all of the features would be far less useful and would be difficult for users to access during a training session.

The hardware user interface is absolutely essential for this to be a realistic training system. Without the physical scanning interface provided by the manikin and sham transducers, this training system would be little more than a video game. The tactile interface to the images on the screen provide a true sense of realism and make for a very convincing training experience.

Chapter 4

Simulation Data Generation

4.1 Introduction

A necessary requirement of this system is a methodology for generating libraries of data for users to work with. To give the user complete freedom in their simulated scans, it is necessary to have data sets consisting of image volumes that are larger than a single sweep of the transducer, so that the user is not limited to scanning a single area along a fixed path. For example, a sonographer may choose to scan at multiple angles across a patient to see different views of a given feature. Furthermore, some large organs and features may require multiple sweeps to capture in their entirety. To generate effective simulation data, it is necessary to develop an appropriate scanning method and a procedure to combine data from multiple sweeps of the transducer.

4.2 Requirements For Generating 3D Image Volumes

The amount of data that can be acquired with a single sweep of an ultrasound transducer is limited to the width and angular range of the transducer. This is

generally not enough data to provide complete freedom in a simulation. To overcome this limitation, it is necessary to be able to acquire multiple 3D image scans and then merge, or register, them into a single, contiguous volume.

Figure 4.1 shows a human prostate gland as an example of how an image volume may be composed from multiple overlapping sweeps. The three rectangles represent the separate sweeps that could be taken where the arrows show the scan direction. It can be seen that a single sweep with a transducer of this size would not be wide enough to encompass the entire gland, necessitating the use of multiple sweeps. Overlap between sweeps is necessary for calculating the precise alignment of the sweeps relative to each other to create a smooth transition between them. It should be noted that with an effective alignment process, the sweeps do not necessarily need to be parallel or the same size.

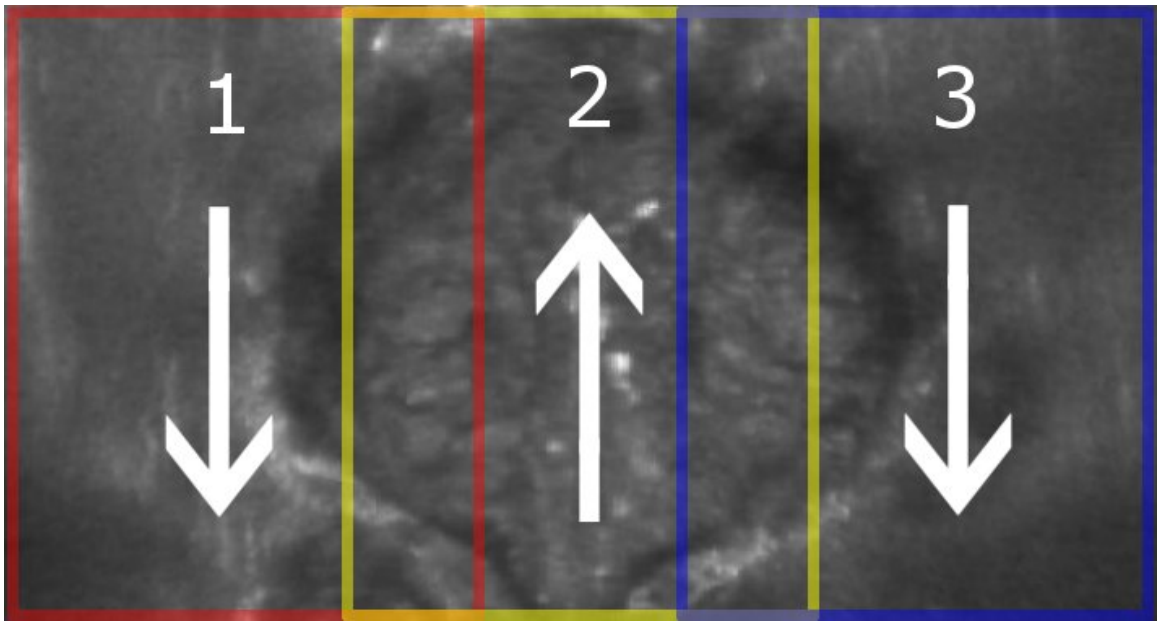


Figure 4.1: Example of multiple sweeps within a larger image volume

Acquiring multi-sweep data requires additional processing to ensure accurate alignment of the sweeps. Preparation of simulation data must be performed as a two-part

offline process. First, the image volume is acquired by a sonographer or researcher. The data must be captured in a manner conducive to merging multiple sweeps, but the method should remain simple for the ultrasound operator. Ideally there would be no constraints on the ultrasound operator; however, if this is not realistic, the constraints must be known, constant, and should not inhibit normal scanning.

Once data have been acquired, the multiple ultrasound sweeps need to be merged into a single large volume. It is important that the merging of volumes can be performed accurately, consistently and reasonably fast. Because this is an offline process, it does not need to be done in real-time, but it also should not take days for processing or require the use of a supercomputer. Ideally this process should be automated to some extent, requiring minimal user interaction. Excessive requirements for user interaction would introduce unnecessary human error factors and would complicate the process.

Figure 4.2 shows an overview of the data acquisition process. A patient or stabilized trauma victim is scanned by hand using a standard ultrasound system with a 6 DoF position sensor added. This scan is performed in a back-and-forth pattern, making multiple overlapping sweeps to cover a large area. These image and position data are combined on a computer to create 3D volumes. The multiple sweeps are merged together to create one large, contiguous 3D volume containing all of the data. The data can then be compressed and stored for later simulation use. Image volumes need only be generated once by the development team, and then can be used by an infinite number of trainees.

4.3 Registration of Multiple Sweeps

The merging or registration of multiple narrow ultrasound 3D volumes is an essential part of generating data for use in the system. Although the first part of the data generation process is the acquisition of ultrasound image volumes, the methods

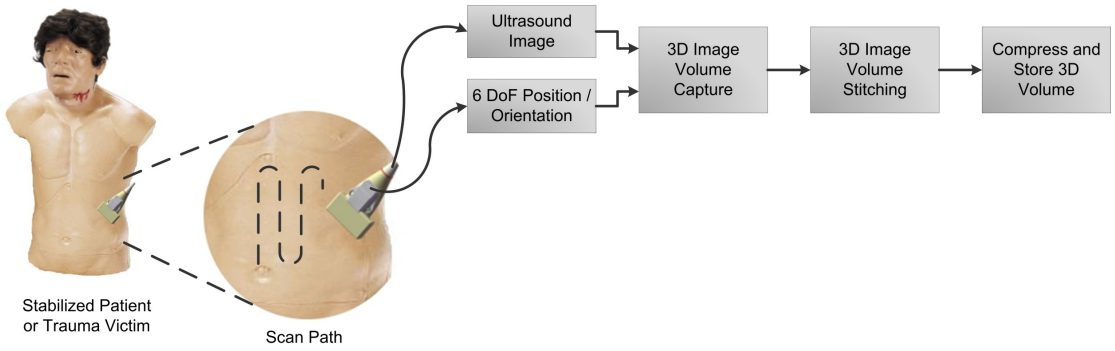


Figure 4.2: Block Diagram of Data Acquisition Process

of acquiring data are highly dependent on the specific requirements to facilitate merging data. For example, a given registration method may require a certain amount of overlap or may have requirements for the probe pressure used. Therefore, it is necessary to first discuss methods of merging volumetric data, which will help to determine the specific scanning requirements.

Splicing together two-dimensional images, such as making a panorama out of series of photos, has become fairly commonplace and there are many software packages available to accomplish this. However, registration of three-dimensional image volumes is a much less trivial problem and has not been studied nearly as much. This process is especially difficult for ultrasound data, which are generally noisy and contain view-dependent distortion.

Figure 4.3 shows a flowchart of a typical registration method that uses gradient descent. The general idea behind this type of registration involves making adjustments, then performing a comparison based on a specific measure of similarity and repeating this process for a large number of iterations until the optimal alignment is found. In some cases the end condition is a maximum number of iterations, while in others the algorithm stops when a certain similarity is achieved. If gradient descent is not used, then the process is similar except that a the same modification is always applied and the program tests all possible combinations to determine the optimal alignment. In

this case the end condition would be completion of testing every combination.

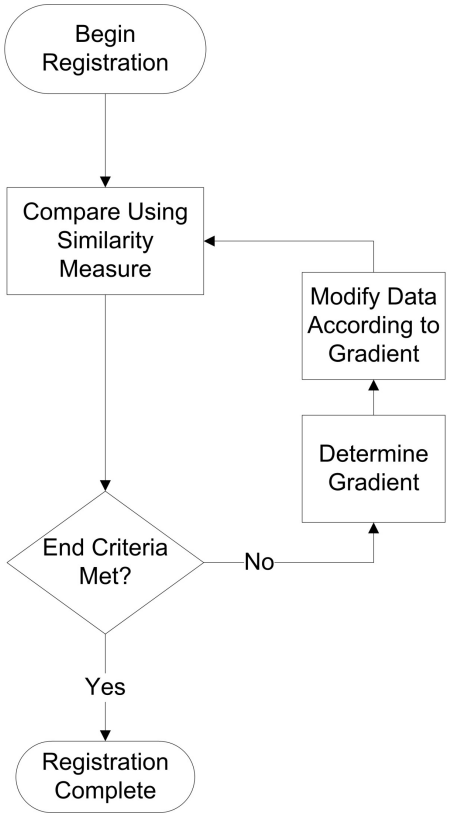


Figure 4.3: Flowchart of a Typical Registration Algorithm Using a Similarity Measure and Gradient Descent

The following sections discuss existing software options for merging multiple 3D image volumes. Most existing algorithms are designed for either multi-modal registration or inter-subject registration. Multi-modal registration is the alignment of images taken using different imaging modalities, for example a clinician may wish to align MRI and CT scans of a patient for concurrent viewing. Inter-subject registration involves the alignment of data from two different subjects for comparison purposes. Inter-subject alignment can also be extended to studying temporal changes in the same subject, for example, looking for new cancerous growths in breast tissue. In most cases, these algorithms are designed for aligning completely overlapping

images and not to make large volumes from smaller ones that only overlap slightly. The following sections discuss different algorithms that were studied and evaluated, including the chosen solution.

4.3.1 Similarity Measures

Most of the methods of registration that are described in this section use some type of comparison-based approach. To perform a comparison, it is necessary to have a certain similarity measure to base that comparison upon.

Similarity measures are typically statistical comparisons of two values. A number of different similarity measures can be used for comparison of 2D images and 3D data volumes, each having their own merits and drawbacks. These similarity measures are all very general and are not limited to comparison on 3D image volumes. These similarity measures perform voxel-wise comparisons and are typically used to register volumes by summing the comparison over a pair of 2D images or 3D volumes. Five different similarity measures are presented:

- Sum of Absolute Differences
- Sum-Squared Error
- Correlation Ratio
- Mutual Information
- Ratio Image Uniformity

Sum of Absolute Differences

Sum of Absolute Differences (SAD), formulated in (4.1), is a simple sum of the differences in gray scale intensity values between the two compared images, herein referred to as A and B. This method is simple, but does not have as high of a success rate as the other available similarity measures.

This measure must be normalized by the number of overlapping pixels to provide a consistent metric for a range of different overlaps. Since it is normalized by the number of overlapping pixels, $n_{overlap}$, Absolute Difference tends to favor alignments with less overlap, which can cause it to incorrectly determine the proper alignment. Another concern is that this algorithm has trouble determining similarity if the intensity values for a given feature differ greatly between the two images, which can occur when sweeps are taken from significantly different angles [13].

$$SAD = \frac{\Sigma|A - B|}{n_{overlap}} \tag{4.1}$$

Sum of Absolute Differences does, however, benefit from the fact that because of its simplicity it can be calculated very quickly, which is beneficial for reducing computational load. It is also not as sensitive to outliers as Sum of Squared Differences, described in Section 4.3.1 [14].

Sum of Squared Difference

Sum of Squared Difference (SSD) is another relatively simple similarity measure, described by (4.2). It differs only from SAD in that the square of the difference is taken, rather than the absolute value.

$$SSD = \frac{\Sigma((A - B)^2)}{n_{overlap}} \tag{4.2}$$

SSD has been shown to be the optimal similarity measure when the only difference between images is Gaussian noise, however this is rarely the case for medical imaging [14]. It is also a very simple similarity measure and can be calculated quite quickly.

Like SAD, it tends to favor reduced overlap because of the normalization that must be applied. Compared with SAD, SSD is more sensitive to a small number of outliers with high intensity values [14].

Correlation Ratio

Correlation Ratio [36] is a more general similarity measure. It assumes a functional mapping of intensity values, meaning that one of the images can be modeled based on the other. The similarity equation used for the correlation ratio is shown in (4.3). Unlike SAD, the Correlation Ratio does not favor alignments with reduced overlap.

$$CR = \frac{Var[E(B|A)]}{Var(B)} \tag{4.3}$$

The variance, which determines a variable's spread around its mean is used as a cost function. The variance is shown to be minimized by the conditional expectation, $E(B|A)$, defined in (4.4), where p(B—A) is the conditional probability density function of B assuming A.

$$E(B|A) = \int B * p(B|A)dy \tag{4.4}$$

The variance of the conditional expectation, $Var[E(B|A)]$, measures the part of B predicted by A [35]. A ratio between this term and the variance of B is taken to compare the prediction to the overall variance of B. The correlation ratio can take on a range from 0 to 1, with a value of 1 occurring when the images are perfectly correlated. Additional details about this algorithm are described by Roche *et al.* [35, 36].

Mutual Information

Mutual Information (MI) [49] is a very general similarity measure. MI is a measure of how much information one image contains about another. Therefore, the goal for MI is to maximize the MI value. This measure is an arbitrary mapping of intensities, represented by (4.5), where H(X) is defined as the entropy of image X and H(X,Y) is the joint entropy between two images. Entropy can be interpreted as a measure of the variability of a random variable. The mathematical definitions of entropy and

joint entropy are shown in (4.6) and (4.7). This essentially compares the degree of dependence between the images or the amount of information that one image contains about the other [20].

$$MI = H(A) + H(B) - H(A, B) \tag{4.5}$$

$$H(x) = -\int p(x) \ln p(x) dx \tag{4.6}$$

$$H(x, y) = -\int p(x, y) \ln p(x, y) dx dy \tag{4.7}$$

The $H(A)$ term is the entropy in the image being compared and the $H(B)$ term is the entropy of the part of the second image into which A projects. The joint entropy term, $H(A, B)$, takes effect when A and B are functionally related and favors cases where A and B are well aligned [48]. The details of this algorithm are described in further detail by Maes et al. [20].

Ratio Image Uniformity

Ratio Image Uniformity (RIU) is an algorithm that was introduced by Woods *et al* [51]. It is also known as Variance of Intensity Ratio (VIR) [14].

The equation for RIU is shown in (4.8) and the definitions of R and $\bar{R}$ are defined in (4.9) and (4.10), respectively. R is a ratio image derived from images A and B. The goal of this similarity measure is to find a transformation that maximizes the uniformity, or normalized standard deviation of voxels, of this ratio image [14].

$$RIU = \frac{\sqrt{\frac{1}{n_{overlap}}}\Sigma(R - \bar{R})^2}{\bar{R}} \tag{4.8}$$

$$R = \frac{A}{B} \tag{4.9}$$

$$\bar{R} = \frac{1}{n_{overlap}} \Sigma R \tag{4.10}$$

This algorithm is typically used for intramodality registration and has frequently been used for registration of Positron Emission Tomography (PET) and MR images.

4.3.2 Image Registration Toolkit

The *Image Registration Toolkit* [37] is an open source registration program developed by researchers at Imperial College in London. Source code and pre-compiled windows binaries are provided, but aside from a few command line examples, minimal documentation is available. The registration program uses mutual information as a similarity measure [38]. Further information about the algorithms used in this program are described in papers by the authors of the software [38, 39, 40].

The *Image Registration Toolkit* was tested with two overlapping halves of an ultrasound image volume. It was able to register the volumes with an affine alignment, but the result was not perfectly aligned. Non-rigid alignment using this program is too slow to be practical and takes hours to run for a 100 MB volume on a fast desktop computer. Although this program could be made to work to some extent, it does not appear to be an effective option.

4.3.3 AIR 5.2.5 (Automated Image Registration)

AIR is an open source program developed by researchers at the University of California Los Angeles and is primarily designed for multi-modal registration of MRI and PET data. It is capable of performing linear, affine, and nonlinear registration. It also includes a large suite of tools for registration and manipulation of 3D volumes. AIR is distributed as source code only and can be compiled under Linux or Windows. The authors have done a fairly good job with the documentation and provide useful information and examples [52].

AIR uses the Ratio Image Uniformity similarity measure, which was developed by the authors of this software [14].

Tests were performed with AIR by attempting to align separated halves of a single volume in the same manner as Image Registration Toolkit. Initial tests proved ineffective and it is likely that better test data was required. Upon discovery of more effective registration methods, further testing was abandoned.

4.3.4 University of Cambridge's *Stradx*

Stradx [29] is an acquisition and visualization system developed specifically for ultrasound by researchers at the University of Cambridge. It is designed to acquire and manipulate ultrasound data with 6 DoF position information. This software is very well documented and numerous papers have been written describing in detail the technical aspects of its various capabilities [13][30][31].

Most of the other algorithms for registering 3D volumes that have been discussed involve 3D non-rigid deformations, which are very computationally intensive processes. Furthermore, the previously discussed algorithms focused on MRI, CT, and PET modalities with no attention to ultrasound. Their approach uses a two stage process; they start with a non-rigid inter-B-scan alignment within each sweep and then perform a rigid translational 3 DoF alignment between the two sweeps. They have optimized this algorithm for speed, and it performs very quickly. These algorithms have been developed specifically for use with ultrasound data and take into account the fact that ultrasound data have much more noise and speckle than other modalities and can be significantly view-dependent.

Alignment Process

The first step in the two-stage *Stradx* alignment process is to align the B-scans within each sweep to create well-aligned sweeps. It is very difficult to maintain con-

sistent probe pressure through an entire sweep, which results in compression of the features being scanned. This alignment process compresses and expands each successive B-scan to match with the one before it, which has the result of causing all of the B-scans to appear as if they were scanned with constant probe pressure. It also has the added benefit of removing respiratory artifacts caused by breathing motion. Non-rigid alignment of 3D data is normally computationally intensive; however, this can be accomplished fairly rapidly in *Stradx* because the process only involves aligning pairs of 2D B-scans to each other, rather than performing a full 3D alignment.

After the scan planes have been aligned within the individual sweeps, multiple sweeps can be aligned to each other. Due to the mechanics of scanning, organs and features are likely to be moved around by probe pressure, but are not typically rotated; therefore, only translations are necessary to line up sweeps [13].

A user-specified dividing plane that cuts through the overlapping region of the two volumes is defined using a graphical tool. The two volumes being merged are then compared at this dividing plane. *Stradx*'s multi-sweep alignment tool performs a multi-resolution search in three dimensions. This search is undirected, meaning that it searches all possible alignments rather than following gradients, because the authors found that false alignments are possible when a directed search is used [13]. The process begins by smoothing and down-sampling the two images it is comparing, then translates them in the x and y directions, comparing at each alignment using one of three user-selectable similarity measures to find the best alignment. One of the volumes is then shifted in the z axis and checked again. The program keeps track of the best alignment found during all stages. After checking all alignments within the user-specified range, the resolution is increased and the process is repeated, starting from the best correlated alignment found at the previous resolution. Four different resolutions are used in this alignment process. *Stradx* provides a graphical interface to show the user the alignment results, so that the alignment parameters can be adjusted if necessary.

Stradx permits the user to select between three different correlation similarity measures. In most cases these perform very similarly; however, if the process fails on a given pair of volumes using one algorithm, it may be helpful to try others. All three of these methods are common, well-proven similarity measures. These similarity measures consider the two images as random variables and use different types of probabilistic analysis to determine alignment. The three options for correlation functions are:

- Sum of Absolute Differences (SAD)
- Correlation Ratio (CR)
- Mutual Information (MI)

These similarity measures were described in more detail in Section 4.3.1. None of the similarity measures are perfect for all sets of volumes, but it is usually possible to find one that is effective. In general, Mutual Information and Correlation Ratio tend to be the two best choices. Selection of a similarity measure is simply a matter of choosing one from a drop-down box in the alignment window (shown in Figure 4.4), which allows a user to test all three measures to select one that produces a visibly correct alignment for the current problem.

4.3.5 University of Cambridge's *Stradwin*

A more recent tool from the group at University of Cambridge is called *Stradwin* [28]. This is a more updated program that is meant to be a replacement for *Stradx*. Until very recently, *Stradwin* was not a viable option for this system's data capture needs, because it did not implement any multi-sweep alignment capabilities. However, in *Stradwin* 3.4, the capability to merge multiple sweeps was added.

Stradwin's multi-sweep alignment is significantly different than that of *Stradx*. Instead of performing mathematical comparisons between the sweeps, it simply lines

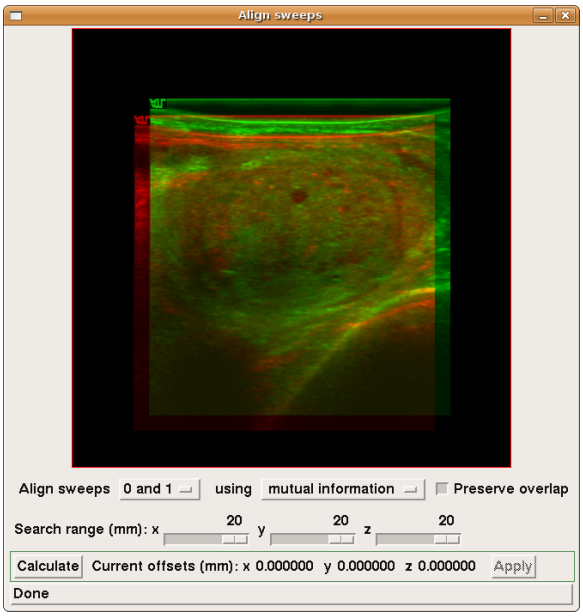


Figure 4.4: Stradx Alignment: Views of sweep alignment at the dividing plane

them up based on the recorded position information. Using the captured 6 DoF position and orientation data, sweeps are aligned in 6 DoF, accounting for both translations and rotations. Although it seems this would be much less accurate, in many cases it is possible to get very good alignment without any additional processing. As no comparison takes place, no similarity measure is used in this process.

Stradwin also implements a modified scanning process designed for multiple sweep acquisition, called 'multi-sweep gated' mode. In this mode, recording starts when the probe has been held still for about a second and stops when the probe is held still again. When the probe is lifted up and moved over, then held still again, another sweep is created and recording resumes. This can be repeated for any number of sweeps to form a multi-sweep volume. This avoids the need to manually specify the extents of the sweeps in the post-processing phase.

Stradwin imposes minimal requirements upon how the data are actually scanned. The one important requirement is that at least one B-scan must be taken with minimal probe pressure. This provides a standard to which the rest of the scans can be

compared to correct for probe pressure artifacts.

While acquisition using *Stradwin* is the most effective way of obtaining data, it is not without its limitations. The primary downside to *Stradwin* is that it requires a position tracking system. It also requires either a Terason system or a computer with frame capture capabilities and a standard ultrasound system. Position sensors are quite expensive, with the most inexpensive models starting at about $2500 [16]. While using *Stradwin*, which is a free program, would be simple in a laboratory environment, it may not be quite as easy to require an ultrasound operator to use *Stradwin* on patients. They would need to first be trained in the use of the software. The cost of the position sensor is the primary concern and as long as that is not prohibitive, the other limitations are fairly negligible.

Stradwin provides simple and intuitive scanning and alignment, while still being able to accurately align multiple sweeps. Because the system uses a position tracking system for the training system, the tracking system will be available for scanning as well. For these reasons, it was chosen as the best option for the image capture method for this system.

4.3.6 Free-Form Deformation Registration Code by Dr. Joyoni Dey

Although *Stradwin* was chosen as the preferred method of capturing and stitching image data, some practical difficulties arose with the use of *Stradwin* for stitching, which are described in Section 6.2. Because of these difficulties, some additional options for multi-sweep stitching have been evaluated. Dr. Joyoni Dey, a researcher at the University of Massachusetts Memorial Medical School, has developed a method of stitching multiple 3D data volumes [10]. This method is aimed toward stitching of MRI volumes, but can be extended to work for ultrasound data. Additionally, it is designed primarily for registration of two similar MRI volumes of the same subject

that are not perfectly aligned.

This code works by first aligning the two volumes rigidly using a 12 degree of freedom affine alignment. This involves aligning the two volumes by translation, rotation, scaling, and skewing. Following the affine alignment, a free-form deformation is performed to non-rigidly align the two volumes. For both of these alignments the Sum of Squared Difference similarity measure is used.

Some initial work was performed with this code, but due to time constraints no significant results were able to be obtained. The alignment process takes quite a while to run, which proved to be a hindrance to testing. For a typical registration, affine registration requires about half of a day of processing on a dual-core server and free-form deformation requires over a day of processing.

The initial testing was performed with MRI test volumes, so ultrasound data has not been tested yet. Additional testing using ultrasound data will be necessary to determine if this might be a viable option for registration of multiple volumes.

4.4 *Stradwin* Capture Details

Whether or not *Stradwin* is used to perform volume stitching, it is a clear choice for data capture. *Stradwin* is capable of interfacing with nearly any ultrasound scanner, including the Terason portable ultrasound systems. By combining the image data from an ultrasound scanner with the position and orientation data from a 6 DoF tracking system, 3D image volumes can be generated in real time. *Stradwin* is a powerful utility that offers many useful features for 3D ultrasound.

Figure 4.5 shows a block diagram of the *Stradwin* capture process. The subject is scanned with a transducer with attached 6 DoF tracking system, making multiple overlapping sweeps of the transducer. Scanning produces streams of image data and position and orientation values, which must be aligned with each other through a calibration process. The calibrated data can be used to generate a 3D image volume.

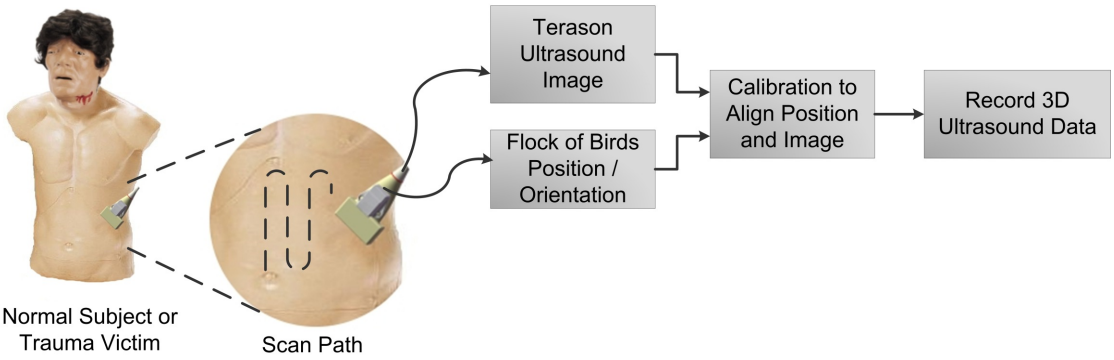


Figure 4.5: Block Diagram of *Stradwin* Capture Process

Although *Stradwin* makes the data acquisition process fairly simple, there are still several necessary details and idiosyncrasies that must be known. Appendix F describes details about the *Stradwin* setup and data capture process.

4.5 *Stradwin* Calibration

Combining position data with ultrasound image data for 3D imaging requires calibration to properly relate the coordinate system of the position sensor to the images that are recorded. For this reason, it is essential that a calibration process be performed before attempting to acquire data with *Stradwin*. In fact, without a valid calibration, *Stradwin* will not permit recording of ultrasound data.

Stradwin implements a fairly simple, semi-automated calibration process in which the user sets some parameters and then goes through a series of specific probe motions while *Stradwin* detects lines formed by the bottom of a water bath. *Stradwin* provides documentation on calibration; however, the procedure in Appendix G may prove helpful, as the instructions are oriented toward this specific configuration and include some additional insight achieved through conversations with the developers of *Stradwin*.

Figure 4.6 shows the four different coordinate systems that are aligned by the

Stradwin calibration process. T is the coordinate system of the tracking system transmitter and R is the coordinate system of the tracking receiver. C represents the coordinate system of the volume being scanned. P is the image coordinate system, which represents the scan plane and the image seen on the monitor.

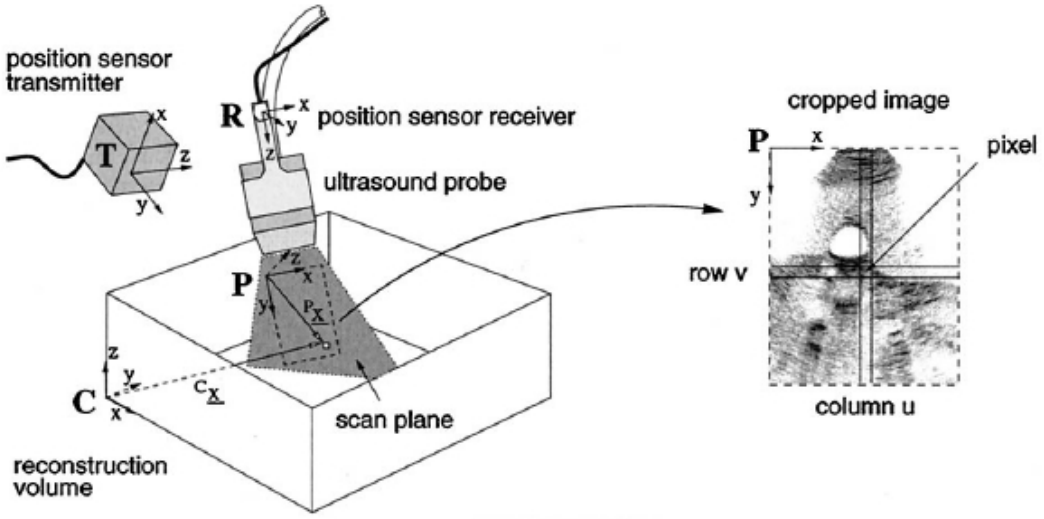


Figure 4.6: Diagram of Coordinate Systems Aligned by *Stradwin* Calibration [32]

Equation (4.11) shows the series of transformations that relate the coordinates of the scan image, $\underline{\mathbf{x}}$, to the volume being scanned. ${}^{C}\mathbf{T}_{T}$ is a transformation between the reconstruction volume and the location of the tracking transmitter and is used to remove any offset between the captured image volume and the tracking transmitter. ${}^{T}\mathbf{T}_{R}$ is the transformation between the tracking transmitter and tracking receiver, which is what is determined by the tracking system. Finally, ${}^{R}\mathbf{T}_{P}$ is the transformation between the receiver position and the scan image, which is what must be determined through calibration.

$$ {}^{C}\underline{\mathbf{x}} = {}^{C}\mathbf{T}_{T}\,{}^{T}\mathbf{T}_{R}\,{}^{R}\mathbf{T}_{P}\,{}^{P}\underline{\mathbf{x}} \tag{4.11} $$

These transformations are defined using translations and fixed angles. The three

rotations are specified as (α, β, γ), which correspond to rotations about the z, y, and x axes, respectively. To properly perform rotations, these must be executed in γ, β, α order [32].

In general, a 6 DoF transformation from coordinate system I to coordinate system J takes the form of Equation (4.12).

$$
{}^{J}T_{I}(x,y,z,\alpha,\beta,\gamma) = \begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) & x \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) & y \\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma) & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.12}
$$

The calibration procedure produces two points per captured image that define the ends of the captured line. Calibration using these values is performed using the Levenberg-Marquardt Algorithm [24], which is an iterative numerical algorithm for multivariate optimization.

4.5.1 Improved Calibration Using a Mechanical Fixture

Stradwin's calibration process is fairly simple to perform, but achieving very accurate calibration is difficult because the motions are unconstrained. Additionally, the bottom surface of a water bath is not a very controlled surface. The designers of *Stradwin* came up with a novel approach to improving the efficiency and effectiveness of the calibration process. They designed a calibration fixture, shown in Figure 4.7, which they call the Cambridge Phantom [13]. The calibration fixture is used in a water bath, but the transducer is held in place pointed at the top of a thin brass bar rather than the bottom of the water bath. Because a brass bar is being viewed, detection is much easier and the quality of the water bath is not important. This fixture constrains the movements of the transducer to only one or two degrees of freedom at a time, making it much easier to isolate the required motions.

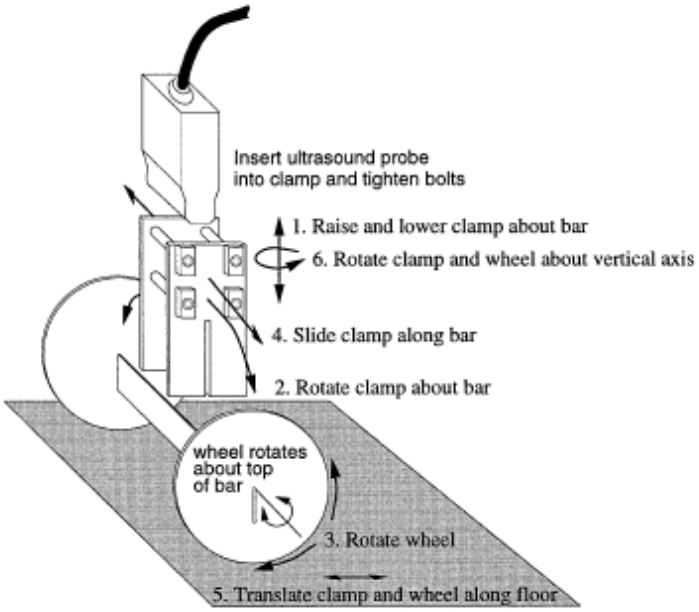
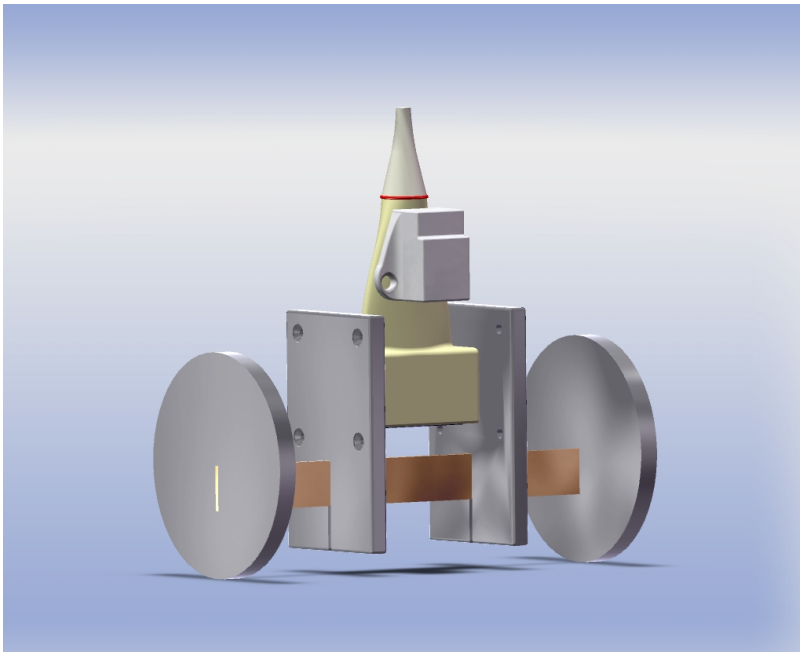


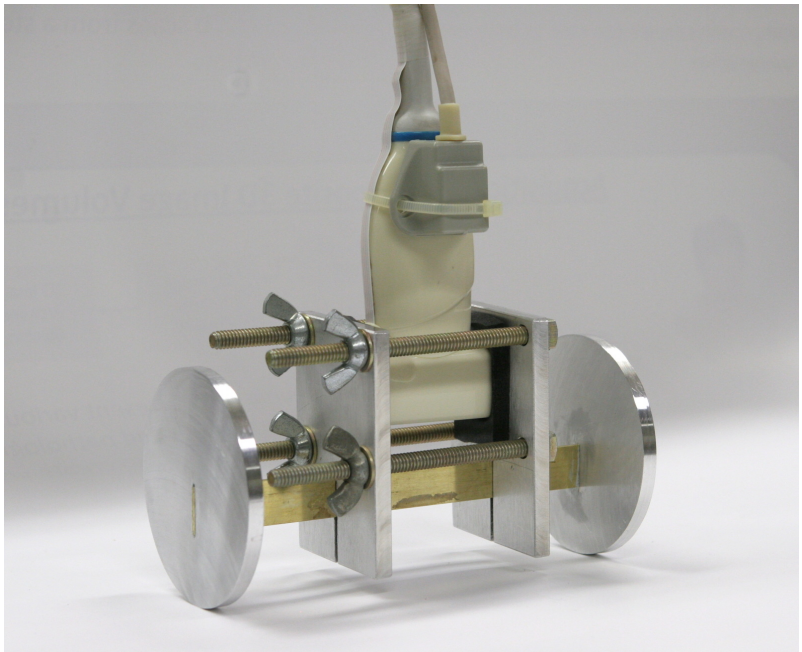
Figure 4.7: Cambridge Phantom [13]

Unfortunately, the purchase cost for one of these Cambridge Phantoms made by the *Stradwin* team is £1600 (approximately $2300), which is prohibitively expensive. However, after studying their papers and diagrams, it was determined that it would be feasible to design and produce a similar calibration fixture in-house.

CAD models of the phantom and required parts were drawn up in SolidWorks. The necessary materials were able to be purchased for a total cost of less than $100 and the parts for the fixture were machined in-house at no cost. Figure 4.8 shows the CAD model of the calibration fixture with a transducer and *Flock of Birds* sensor held in the clamp and a photo of the actual manufactured fixture. Figure 4.9 shows a CAD drawing of the calibration fixture, and indicates its dimensions in inches. The manufacturing process for this calibration fixture is described in step-by-step detail in Appendix M.



(a) 3D CAD Model



(b) Photo of Manufactured Fixture

Figure 4.8: WPI Calibration Fixture Images

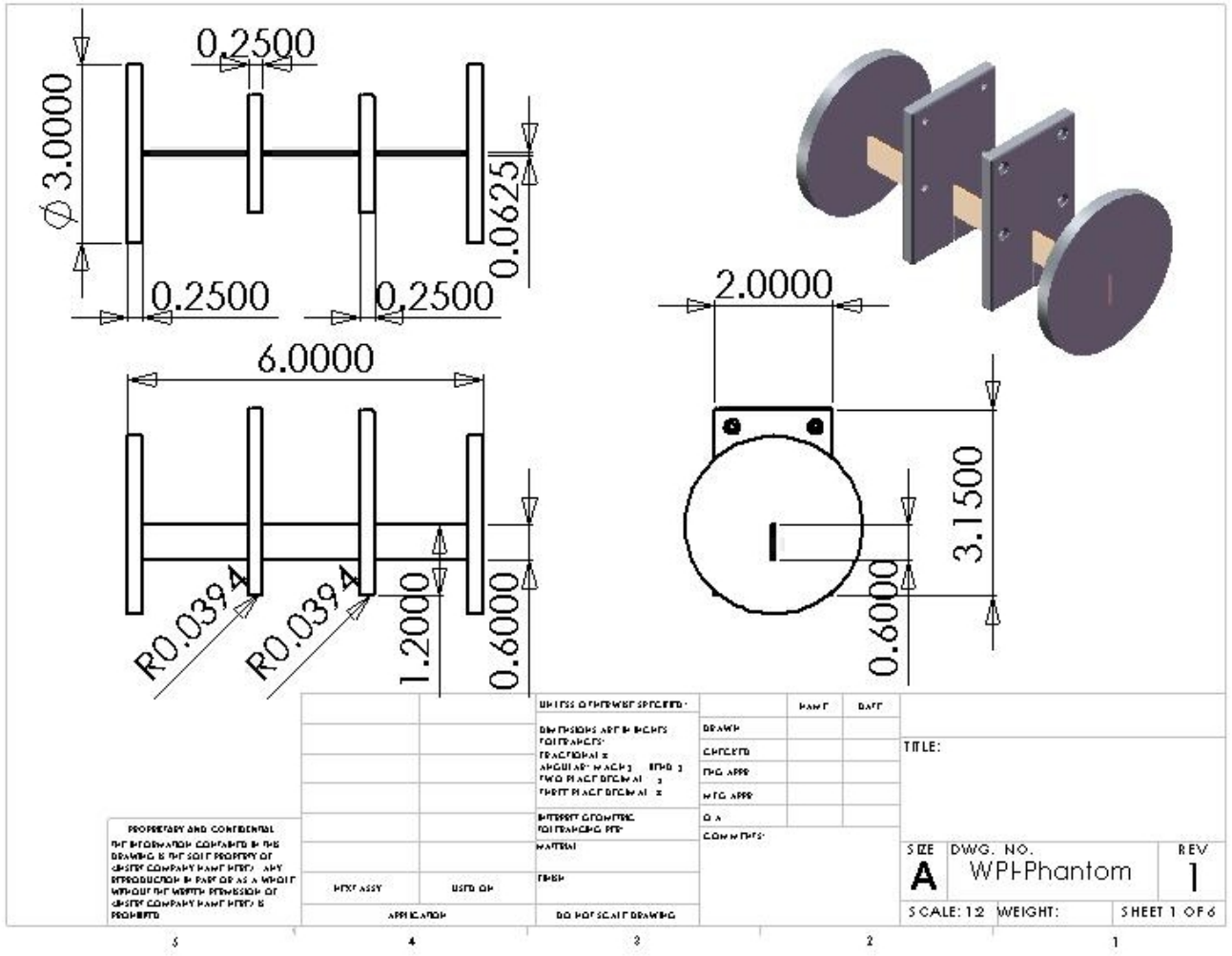


Figure 4.9: CAD Drawing of WPI Calibration Fixture

The materials for the calibration fixture were carefully selected to minimize interference with the magnetic tracking system. Although the *Flock of Birds* is relatively immune to interference from nearby metal, it is still affected to some degree, and some materials are worse than others. Ascension Technology Corporation specifies that aluminum and 300-series stainless steel are generally not problematic, so these materials were used whenever metal was necessary. The main parts of the fixture are machined from aluminum with a brass strip used for the bar that is scanned. In the calibration fixture's current state, the bolts and washers are made of 300-series stainless steel and the wing nuts are nylon. Because these interference concerns were kept in mind during material selection, the affect on the performance of the tracking system is negligible.

4.6 Registration of Data Volumes to Manikin

After volumes have been created, some additional processing must take place to use the volumes with the simulation system. A volume must be appropriately scaled and aligned with the manikin so that the represented anatomical features are positioned correctly. The individual scans are registered and the overall resulting image volume is scaled to the size of the manikin.

Since this registration process needs only to be performed once for a given volume, it is currently achieved manually. It is assumed that the top of the image volume is the skin surface and that the volume is centered in the torso of the manikin. The assumption is also made that the torso shape of the patient from whom the data were captured is similar to that of the manikin. Thus, the only adjustments that are necessary are the isotropic scaling of the volume and the longitudinal offset within the torso. These two parameters are adjusted through trial and error to achieve an appropriate fit between the image data and manikin.

This process could be improved by the creation of an automated scaling and

offset process. One method of doing this would involve capturing a series of points to represent the surface of the patient in a similar manner to the method described in Section 5.3.2. This surface capture could be compared to a similar capture for the manikin to determine the necessary scaling and offsets. Additionally, through a non-rigid deformation process, it could be possible to account for significant differences in torso shape between the patient and manikin.

4.7 Image Data Header File

In order to display a 3D volume, it is necessary for the software to know some basic parameters, such as the dimensions of the volume. While volumes can be loaded by selecting the data file and manually specifying the parameters, the file loading process can be streamlined by using a header file that contains all necessary parameters and the name of the appropriate data file. The use of a header file also permits a larger number of parameters to be easily specified for any given volume and simplifies record keeping.

To accommodate the specific needs of this system, a custom header file was designed. This header has the extension '.ghd' and is an extension of the ITK '.mhd' header file format. The header file is an ASCII file containing a number of parameters, one of which is the filename of the data file, which should have a '.raw' extension. Table 4.1 describes the different parameters that can be specified in the header file.

Parameter	**Values**	**Description**
NDims	2,3	Specifies the number of dimensions for the image, this must be 3 for these volumes
DimSize	xxxx yyyy zzzz	Image volume dimensions in pixels
ElementDataFile	data_filename.raw	Name of the file containing the raw image data
Offset	x.xxx y.yyy z.zzz	Amount by which to offset the volume from the center of the tracking system, used to align data to the manikin
VolumeScaling	x.xxx y.yyy z.zzz	x, y, and z scaling factors to fit the volume to the manikin
FeatureN	x1 x2 y1 y2 z1 z2 x3 x4 y3 y4 z3 z4	Coordinates of features within the volume, specified as two rectangular bounding boxes by pairs of x, y, and z coordinates. N denotes the feature number.

Table 4.1: Header File Parameters

4.8 Conclusions

The capability to effectively generate simulation data is an important part of any simulation system. Although not readily apparent to the end user, this is very important for the developers, as providing an extensive library of data is essential for this system to be useful.

A variety of similarity measures have been presented and a number of options for data capture and registration have been evaluated. *Stradwin* appears to be a clear choice for data capture and was the initial choice for registration as well. Results of multi-sweep data captures from a phantom and human subject are discussed in Section 6.2.

Details of the *Stradwin* capture process have been presented to provide a clear description of the necessary steps for successful data capture. The calibration process is presented, along with details on the calibration fixture that was constructed to simplify the process.

Chapter 5

Learning Outcomes Assessment

5.1 Introduction

Learning outcomes assessment features are what make a training system more effective than a simulation system. While a simulation system can be a valuable tool for learning the mechanics of scanning, it does not generally provide any means of evaluating skills or improvement, so features have been added to this system that can assess the effectiveness of the training. These learning assessment tools provide opportunities for users to hone their skills and methods for instructors to assess whether trainees are improving.

By providing tools to help teach and evaluate students, the training system is made much more valuable. Students can be asked to identify and localize specific pathologies or traumas and be told whether they were correct. The scan path can be recorded and played back for analysis to determine whether trainees are scanning effectively. Still images can also be captured to provide permanent records and evaluate a trainee's ability to capture quality scan images. These tools serve to help the trainee learn and to provide an effective means for a trainer to assess learning outcomes.

5.2 Region of Interest Selection

An important training feature is verification of the trainee's ability to identify specific anatomical features or abnormalities and select them. This tool provides both the interface for localizing specific traumas or pathologies using the touch screen and the ability to determine if features were correctly selected.

The user can asked to scan for and locate a specific pathology or anomalous condition. Then, using the touch screen the user may select the specific feature within the scan plane. After the user has made their selection, the are graded on whether they correctly selected the specified anatomical feature.

Figure 5.1 shows a block diagram of the region of interest selection system. This is divided into three major parts: drawing feature objects, user selection, and automatic detection. Drawing feature objects involves reading the objects from the data set and drawing them relative to the volume being displayed. In most cases, these will be hidden and will not actually be displayed to the user. User selection is where the user draws a selection with the touch screen in an attempt to correctly select the desired region. Automatic detection involves determining if this selection was made correctly.

5.2.1 Inclusion of Features in Data Set

Figure 5.2 shows how regions of interest are specified in the data sat. The manikin skin surface shown represents the bounding area of the data volume. The red shape in the middle is a specific anatomical feature that trainees will be requested to identify. The inner black box that closely surrounds the region of interest is the inner boundary and the outer black box forms the outer boundary. For clarity, both bounding boxes reside outside of the region of interest in this figure, but the inner one could certainly be located within the desired region to permit the user to select right at the edge of the region of interest. The user is expected to create a selection that is positioned between the two bounding boxes, such as the yellow box shown in Figure 5.2.

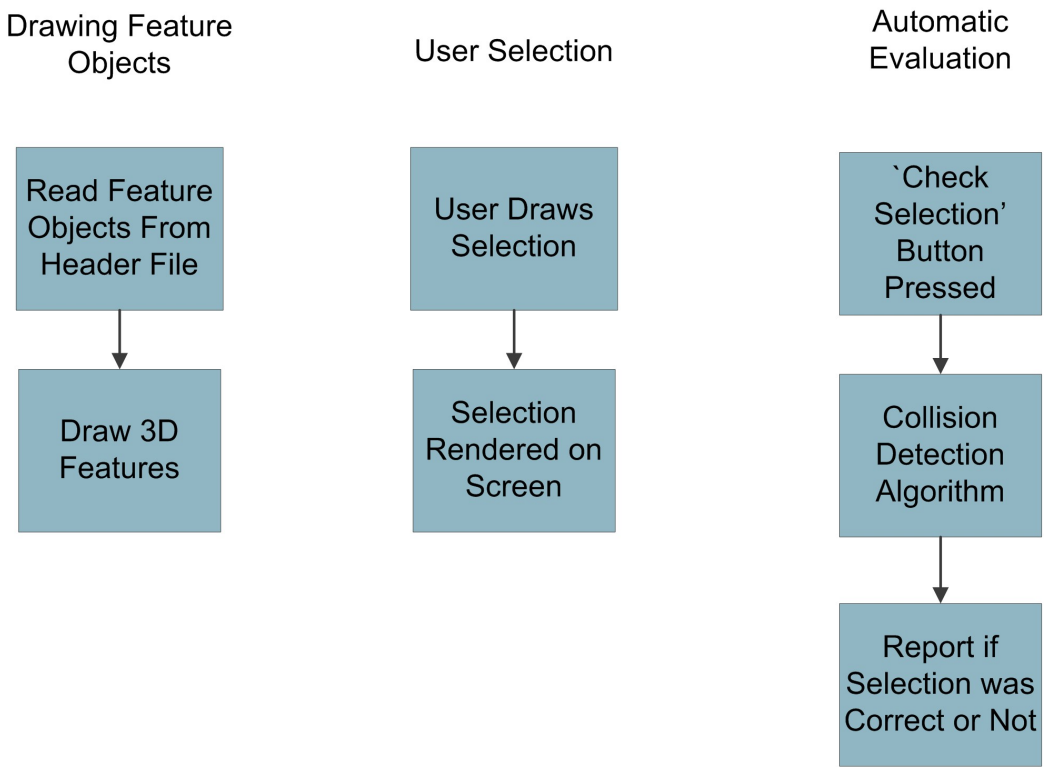


Figure 5.1: Region of Interest Selection Block Diagram

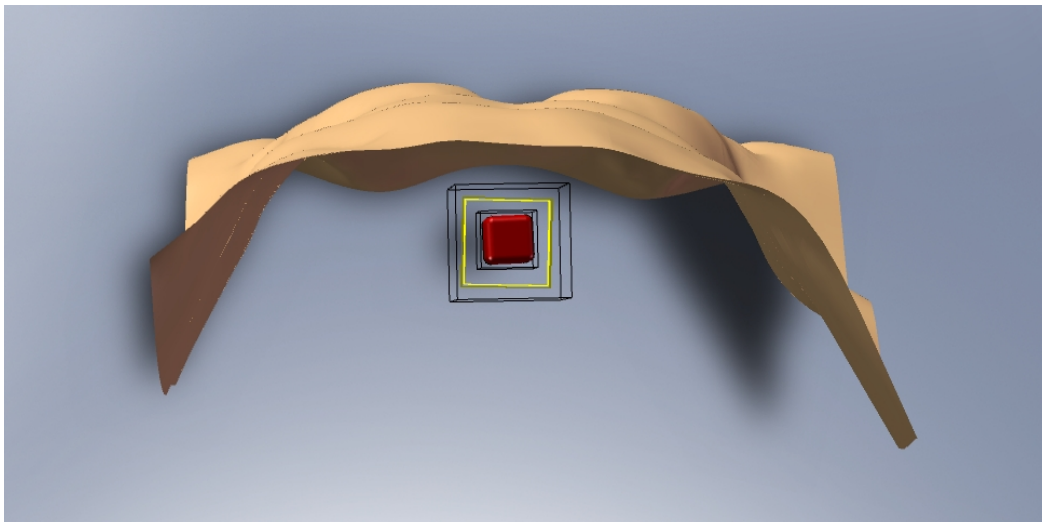


Figure 5.2: Feature Object Specification and Correctly Selected Region of Interest

Regions of interest can be specified in each data set in the form of *feature objects* containing a series of points that make up the outline of the bounding boxes. These features are specific to a given volume and are encoded in the volume's header file, which was described further in Section 4.7. These features are currently specified by twelve floating point values in the form of four (x,y,z) points, which represent two diagonally opposite corners for each of two rectangular prisms.

A feature is specified in the header file as "FeatureN = x1 x2 y1 y2 z1 z2 x3 x4 y3 y4 z3 z4", where N is the number of the feature within the data set. The inner bounding box is formed by the points (x1,y1,z1) and (x2,y2,z2). The outer bounding box is specified by the points (x3,y3,z3) and (x4,y4,z4). For initial development, currently only one feature ('Feature1') may be specified, but with further development, any number of features could be specified for a given data set as long as they are given unique and serially increasing numbers. Although the bounding boxes are currently specified as rectangular prisms, this is only to simplify initial design and testing, and the software could be modified to work with any shape. Other shapes could be used by specifying proper points and setting up an appropriate drawing function.

The features are normally drawn as invisible objects so that the user will not know where the features are located. However, an option is included to display the specified features for debugging purposes, which was described in Section 3.3.3.

5.2.2 User Selection

During a training session, trainees may select specified regions of interest using the touch screen, as shown in Figure 5.3. By touching the screen and dragging one's finger, a user may specify a rectangle, circle, or oval around the object they have observed. The specific shape depends on the software settings, as multiple selection shapes have been developed.

When a touch or mouse click is registered, a new object is created at the current drawing depth. The initial point that is touched is stored as the shape's starting

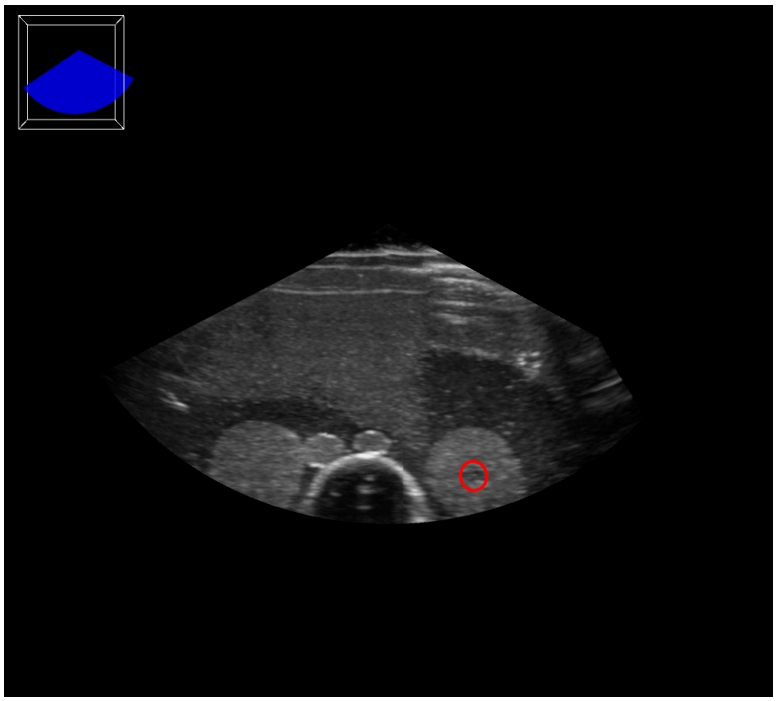


Figure 5.3: Region of Interest Selection Using an Oval

point and the second point also starts out at this location. As the finger is dragged, the second point is updated to the new location, providing two distinct points. The software continuously updates the shape with every screen update, so the user can see the shape change as they move their finger. When the finger is lifted up, the selection is saved and continues to be displayed.

For simplicity of initial automatic detection, a rectangular region is used for the selection to match up easily with the rectangular prism bounding boxes. This is created as a thin rectangular shell that is open on the front and back ends. It is specified using the two points captured by the user selection.

Alternatively, an oval shape can be used. To create an oval-shaped selection object, a circle of diameter 1.0, specified in OpenGL units, is created. The translations and rotations described in equations (5.1)-(5.5) are applied to scale this to the correct size and properly position it. The center of the shape is positioned at the arithmetic mean

of the starting point and current point. The dimensions of the circle are scaled to the absolute value of the difference between the points to form an oval with elongation based on the angle of the selection. The z-axis is generated based on the current position of the sham transducer, to ensure that it lines up with the view seen by the user.

$$translate_x = \frac{x1 + x2}{2} \tag{5.1}$$

$$translate_y = \frac{y1 + y2}{2} \tag{5.2}$$

$$r = \sqrt{|y2 - y1|^2 + |x2 - x1|^2} \tag{5.3}$$

$$scale_x = |x2 - x1| \tag{5.4}$$

$$scale_y = |y2 - y1| \tag{5.5}$$

5.2.3 Automatic Evaluation

The system is developed to provide feedback to the user as to whether he or she has correctly identified one of the predefined anatomical features. This uses the method of collision detecting which is common in computer graphics. Collision detection is frequently used for games and physical simulations to detect when moving objects collide, so they do not overlap. An example where collision detection would be used is determining if an object being thrown collides with a wall.

Collision detection is applied in this case by testing whether the selection collides with the two bounding boxes. The two boxes are specified as solid rectangular prisms, so that anything inside of them will register as collisions. The selection object, on the

other hand, is specified as a open-ended shell, so that it will not collide with other objects inside of it. As shown in Figure 5.2, the selection should lie within the outer bounding box, but outside of the inner bounding box. This equates to a collision with the outer box and no collision with the inner box.

Collision detection was been implemented using a library called *SOLID*, which stands for "Software Library for Interference Detection" [11]. This library uses specially defined graphical objects that contain additional information for the system to be aware of the space that they occupy. Various objects, such as cubes, spheres, cones, and cylinders are already defined and available for use.

Initially, a simplified scenario was used, where only a single bounding box was generated to define the region of interest. The goal of this initial experiment was to automatically detect if a rectangular prism drawn based on the user selection intersects with the defined bounding box.

The first step toward this goal was to implement the framework for evaluating a selection. This consists of a GUI button to call the evaluation function and a textbox within the GUI to output the results.

SOLID was set up to check for collision of two fixed, overlapping graphical objects. A basic *SOLID* configuration was built based on examples included with the library. It was also tested that intersections would not be detected when the graphical objects were moved to no longer overlap. This basic test was successful, paving the way to generate the selection object based on the user selection.

The user-specified coordinates were used to dynamically define the selection box, and again, intersection was tested. This test was also successful, showing that dynamically defined objects could be tested for collision.

Due to time constraints, automatic detection could not be entirely completed. Some additional effort is required to create a rectangular shell *SOLID* object for the user selection, so that when properly selected, it will only intersect with the outer bounding box. This shell should be comprised of four rectangles to form a

rectangular open-ended shell. A correct selection would be detected when the user selection intersects with the outer bounding box, but not the inner box.

5.3 Scan Path Recording and Display

Another valuable skill for a sonographer is the ability to scan efficiently, so that a diagnosis is reached in a short time, yet without missing important features. This is especially relevant in emergency ultrasound, where it is critical that the internal trauma be identified quickly. By recording the scan path of the transducer, the opportunity exists for the trainee and instructor to later view the scan path and discuss the trainee's scanning methods.

Figure 5.4 shows a block diagram of the scan path recording and display system. The scan path is recorded within the training system environment, then the MATLAB engine is called to process and display the data within the MATLAB environment. A stored surface model of the manikin is displayed by the overlay script and a GUI is drawn on the graph for control of the playback.

5.3.1 Data Recording

Whenever the 'Record Data' checkbox is enabled in the GUI, every data point from the tracking system is recorded to a log file as the user scans the manikin. This log file is formatted as a *Comma Separated Value* (CSV) table. A typical record is in the form 'x, y, z, a, e, r, timestamp', where the records are double-precision floating point values. The parameters x, y, and z represent the three translational axes while a, e, and r are the three rotational axes and stand for azimuth, elevation, and roll, respectively. CSV was chosen because it is a simple format to read and write, and is compatible with Excel and MATLAB. The log file is automatically saved as a file named 'log_data.csv'.

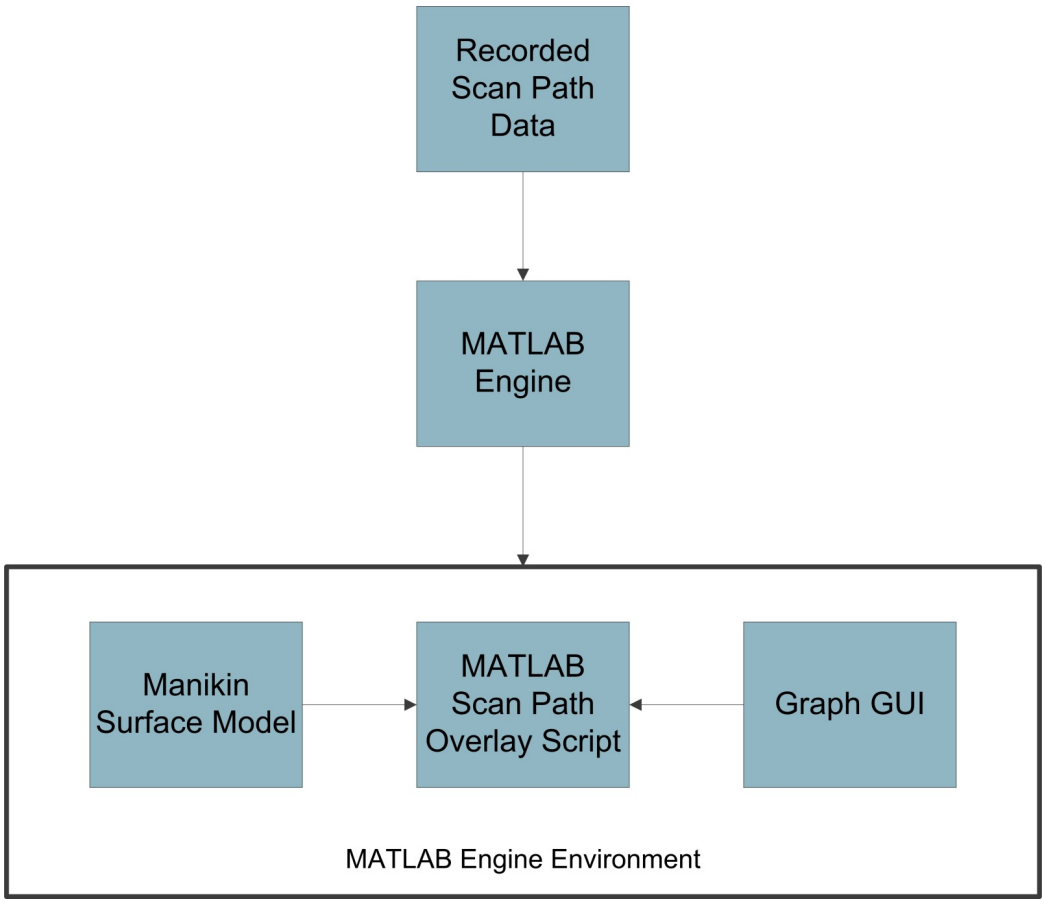


Figure 5.4: Scan Path Recording and Display Block Diagram

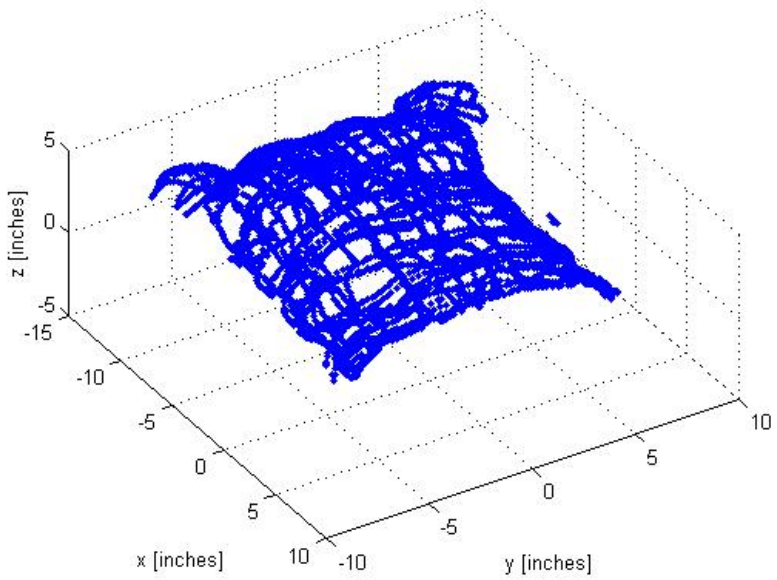
5.3.2 Manikin Surface Model Generation

The surface generation script generates a 3D surface mapping of the manikin based on a captured data set. The data consist of a series of x,y,z coordinates captured by scanning the surface of the manikin with the position tracking system. Data were captured using the *ts_capture* software. Points were captured in a grid by making tight back-and-forth motions, spaced approximately 1 cm apart. A secondary, similar grid oriented perpendicular to the first one provided additional detail. As data points are captured at approximately 50 points per second, scanning can be done at a comfortable pace. Figure 5.5(a) shows a set of points used to generate a surface plot.

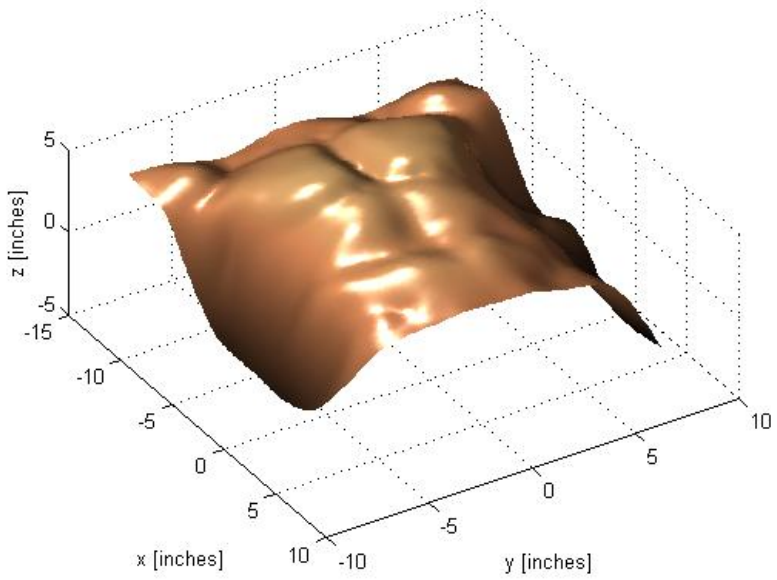
To properly create a surface rendering, interpolation was performed between the captured data points. To do this, a user-contributed library from MATLAB's File Exchange called *gridfit* was used. This creates a set of interpolated points fit to a square grid. The new interpolated data were plotted in MATLAB as a surface plot. Copper shading tones were used and appropriate lighting was applied to create a realistic-looking surface model. Figure 5.5(b) shows an example of a surface rendering generated by this script. It can be seen that the surface rendering is a visually accurate representation of the surface of the manikin, showing even the most minute details. The MATLAB code for the surface generation script can be seen in Appendix H.1.

5.3.3 MATLAB Engine

MATLAB provides the MATLAB Engine for interfacing with other applications. This engine can be opened from within C or C++ code by including the appropriate headers and libraries and issuing a call to the engine. The engine is opened as a MATLAB command terminal, rather than opening a full graphical instance of MATLAB. Once the engine is open, MATLAB commands can be run simply by issuing calls to the engine with the command text. This can be used to run specific commands or



(a) Captured Points for Manikin Surface Plot



(b) Manikin Surface Plot

Figure 5.5: Manikin Surface Model

call a script to run a pre-defined routine. The one drawback to using the MATLAB engine, is that MATLAB is required to be installed on the computer running the software. This is a significant limitation, but for the purposes of this software, it can be expected that it will be run in an academic environment where MATLAB would be available or on a system-specific laptop, which would include a MATLAB installation.

The scan path overlay script is called as a MATLAB function. Once the function call has been issued, there is no need for further communication between the training software and the MATLAB engine, as a GUI is provided for the MATLAB script so that it may be controlled without additional communication between the software and the MATLAB engine.

5.3.4 Scan Path Overlay

The scan path overlay script is called by the training software and generates a display of the recorded scan path drawn onto a surface model created using the script in Appendix H.1. The points are drawn one at a time at a time step of 0.02 seconds; corresponding to the training system capture rate. This permits accurate visualization of the scanning path and motion.

The plotted points represent only the position of the recorded points and do not reflect the orientation of the sham transducer during the capture. In most cases, displaying only position is adequate to visualize and evaluate the scan path. However, if rotations were necessary, the data points could be shown by drawing a series of rectangles with appropriate rotations applied rather than dots.

Figure 5.6 shows an example of a scan path overlay on top of a surface rendering, which was generated by this script.

To improve usability, a GUI was added to the MATLAB scan path overlay script. This was designed using MATLAB's Graphical User Interface Development Environment (GUIDE). GUIDE provides a simple graphical interface for designing GUIs,

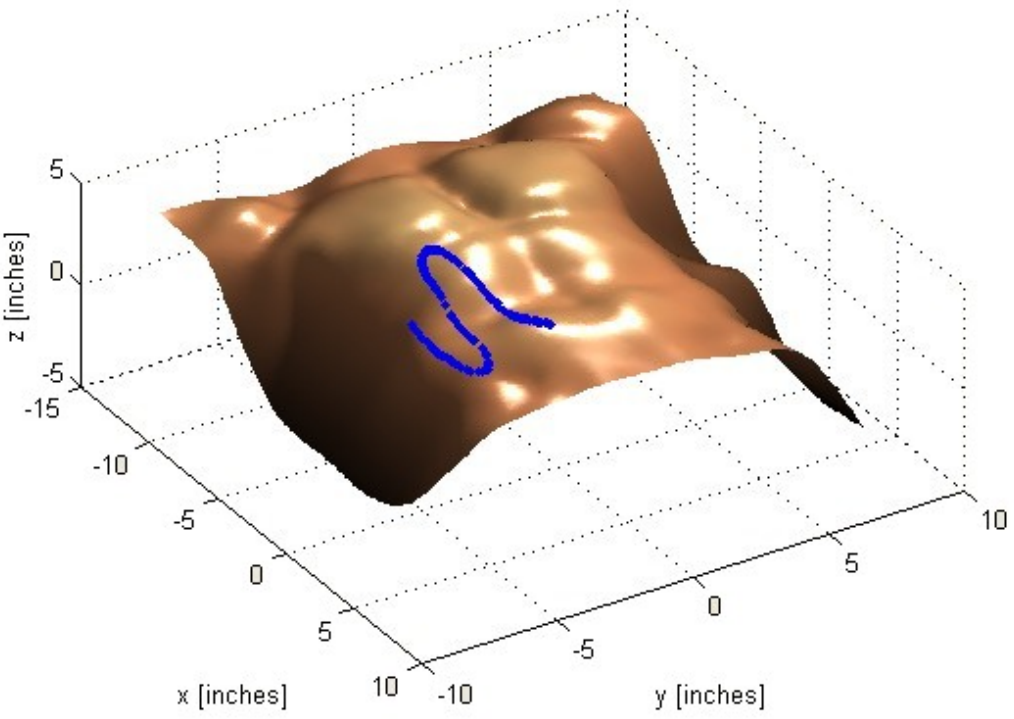


Figure 5.6: Manikin Surface Plot with Scan Path Overlay

which permits the user to add various objects to a form. These objects are then linked to callback functions, which can be edited by the user.

Four buttons are implemented along with the graphical plot. 'Plot Path' plots the entire scan path instantly. 'Play Path' plays through the scan path point-by-point, showing a real-time replay of the scan. 'Fast Play' plays back the path in the same manner, but at a rate twice as fast. This acts like the fast-forward button on a videocassette player. Finally, 'Clear' erases the plot so that the user may start over.

Figure 5.7 shows a screen capture of the scan path overlay with the GUI. MATLAB code for the scan path overlay script can be seen in appendix H.2.

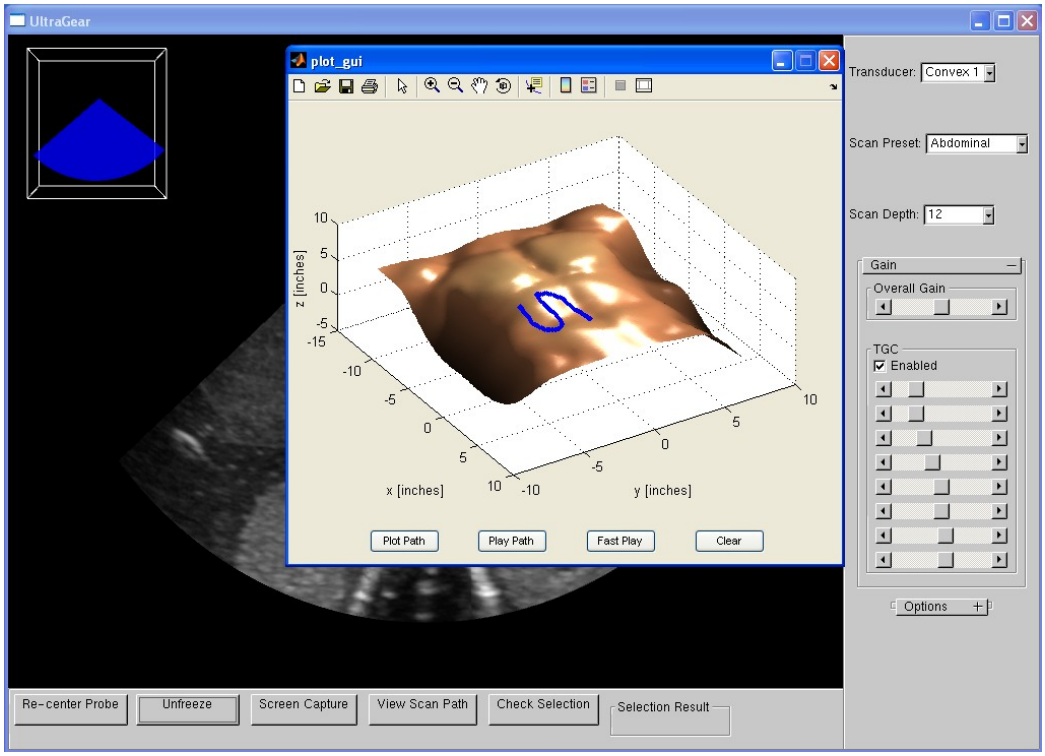


Figure 5.7: Scan Path Overlay with GUI

Figure 5.8 shows the software process that occurs when the 'View Scan Path' button is clicked. First, the MATLAB engine is started. Next, the display is frozen, recording is stopped, and the recording file is closed so that it can be opened with

MATLAB. The MATLAB script is then run, which draws the manikin surface model and MATLAB plot GUI. Meanwhile, the program stays in a frozen state until the 'Unfreeze' button is clicked, which will create a new data file and resume normal program operation.

5.3.5 Uses for Assessment of Learning Outcomes

Scan path recording and display is most useful when combined with analysis from a human trainer. The trainer can view the playback of the recorded scan path and comment on what the trainee was doing properly or incorrectly. The primary benefit is that the trainer does not have to be present at the time of scanning. It is also possible to play back the scans at an increased speed to save time when viewing long scans. A module within a training course for sonographers could involve the trainees recording their scan path for a given scan type. The trainees could be asked to submit all of their scan path files for the trainer to review. In turn, the trainer could demonstrate what the scan path of a professional might look like and provide comments.

In addition to being used with the training software, the scan path overlay script can be used as a standalone scan path viewing utility. This can be especially useful for permitting trainers to view the scan paths captured by trainees. To use the script in this manner, the script can be opened in MATLAB and the filename of the desired scan path can be entered into the script.

In the future, it may be possible to design a system for automatically detecting whether a scan path exhibits desired characteristics and covers the correct areas of the manikin. These features have not yet been implemented, but are a conceivable extension of the current scan path recording system.

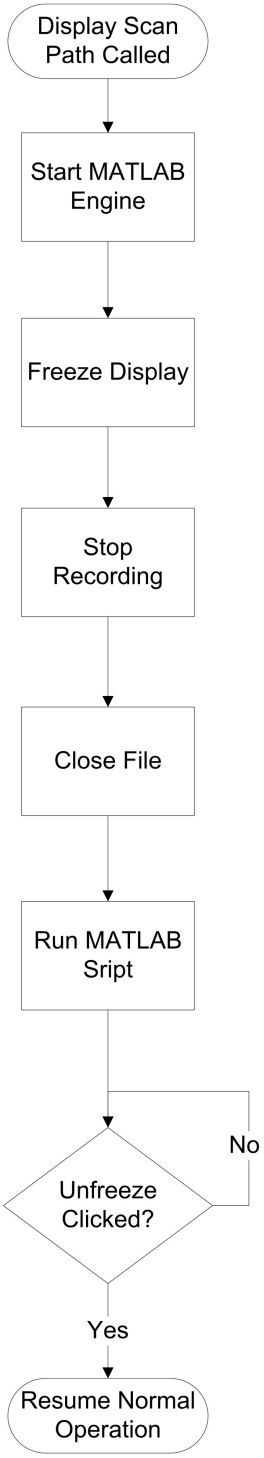


Figure 5.8: Scan Path Playback Software Flowchart

5.4 Still Image Capture

Capturing still images is an important part of a typical ultrasound exam. While it is important that the sonographer analyzes the patient during the scan, in most cases the sonographer's main role is to capture images that will later be reviewed by a doctor. Image captures are essential for record keeping, second opinions and further analysis. Often, following an ultrasound examination, the recorded images will be looked over by the patient's doctor. Additionally, like x-ray scans, the ultrasound images become part of the patient's record, providing documentation of what was found.

As a very commonly used feature of ultrasound, still image capture is essential to implement in the ultrasound training system. Not only does this provide an expected feature; it also provides some additional training and assessment opportunities. By requiring trainees to capture images of certain anatomical features and pathologies, their ability to scan for features and capture quality images can be evaluated. By capturing scan planes, trainees can keep records of their work. These saved scan images can later be analyzed with a human trainer to discuss the quality and content of the captured images.

The captured scan images are in the standard JPEG format, which is very commonly used for digital camera images. Because of this, they can be opened in nearly any image viewing application.

5.4.1 Implementation of Still Image Capture

Screen capture capabilities were added to the training system software through the use of an open source library called *mkOpenGLJPEGImage*. This class simplifies the process of capturing a JPEG image from the current OpenGL display, condensing the process to only a couple of simple lines of code.

To permit capturing of multiple images within a single session, it is necessary

that the captured images be appended with a unique serial number. To do this, a variable is stored that contains the number of the current image. Each time the capture function is called, the current number is appended to the image in the form 'captured_image#.jpg', where # is the image serial number. After the image is stored, the serial number variable is incremented in preparation for the next capture.

There are two different ways to trigger the screen capture. A button has been placed on the GUI toolbar as described in Section 3.3.3. This button can be seen in Figure 3.12. Alternatively, the user can press the 's' key on the keyboard to trigger the image capture function.

5.5 Conclusions

The learning outcomes assessment tools described in this chapter present new learning and assessment opportunities that are not always possible with traditional ultrasound systems. Traditionally, training has been performed with a personal instructor and by evaluating only the captured scan images outside of the training session. These improvements allow the instructor much more flexibility in the training process. The instructor can give an assignment to an entire class and then evaluate their progress at a time most convenient for them. It could even be possible for a trainer to teach remotely via the internet. By saving screen captures and scan paths, the scanning and image capture skills of the sonographer can be evaluated. Region of interest selection provides the opportunity to automatically evaluate whether students can correctly locate various pathologies and anatomical features.

Overall, these features significantly improve the training process. Although the system cannot provide a complete evaluation without any human intervention, it certainly simplifies the process. Trainees are given more freedom in when they can perform the training exercises, because they do not need an instructor present during the training session. Similarly, trainers do not need to be present for the training

session and can afterward compare the results from the entire class. Through these learning assessment tools, the training process is simplified and streamlined.

Chapter 6

Results and Discussion

The following sections present some results and discussion for each of the components of the system that were described in the previous chapters. Additionally, the results of a clinical evaluation are presented in this chapter.

6.1 Ultrasound Training System

Figure 6.1 shows a screen capture of this training environment. The training system interactively generates scan planes in real time, based on a user's movement of the sham transducer. The update rate is approximately 20 frames per second on a standard desktop computer, which provides a smooth, realistic display of scan images.

The training environment includes a number of features to make it intuitive and realistic. The user can select between different probe geometries and change the scan depth while scanning. The gain and TGC can be adjusted to improve the display of certain features. The display can also be frozen for further analysis of a given scan image. An effective GUI has been implemented to control all of the features and the program can be displayed in either windowed or fullscreen modes.

The tracking transmitter has been embedded into a manikin by mounting it on a board with rods that slide into pipes embedded in the manikin. This hides the

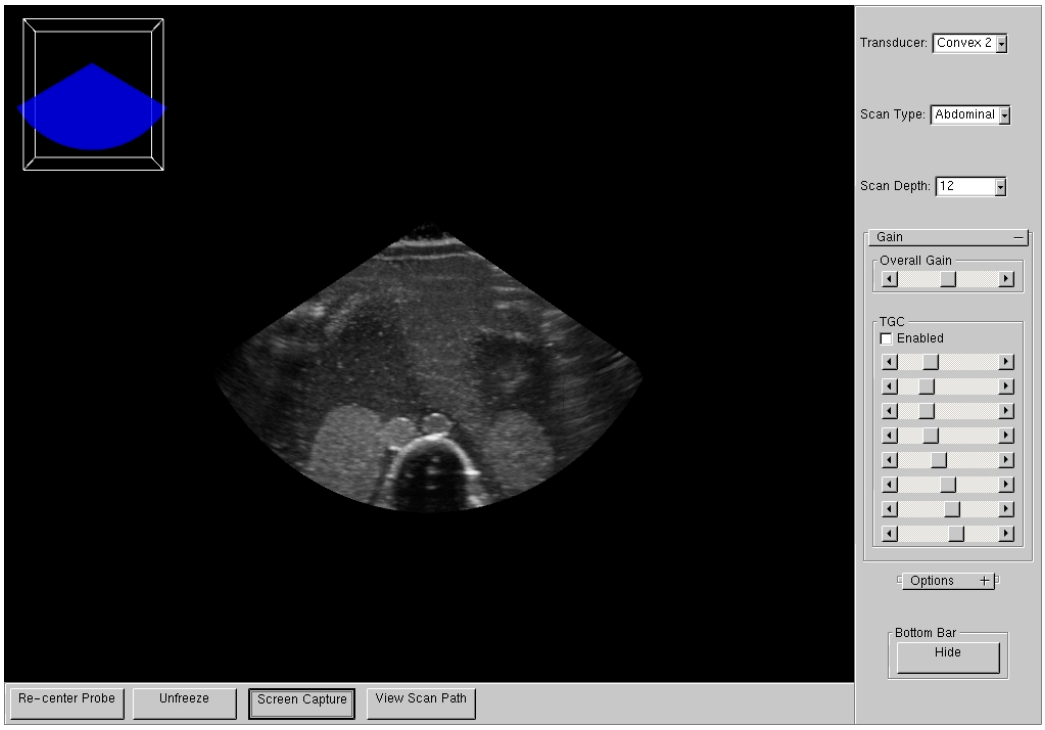


Figure 6.1: Screen Capture of Ultrasound Training System Software Environment

tracking system and ensures that the sensor can be removed and put back in without changing the location relative to the manikin. The sham transducers are very convincing and could easily be mistaken for real transducers. Both linear and convex array sham transducers have been constructed and can be switched by simply selecting from a menu in the software. A touch screen has been added for simplified use and intuitive access to learning assessment features.

Overall the software and hardware interfaces make for a very realistic user experience and provide an authentic training environment. The software has been designed to emulate that of computer-based portable ultrasound systems and provides a similar interface and most of the scanning features of these systems. The hardware interface is very realistic and could easily give a user the impression that they are scanning a specially-designed anthropomorphic phantom rather than a generic training manikin.

6.2 Data Acquisition

A means of acquiring data for use with simulations has been developed and evaluated. The use of *Stradwin* to capture multi-sweep ultrasound data makes the capture process fairly simple and effective. Multiple sweeps are taken with *Stradwin*'s 'Multiple Gated' capture mode. These data are recorded with calibrated position information to spatially align the image data. Using *Stradwin*, the multiple sweeps can be aligned to form a single, contiguous volume, which can be exported as evenly-space voxel data.

Multi-sweep data were captured by scanning an ATS anthropomorphic phantom and stitched with a fair degree of success. Figure 6.2 shows a comparison between 3 individual sweeps and a composite volume created by stitching together the three overlapping sweeps. This figure represents one of the better parts of the volume and there are some portions that do not line up quite as perfectly. Overall, the stitched volume looks fairly good, having a well-aligned middle section that shows almost no

discernible lines between sweeps. However, the outer portions of the volume are not quite perfect and show more distinct lines at the sweep boundaries. This phantom only accounts for a small slice of the abdominal region and therefore can only be used for proof of concept, rather than generating simulation data; therefore, capturing data from human subjects is essential.

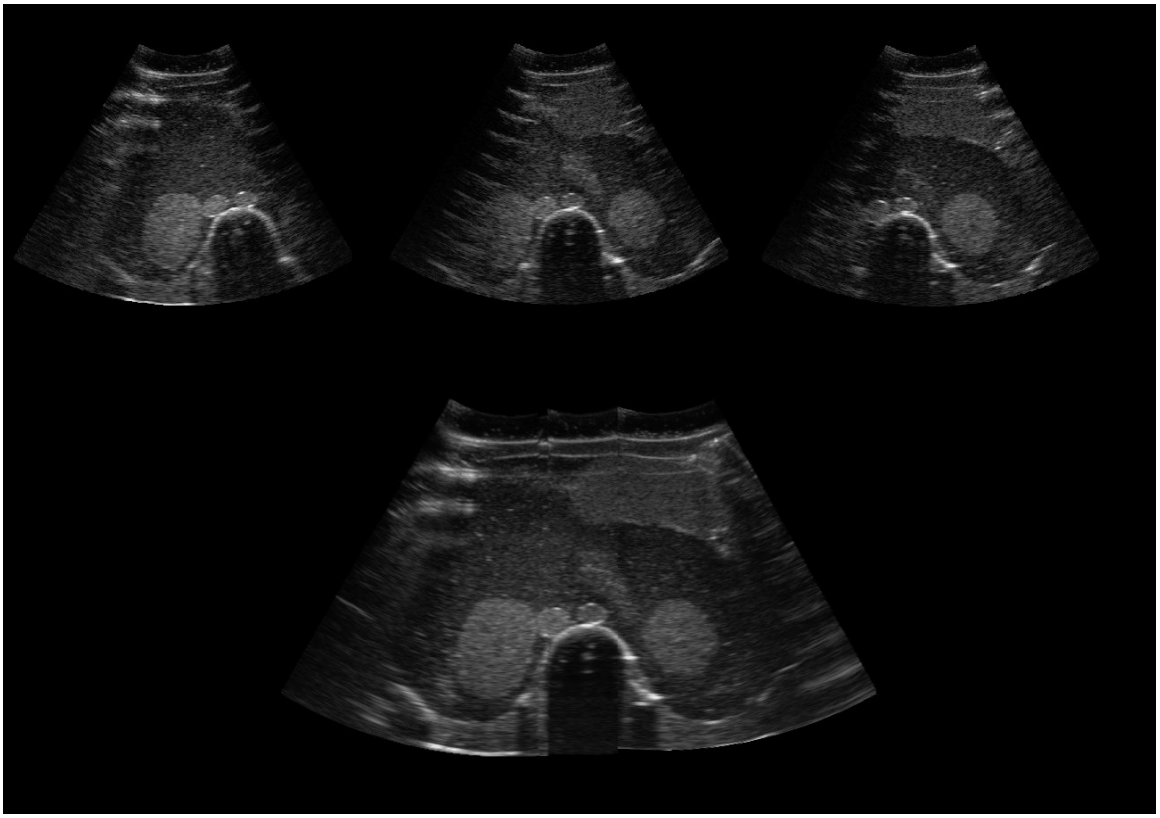


Figure 6.2: Comparison of Individual Scans and Stitched Volume

While capturing data from a phantom proved to be moderately successful, obtaining quality data from a human subject is a much more difficult process. From discussion with clinical contacts, we learned that the primary source of difficulties in stitching 3D image volumes from actual human subjects is motion of the body and organs due to internal movements and external forces. Internal movements are related to motion within the body during scanning, such as those caused by breathing, heart

motion and intestinal gas. This causes two scans of the same area to contain different data and adjacent sweeps that, based on position data should overlap, do not line up perfectly. Intestinal gas proves to be especially troublesome, as the gas bubbles cause significant image degradation. External forces consist primarily of probe pressure. When different pressures are applied, internal organs are compressed to different degrees. While *Stradwin* attempts to adjust for differences in probe pressure, it cannot always perform this process perfectly. Sweeps in different directions will also push organs in slightly different ways, further altering the captured data, so it is important that sweeps be taken in the same direction.

The result of these scanning difficulties would be a data set that does not line up properly when stitched together using *Stradwin*'s multi-sweep stitching feature. This makes it apparent that refinement of the stitching process is required. *Stradwin*'s method of stitching together multiple sweeps based solely on position data cannot be entirely effective for data where motion and deformation have occurred. What is really necessary is some type of non-rigid registration, which would deform the captured data to properly create a stitched volume. Dr. Joyoni Dey's code that was described in Section 4.3.6 may be an effective solution to this issue. By performing both affine transformation and free-form deformation to the multiple sweeps, it may be possible to produce accurately stitched image volumes.

Although the current stitching process does not appear to be effective for human subjects, *Stradwin* does seem to be the best choice for data capture, even if it is not used for stitching. *Stradwin* provides a clean and effective interface for scanning 3D data volumes with position data. The software is well supported by the team at the University of Cambridge and software updates are frequently released with new features. The calibration mode, although somewhat difficult, is generally effective for aligning ultrasound image data to the tracking system coordinates. Overall, *Stradwin* is a quality piece of software and greatly simplifies the process of capturing multi-sweep data with accurate position and orientation information.

6.3 Learning Outcomes Assessment

Software tools for learning outcomes assessment have been implemented to simplify the training process and provide improved opportunities for evaluating skills. These tools focus on providing stored data that can be analyzed off-line, as well as providing some degree of automatic assessment. Region of interest selection permits the user to select a specific pathology or anatomical feature and be graded on whether they were able to correctly identify the requested feature. Although automatic evaluation of these selection has not been completely implemented, the framework exists and proof of concept has been verified through partial implementation. The scan path recording and playback features permit the scan path to be recorded and later played back, showing the recorded path superimposed on a surface rendering of the manikin. This enables evaluation of a trainee's scan path and whether they were scanning efficiently and effectively. The ability to capture still images is implemented to provide a permanent record of notable scan images and to evaluate a trainee's ability to capture meaningful images. These features have been successfully designed and are implemented directly within the training system software where they can be accessed using the graphical user interface.

These learning assessment tools, when combined with professional analysis by an instructor have to potential to be very effective for streamlining the training process. By providing standardized sets of captured images and scan paths, the process of evaluating a trainee's progress is greatly simplified. Furthermore, region of interest selection provides some degree of automation to the process by providing instant feedback as to whether features were correctly selected.

Due to time constraints, these tools have not yet undergone a clinical evaluation. The clinical evaluation will be an essential measure of whether these tools are effective for their intended purpose and how they might be improved. Tests should be performed to determine whether a user's skill can be determined based on captured

images and scan paths and whether the region of interest selection tool is effective. Ideally, this evaluation should happen as part of a traditional course for sonographers, which could provide opportunities for comparison, along with opportunities to discuss the training merits with students taking a course.

6.4 Clinical Evaluation

An initial clinical evaluation of system performance and images was performed by two sonographers at the University of Massachusetts Memorial Medical Center in October, 2008. They were given the opportunity to test the system and asked to provide feedback. They found the resolution of the tracking system to be realistic and felt that the frame rate was adequate. They liked the idea of an ultrasound training system and pointed out the value of an ultrasound training system for studying a variety of conditions in a hands-on environment. At the time of evaluation, none of the learning outcomes assessment tools had yet been implemented, so no evaluation of those features was possible.

One observed shortcoming was the low quality of the current image data. This observation is quite understandable, as the available image data set was derived from a single sweep of a prostate scaled up to cover a larger area. As data capture methods are improved and better data are captured, this problem should be resolved. Overall their reaction was positive and enthusiastic.

The opportunity to discuss the benefits and shortcomings of the system with experienced sonographers was a valuable experience and provided some useful insight. This helped to refine some of the project goals and provided reassurance that there is a market for a system like this.

In the future, further clinical evaluation will be necessary to determine the effectiveness of the system and quality of new data sets. Specifically, the system should be tested on sonographers, ultrasound trainers and sonographers in training to de-

termine the strengths and shortcomings of the system. This will be very helpful for determining future development goals.

Chapter 7

Conclusions

In this thesis, a successful prototype of an interactive training system for medical ultrasound has been developed. The simulation system permits a trainee to scan a manikin with a sham transducer and view scan planes on the computer screen, updated in real time. With the addition of the hardware interface, the software can be run on a standard PC-based desktop or laptop computer. This system provides a realistic scanning experience, complete with adjustable image parameters such as scan depth, gain, and TGC. Tools for learning outcomes assessment are provided, including region of interest selection, still image capture, and scan path recording and playback.

This system provides an environment for learning and experimentation without the need to study human patients or use a costly ultrasound system. Furthermore, once data for a given trauma or pathology has been captured, they can be used repeatedly for training. This provides the opportunity for sonographers to get hands-on experience studying conditions that would otherwise be too rare for training opportunities to arise.

Data capture has been the most problematic part of this system and further work is still needed in this area. Although, as shown in Section 6.2, scans of phantoms can be captured with a fair degree of success, capturing data from human subjects

proves to be much more difficult. After consulting with clinicians about the merits of combining multi-sweep data captured from human subject, it has become apparent that there may be a need to look back at comparison-based registration methods.

This completed prototype system is a major step toward a marketable ultrasound training system. If an effective library of data can be compiled, this system has the potential to revolutionize the way that sonographers are trained. By streamlining the training process, a computer-based training system could make ultrasound much more accessible and encourage its use to become more widespread.

7.1 Future Work

Although an effective demonstration system has been developed, there are still some improvements that could make the system more effective for training and more commercially viable. Many of these ideas were not goals of the initial project, but became obvious opportunities for improvement during the development process. Due to the limited scope and time frame of this project, a number of features could not be implemented. This section describes these ideas in the hope that they will be considered for future development opportunities.

The most significant future work that is required is in the area of simulation data. As noted by the clinicians who evaluated the system, the current simulation data are of fairly low quality and could certainly use improvement. Because data capture has proven difficult, image quality has been lower than desired and has not been conducive to the simple *Stradwin* stitching process. The inability to effectively stitch together multiple volumes captured from human subjects has prevented complete data sets from being produced. The most beneficial development would be to find a way to successfully perform a non-rigid alignment of multiple ultrasound sweeps. While Dr. Joyoni Dey's code shows potential, further effort is necessary to determine if it will produce acceptable results with ultrasound data. If this method proves ineffective,

design of custom software may be necessary. A custom multi-volume registration program with these goals in mind has the potential to be much more effective than attempting to make something else work for the needs of this project. This would, of course, require significant effort and should only be undertaken after ensuring that there are no existing solutions that could potentially work.

Another major area for improvement would be in the area of real-time image processing. By performing additional processing on the image data and scan planes during the training session, more realistic images can be generated. With further improvements, it may be possible to make the training system nearly indiscernible from a real ultrasound system.

The following are some image processing improvements that could enhance the system and are described in the proceeding paragraphs:

- Dynamic Shadowing
- Transducer Geometry Conversion for Image Data
- Focus Adjustment Emulation
- Probe Pressure Processing

Dynamically generated shadows would be beneficial for creating more realistic scan images. Fixed shadow direction and placement are unrealistic and make the training system distinguishable from using a real ultrasound scanner. Shadows generated during the simulation could change based on how the sham transducer is held, for a more realistic appearance.

The ability to properly convert data captured by one transducer geometry to look like another geometry would also be quite useful. Without any additional image processing, it is always obvious what type of transducer was used to capture a data set, regardless of the mask being applied. To make this a reality, a method for properly changing the speckle patterns and shadows would be necessary. This could

alternatively be performed as an offline process and different data sets could be loaded when changing probe geometries.

Adjustment of focus is an important part of ultrasound imaging that is currently not emulated by this system. The data inherently show the focus with which they was captured. It would be beneficial to perform some type of data processing to give the appearance of adjustable focus. While it would be difficult to improve upon the focus of the original image, it may be possible to at least make areas appear out of focus through Gaussian blurring. If the simulation data are captured with good, multiple-depth focusing that maintains a well-focused image throughout, then this blurring could be used to generate the appearance of adjustable focus. It is likely that this would be implemented in a similar manner to TGC, generating an interpolated gradient between focus points. However, instead of applying brightness, blurring would be incorporated.

Ideally, probe pressure would also be accounted for. The image should accurately compress when enough pressure is applied to the manikin to compress it. The image should compress properly to show the shapes of internal organs being temporarily deformed. This pressure being applied could be measured either using a pressure or force sensor, or by analyzing the probe displacement relative to the manikin surface. Additionally, when the probe is not in contact with the manikin's surface, no image should be displayed. Similarly, there should be no image if ultrasound gel is not used. Skin contact could be measured using a pressure sensor, or a surface map of the manikin could be used to determine in software if the probe is in contact with the manikin. The presence or absence of ultrasound gel could be detected using a moisture sensor.

The learning assessment tools could be improved by introducing more automated assessment. For example, the capability to determine the quality of a given scan path compared with a scan performed by a professional would streamline the scan path evaluation process and limit the need for a human instructor. Image processing tech-

niques to determine the quality and content of captured images would also enhance the learning process and help to create a more automated training process.

A potentially useful extension of this project would be to implement 4D simulation by adding a time dimension. A major difficulty with a 4D approach is the large amount of data that would be necessary. There would essentially need to be an entire image volume for each time step within the sequence. While this might be realistic for very limited volumes, a 4D image volume of the entire abdominal region would be likely to exceed the memory capabilities of most current computer systems. However, as time progresses and computer systems improve, this may eventually become more realistic.

Appendix A

Software Design Considerations

In any software project, there are some initial design choices to take under consideration. These include development environment, linking method, and source code control.

The development environment refers to the operating system, integrated development environment (IDE), and compiler that are used. As most work for this project was performed in the Window XP operating system, this was a clear choice for development. Although Windows development can be slightly more tricky than development on Linux or Unix systems, most of the difficulties come with initial acclimation to the specific errors that can occur. The selected IDE and compiler are Microsoft Visual C++ .NET 2003. Microsoft Visual C++ is the standard for Windows C++ development and is well-supported. Although 2003 is not the most recent version, it provides all of the necessary functionality required for this project.

An important development choice is the selection between static and dynamic linking. Static linking packs all of the libraries into a single executable, which creates a larger file size, but does not require any external libraries. This avoids difficulties with missing library files. Dynamic linking uses Dynamic Link Library (DLL) files, which are external to the program and must be available on the system running the software. The primary benefit of dynamic linking is that if multiple programs are

being run using the same libraries, only one copy of the library must be loaded into memory, which can be beneficial if multiple programs are sharing the libraries. Static linking is a clear choice for this system because for a relatively small program like this that is using mostly unique libraries, there would be little benefit to dynamic linking. The benefits of being able to create a single executable and avoid missing DLL issues far outweighs the slight reduction of memory requirements gained by dynamic linking.

Source code control is an important part of any software project, as it allows revisions to be tracked and facilitates keeping frequent backups and records of progress. Subversion (SVN) and CVS are the two most common types of code repositories. While either would be an acceptable option, SVN was chosen because of prior experience with it, which reduces the required time to get started working with it. All source code, along with the documentation and multimedia files for this thesis were kept under subversion control using WPI's SourceForge server.

Appendix B

Ascension Technologies Corporation *Flock of Birds* Datasheet

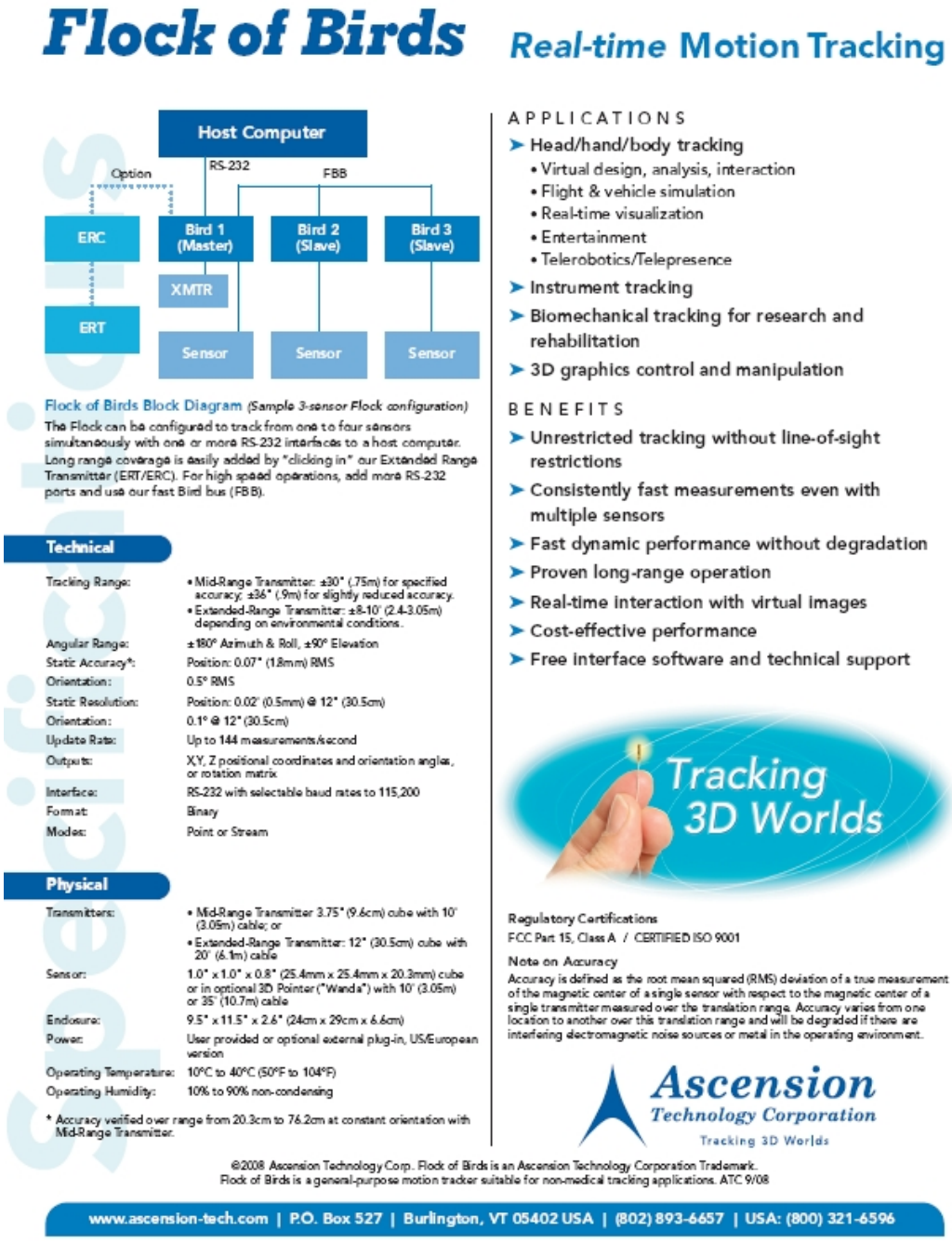


Figure B.1: Ascension Technologies Corporation *Flock of Birds* Datasheet [4]

Appendix C

Ascension Technologies Corporation *trakSTAR* Datasheet

3D Guidance trakSTAR

Specifications

Technical

Sensor Configurations	Model 800 (8.0 mm), Model 180 (2.0 mm) Model 130 (1.5 mm)
Degrees of Freedom	6 (Position and Orientation)
Update Rate	Up to 420 updates/second for each sensor (Default: 240 updates/second)
Translation Range	**MODEL 800 SENSOR** • Mid-Range Transmitter: 78 cm (31 inches) • Short-Range Transmitter: 46 cm (18 inches) **MODEL 180 SENSOR** • Mid-Range Transmitter: 58 cm (23 inches) • Short-Range Transmitter: Contact Ascension for latest test results. **MODEL 130 SENSOR** • Mid-Range Transmitter: 46 cm (18 inches) • Short-Range Transmitter: Contact Ascension for latest test results.
Angular Range	All Attitude: ± 180° Azimuth & Roll, ± 90° Elevation
Static Accuracy*	Position: 1.4 mm (0.055 inch) RMS Orientation: 0.5° RMS *Higher accuracies achievable in smaller tracking volumes. *Accuracies vary depending on specific transmitter-sensor configurations.
Static Resolution	Position: 0.5 mm (0.02 inch) at 30.5 cm (12 inches) Orientation: 0.1° at 30.5 cm
Outputs	X, Y, Z positional coordinates, orientation angles, orientation matrix or quaternions
Interface	USB 1.1 / 2.0 or RS-232
Data Format	Binary data records
Communication	Windows API and Drivers

Physical

Electronics Unit	18.5cm (7.3 inch) x 29.2 cm (11.5 inch) x 6.4cm (2.5 inch)
Transmitters	• Mid-Range: 9.6 cm (3.8 inch) cube with 2.3 m (10.8 ft) cable • Short-Range: 6.27 cm (2.5 inch) x 4.6 cm (1.8 inch) x 5.2 cm (2.1 inch) with 2.3 m (10.8 ft) cable
Passive Sensors	**MODEL 800:** 8 mm (.31 inch) x 20 mm (0.78 inch) with 2.3 m (10.8 ft) cable **MODEL 180:** 2 mm (0.07 inch) x 9.7 mm (0.38 inch) with 2 m (6.6 ft) cable **MODEL 130:** 1.5 mm (0.05 inch) x 7.7 mm (0.30 inch) with 2 m (6.6 ft) cable Model 180 & 130 only: • Ascension MediMag Cable, USP class 6 jacket material. • USP class 6 sensor housing. • Sensor assembly and cable materials are ISO and cold sterilant tolerant. Warning: Semiconductor devices in sensor connector are not gamma shielded and may be damaged or erased if exposed to gamma radiation and/or autoclaving. • Sensors and cable assemblies are fragile components and must be sheathed, isolated and safeguarded prior to use in patients.
Power	100 - 240V ~ 50/60 Hz
Operating Temperature	5°C to 40°C, 95% non-condensing humidity
Environment	Ferromagnetic objects and stray magnetic fields in the operation volume may degrade performance. Contact us for assistance in using our Optimization Tools to minimize metallic distortion and noise interference.

FEATURE	BENEFITS
metal tolerant	80% less distortion due to non-magnetic conductive metals compared to AC magnetic trackers. Outputs unaffected by composite materials. Capable of driving errors induced by highly conductive metals (such as aluminum) to zero by adjusting measurement rate.
Advanced new magnetic technology and signal processing	• Better dynamic performance over longer ranges. • "Power-line" noise filtered out.
Occlusion and drift free	Clear line-of-sight between transmitter and sensor(s) is not required.
Body mountable transmitter	New lightweight coil set can be externally mounted on head or body.
Onboard diagnostics	Self-diagnostics and run-time monitoring for improved tracker reliability and safety.
Software support	XP/Pro and XP embedded compatible with SDK and sample programs. API with expert support facilitates incorporation into user applications.

Regulatory Certifications

- Class I Device with Type B Applied Part (Sensors), EN60601-1 Compliant.
- RoHS and WEEE compliant.
- Medical users must comply with all pertinent FDA/CE/IRB certifications prior to using this device in human patients.

Note on Accuracy

Accuracy is defined as the root mean square (RMS) deviation of a true measurement of the magnetic center of a single sensor with respect to the magnetic center of a single transmitter measured over the specified translation range. Accuracy varies from one location to another over this range and will be degraded if there are interfering electromagnetic noise sources or metal in the operating environment, which have not been identified and minimized.

Ascension Technology Corporation
Tracking 3D Worlds

www.ascension-tech.com | P.O. Box 527 | Burlington, VT 05402 USA | (802) 893-6657 | USA: (800) 321-6596

Figure C.1: Ascension Technologies Corporation *trakSTAR* Datasheet [4]

Appendix D

Discussion of Precision and Accuracy

Two scientific terms that are commonly confused and misunderstood are precision and accuracy. By definition, accuracy of measurement is described as the "closeness of agreement between a measured quantity value and a true quantity value of a measurand" [17]. Measurement precision is defined as the "closeness of agreement between indications of measured quantity values obtained by replicate measurements on the same or similar objects under specified conditions" [17].

In practical terms, accuracy is the closeness of a measurement to the true value, while precision is the variation of the measured values. This can be thought of in terms of the target example shown in Figure D.1. An accurate marksman will shoot close to the bull's eye, but the shots may not be close together. A precise marksman may not necessarily hit near the bull's eye, but a series of shots under the same circumstances will be very close together. One who is both accurate and precise will be able to form a tight pattern near the bull's eye.

Figure D.2 shows a graphical example of accuracy and precision. The deviation of the mean from the reference value is the accuracy and the precision is the variation about the mean, generally expressed in terms of the standard deviation.

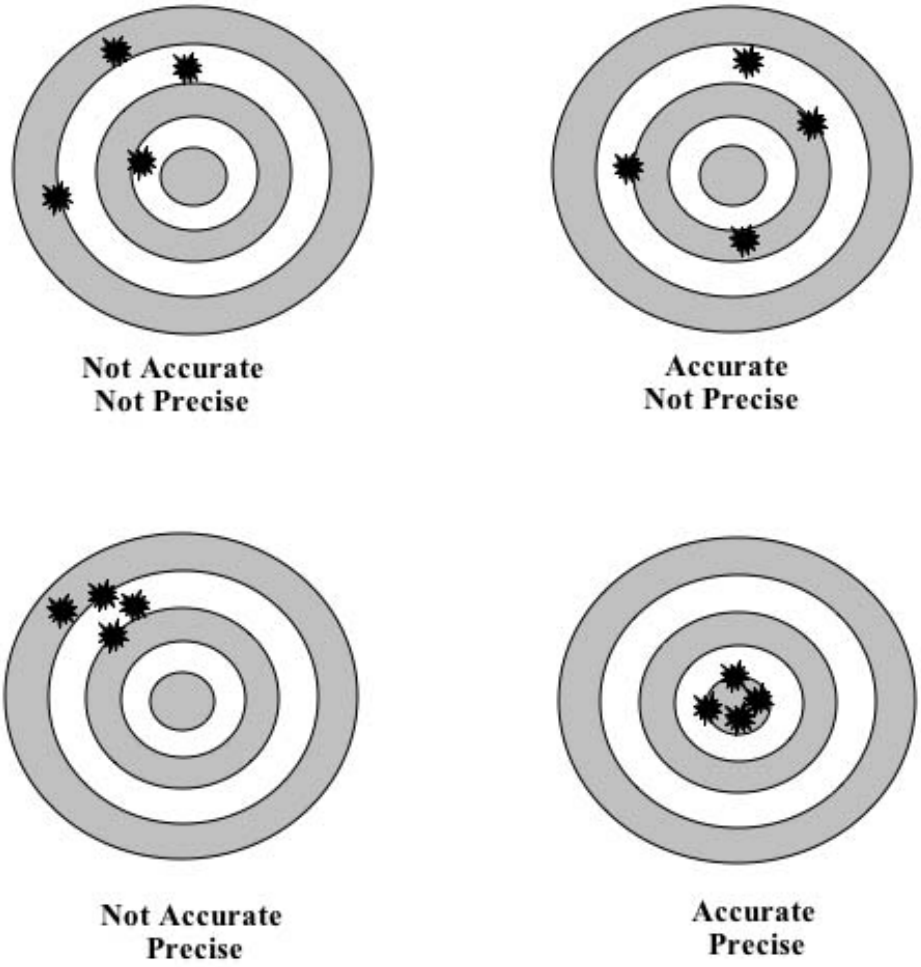


Figure D.1: Accuracy vs. Precision, Target Example [25]

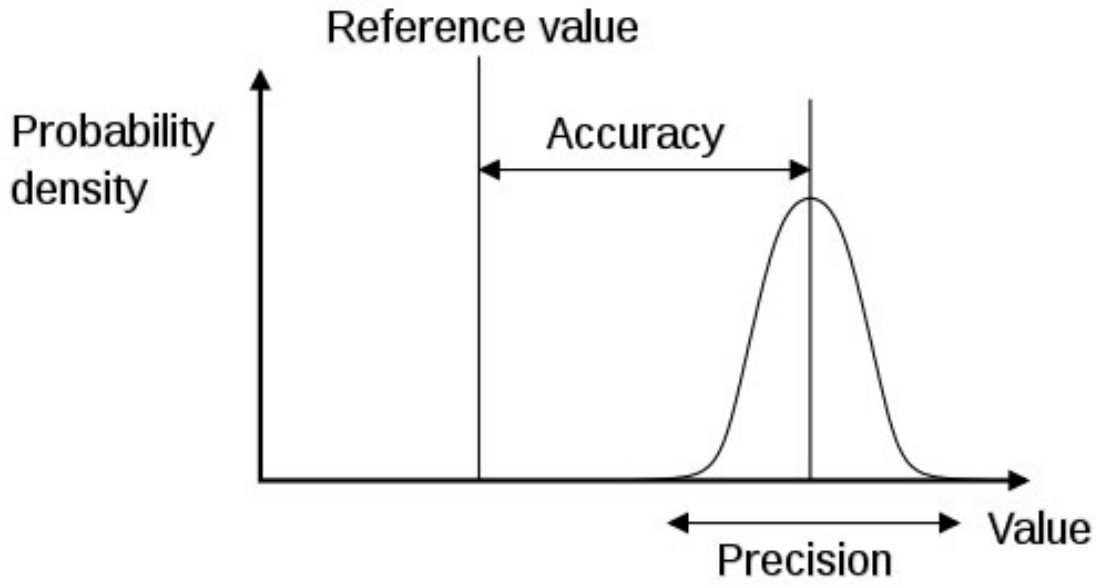


Figure D.2: Accuracy vs. Precision, Graph Example [50]

Appendix E

Comparison of AC and DC Magnetic Tracking Systems

Overcoming Performance Issues
The Advantage of DC Magnetic Tracking

Here is a chart that compares the features of AC vs. DC magnetic trackers.

Note there are two columns for the DC trackers.

1. Polhemus Inc. often cites the first two columns in comparing its AC trackers to Ascension's DC trackers. For argument's sake, we present the first DC column exactly the way it appears in the Polhemus White Paper "AC & DC Motion Trackers." Footnotes are provided to correct obvious errors in the Polhemus paper.

2. The second DC column, prepared by Ascension, reflects major improvements in DC tracking now available with our *3D Guidance* trackers.

Feature	AC Magnetic	Old DC Trackers (Legacy Products)	New DC Trackers (3D Guidance Tracker Line)
Accuracy	high–very high	medium high	high
Resolution	high–very high*	medium*	high – very high*
Speed	high–very high	low-high	ultra high[1]
Latency	low-very low	medium-high	low-very low
Initialization	no	no	no
Range	short-medium	short[2]	short-medium
LOS Limits	none	none	none
Noise/Interference	in-band signals	power mains, base band	minimal[3]
Small Sensors	yes	yes	yes
Microminiaturized Sensors	no	no	yes[4]
Tethered Sensors	yes	yes[5]	yes
Cost	low-medium	low-medium	low-medium
System Size	small	small-medium	small-sub compact[6]
Calibration	self-cal	self-cal	self-cal & run-time monitoring/ self diagnostics
Metals That Distort Measurements	good conductors[7]	earth-ferromagnetics & conductors unless update rate reduced[8]	magnetic metals not shielded by flat transmitter[9]
Multiple systems	yes	no[10]	yes[11]

Figure E.1: Ascension Technologies Corporation AC vs. DC Tracking System Comparison [7]

Appendix F

Stradwin Capture Process

This appendix details the *Stradwin* setup and calibration process. Instructions are provided to assist with configuring the hardware and software and performing a scan. These descriptions are specifically written for *Stradwin* 3.6 with a Terason t2000 [43] or t3000 [44] ultrasound system and an Ascension *Flock of Birds* [5] position sensor. While this information should be applicable to most *Stradwin* configurations, some modification to the procedures may be required for other systems.

F.1 Initial Hardware and Software Setup

A complete *Stradwin* acquisition requires the *Stradwin* software, an ultrasound image source or RF data source and a position sensor. Figure F.1 shows the specific setup used for this project. The handheld probe consists of a transducer with an attached *Flock of Birds* 6 DoF tracking sensor. A Terason t2000 or t3000 system is used for the image source. The Terason system is a computer-based ultrasound system that has an external probe and performs the processing and display on a standard PC-based computer system. The Terason hardware connects to a PC through a Firewire interface and the RF data are processed in Terason's software. Terason provides a full-featured ultrasound application, but also provides the capability to export the

raw RF or image data to other programs.

The position sensor is an Ascension *Flock of Birds*, which is a magnetic 6 DoF position tracking system. The *Flock of Birds* system was described further in Section 2.4.2. It provides accurate position data through a standard RS232 serial port.

Stradwin takes the data provided by Terason, as well as the position and orientation information generated by the *Flock of Birds* and combines them to capture 3D image data.

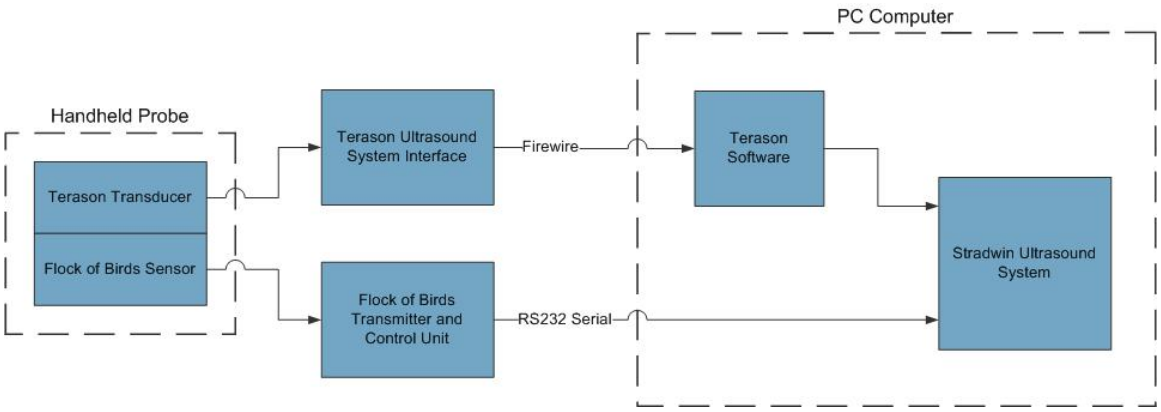


Figure F.1: Block Diagram of *Stradwin* Acquisition Hardware Interfaces

To use Terason with *Stradwin*, it is necessary to acquire some special ActiveX controls from Terason to allow the Terason image data to be used outside of their software. These files are called 'TTAutomate.ocx', 'TTFrameReceiver.ocx', and 'TTRFReceiver.ocx'. It is a good idea to put these files into the 'C:\Windows\System32' folder. They must be registered using the Windows regsvr32 program. This can be done by going to 'Run' in the start menu and typing in "regsvr32 < path + name of file >". This process must be performed for each of the three files.

Before starting to work with *Stradwin*, it should be verified that the Terason ultrasound system and the *Flock of Birds* are each functioning correctly. To test Terason, one must ensure that the Terason probe is connected to the computer through a functional Firewire port and then open the Terason software. If this is the first time

running it, it may be necessary to contact Terason for a registration code. If it is working, the display should be visible and scanning a phantom or water bath should show something on the display.

To test the *Flock of Birds*, it is necessary to verify that the system is properly connected and the main switch is in 'Fly' mode. The 'winbird.exe' program provided by Ascension can be used to test the system. In the 'SetUp' menu, 'RS232' must be selected. The COM port that the *Flock of Birds* plugged into must be selected; this is typically COM1. If the baud rate has not been changed by opening up the *Flock of Birds* and changing the configuration switches, the baud rate will be set to the default of 115,200; the baud rate should be set accordingly in *Stradwin*. In most cases, the default options will be correct unless a different COM port is used. If the main display says 'Setup Complete', the sensor can now be tested. Selecting 'Animate' under the 'Take Data' menu will display a set of axes on the screen. By moving around the sensor within a 1 m radius of the transmitter, it can be verified that the sensor responds to rotations and translations.

If both Terason and the *Flock of Birds* pass basic functionality tests, *Stradwin* can now be started. Before *Stradwin* can be opened, Ascension's winbird test software must be closed and the Terason software should be running. *Stradwin* can be accessed using the start menu or desktop icon.

The first set of options are on the 'Record' tab near the left section of the screen. It should be verified that above the 'Configure Image Source' button, 'Using Terason' is displayed. If above the 'Configure Position Source' button, 'No position sensor available' is displayed, 'Configure Position Source' can be clicked to configure the position sensor options. The same COM port and baud rate settings as were used for the winbird test software will work for *Stradwin* as well. The option to probe for 'Bird' tells *Stradwin* to search for a *Flock of Birds*. The power switch on the *Flock of Birds* should be changed to 'STBY' and then back to 'FLY' and then in *Stradwin*, 'Reprobe for Sensor' can be selected. At this point, the dialog should display 'Bird

sensor detected', meaning that the sensor is set up and ready to be used.

The display icon on the toolbar, shaped like a computer monitor, will turn on the ultrasound display. If everything is working properly, one should see the ultrasound image on the left main display and a green semicircle on the right main display. If either of these is not displayed, this can often be remedied by pressing the display button again to turn it off, resetting the *Flock of Birds*, and then turning the display back on. At this point, *Stradwin* is working and it is possible to continue with calibration or data acquisition.

F.2 *Stradwin* Data Acquisition

After initial hardware and software setup has been performed and the system has been calibrated as described in Appendix G, it is possible to begin capturing data.

If the steps in Section F.1 were followed and calibration has been performed, it should be fairly simple to begin capturing data.

Ensure that the ultrasound image display is visible and that the position is shown in the right display. Select the 'Record' tab and set the Video Rate to either 'Gated' for single-sweep capture or 'Multiple Gated' for multi-sweep capture. Gating Threshold adjusts the spacing between captured scan planes and also the sensitivity of the gating algorithm. A value of 3 is typically a good choice.

To capture a single sweep, click the red record button on the top menu, near the upper-left. Hold the transducer still against the surface being scanned until an audible start sound is heard, which sounds like a rising beep, and then begin scanning. After completing the scan, again hold the transducer still until the audible stop sound is heard, which should sound like a falling beep. If the scan fails due to the probe movement being too fast or unsteady, a 'dush' sound will be made, and the scan must be restarted. After a successful scan has been captured, click the red record button again.

To capture multiple sweeps in multiple gated mode, capture a single sweep in the same manner as in gated mode, but after making a single sweep and hearing the stop sound, move the probe over to where the next sweep will be captured and hold it still until the start sound is heard again, at which point another sweep may be captured. Multiple sweeps may be captured in this manner. When all sweeps have been completed, acquisition can be stopped by clicking the record button again.

To view the captured data, click the display button in the upper-left corner, which looks like a computer screen. The captured scan will be displayed and can be viewed in different ways using the options in the right panel. These viewing options are described in further detail in the *Stradwin* documentation [28].

Appendix G

Stradwin Calibration Procedure

The following calibration procedure is based on the procedure described in the *Stradwin* documentation, but includes additional insight derived from personal experience and conversations with the developers of *Stradwin*.

Calibration requires a water bath with a flat, textured bottom surface that produces a distinct backscatter free of excessive other reflections that may interfere with the line detector's ability to discern the actual bottom of the water bath. An acrylic bath with a standard mouse pad glued to the bottom has worked well in preliminary tests. The water bath should be filled with clean room-temperature water to a height a couple of centimeters less than the desired scan depth. If it is not possible to get room-temperature water, the temperature should be allowed to settle for a while to achieve a stable temperature. It is important that the water be at a stable temperature because the speed of sound in water varies with temperature. While *Stradwin* can account for different water temperatures, it is not designed to work well for changing water temperatures.

When using a magnetic tracking system, there are some noise concerns that must be considered. It is important that calibration, as well as scanning, be performed in an area free of excessive ferromagnetic metal. While the *Flock of Birds* tolerates the presence of metal better than do AC magnetic systems, it still does not work well

with large pieces of metal in close proximity. An especially problematic case is where the transmitter or water bath is placed directly on a metal desk or table. If a CRT monitor is being used anywhere near the operating area, it is advisable to use the monitor synchronization cord provided with the *Flock of Birds* to allow it to account for noise transmitted by the monitor.

It is important to make sure that *Stradwin*'s display is turned off before working with the Terason application, as the Terason window will only be available if *Stradwin*'s display is not active. One should ensure that the desired depth is set and verify other probe settings. The focus should be set to the approximate depth of the bottom of the water bath with the probe held under water. It is also generally helpful to reduce the gain and TGC to their minimum values for calibration, as this will diminish reflections and show only the more distinct boundary between the water and bottom of the water bath.

In *Stradwin*, on the 'Record' tab, the 'Video Rate' option should be set to 'Full speed' for calibration. Next, it should be verified that the video stream from Terason is active and visible in *Stradwin* and that the position display (green semi-circle) is shown; the display should look like Figure G.1. Clicking 'Configure image source' will bring up a set of options, and in this dialog, 'Auto Crop Video' can be selected to crop the image to only the visible portion.

Next, the 'Probe Calib' tab can be selected to enter the calibration mode. At this point, one may find it helpful to adjust the sizes of the subwindows in *Stradwin* to make the ultrasound data display larger. The temperature of the water should be measured or approximated and entered into the temperature box, then confirmed by pressing 'Set'. The next step involves setting the scale of the image on the screen to correspond to real distances. One measurement must be specified for each of the x and y axes. The first measurement can be set as the vertical distance by noting the probe depth set in Terason and marking points on the top and bottom of the image corresponding to the full depth. One should make sure to click the 'Finished

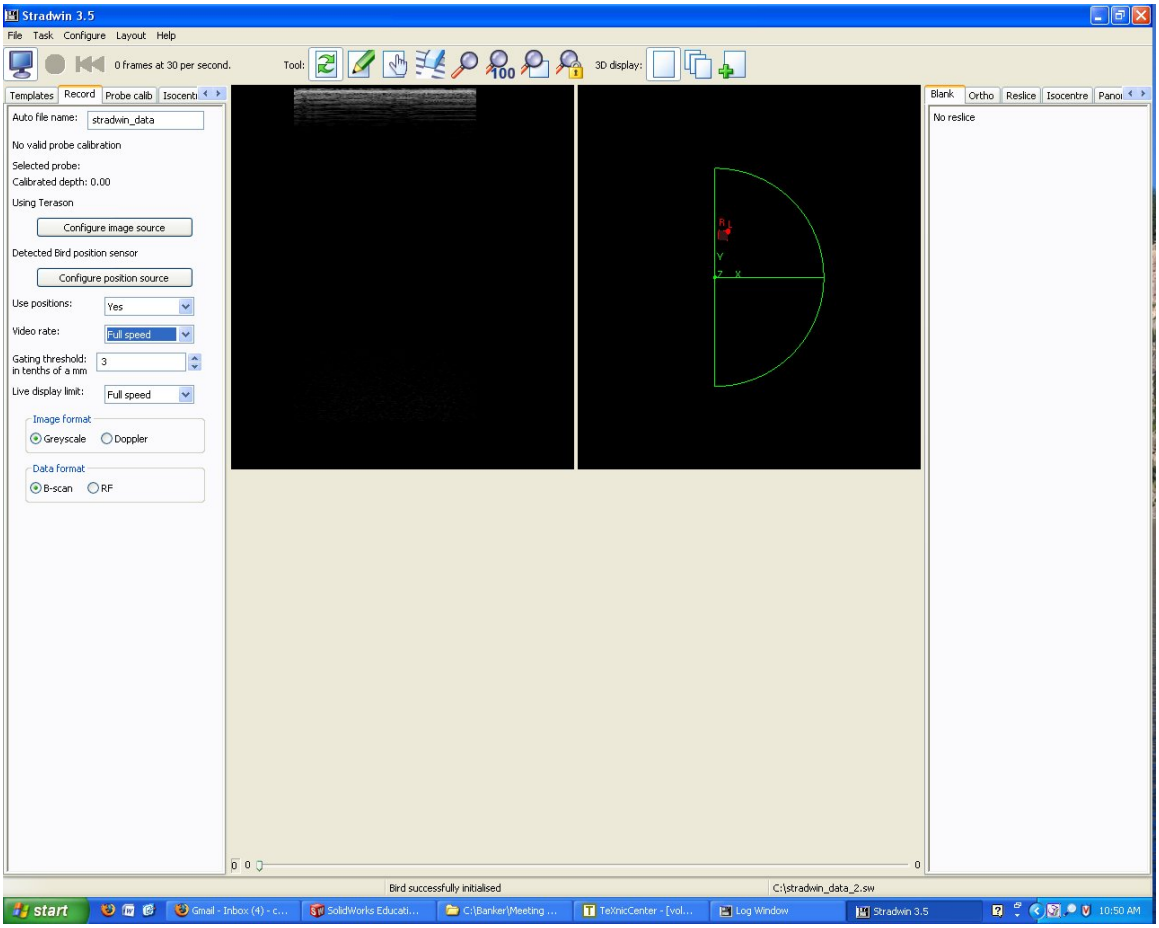


Figure G.1: *Stradwin* Display With Properly Configured Video and Position Sensor

locating ends of line' button when done. The horizontal measurement must be set by using an object of known width, placed in the water bath. This could be a metal or plastic block that has been precisely measured. The object should be placed in the water bath and viewed on the screen with the ultrasound system. The slider must be adjusted to the object's width, then points can be marked on the left and right sides of the object seen on the screen.

A region of interest can be defined to prevent erroneous data from being detected. A good starting range is from the bottom of the display to about the mid-point of the scan plane. After clicking the 'Define region of interest' button, a boundary can be defined by left-clicking at each corner and then right-clicking to set the final point and close the region.

The next step is to perform a specific series of motions and accept lines formed by the bottom of the water bath that *Stradwin* automatically detects. Figure G.3 shows a graphical representation of these motions. An example of what these lines should look like is shown in Figure G.2. To optimize calibration, these motions must be performed carefully. After each probe movement, while holding the probe in place, it is necessary to verify that *Stradwin* has properly detected the line formed by the bottom of the water bath and then click 'Accept Line for Calibration'. The goal of the calibration process is to capture approximately 40 lines. If at any point, a line is accepted that does not represent the water bath, it is necessary to click 'Delete All Lines' and start the series of motions over, as one bad line can ruin the calibration.

The following is a description of the specific probe movements that are required, which are illustrated in Figure G.3:

(a) Place the probe into the water bath at a depth of about 2 cm. The probe should be vertical and not tilted in any axis. Acquire one line at this location. Then move the probe vertically up and down taking a line in each position.

(b) Rotate the probe side to side without significantly lifting or tilting the probe

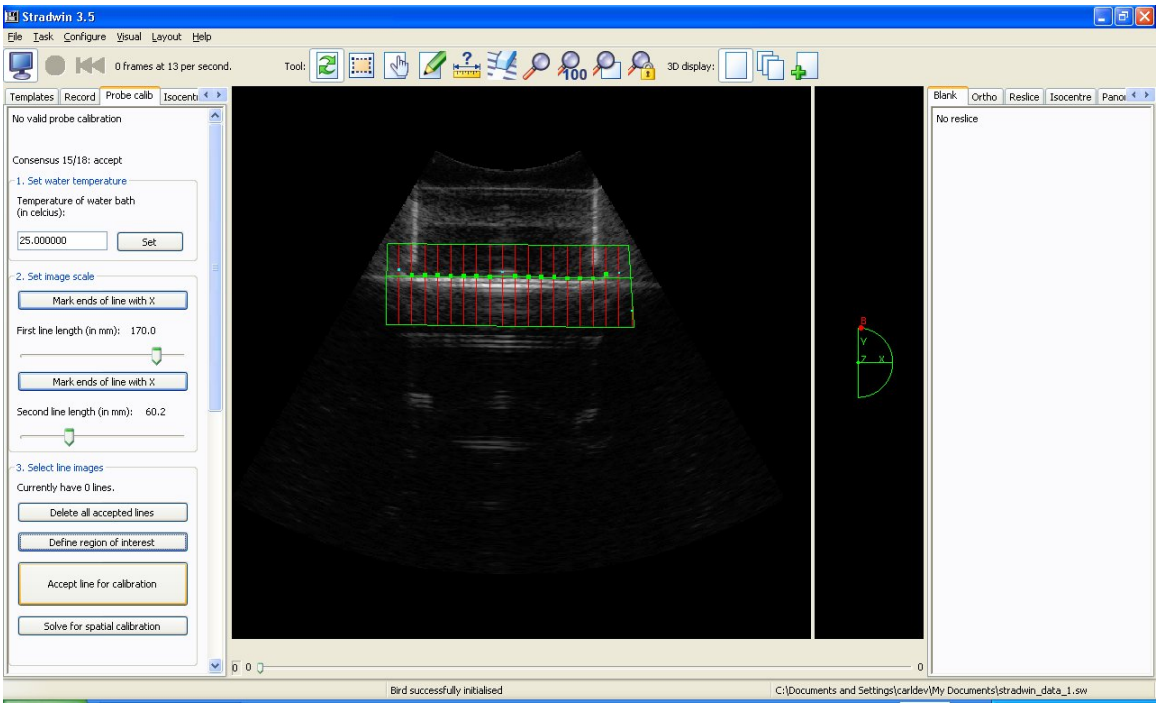


Figure G.2: Capture of a Line for Calibration in *Stradwin*

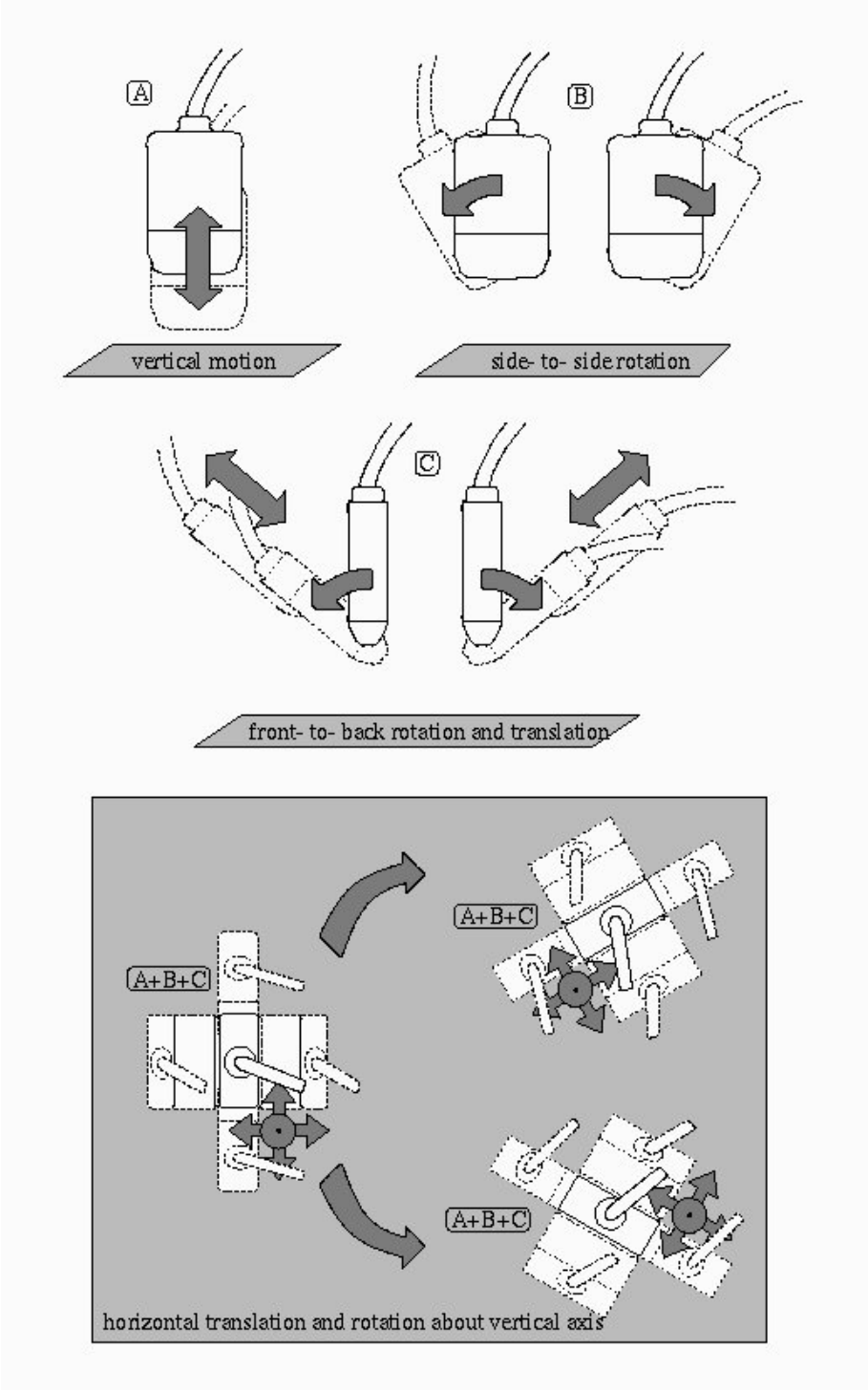


Figure G.3: *Stradwin* Calibration Motion Sequence [28] Depicting Necessary Translations and Rotations for Calibration Process

in any other axis as shown in Figure G.3(B). The rotations should be approximately the same angle in each direction. Accept lines at both rotations.

(c) Rotate the probe forward and accept a line. While keeping the probe oriented the same way, move it up and down in the direction the probe is pointed as shown in Figure G.3(C), taking two more lines. Then rotate the probe backwards to the same angle and repeat to record three more lines.

(d) Rotate the probe 45° in each direction about its vertical axis, accepting two more lines.

(e) Translate the probe to different area of the water bath and repeat steps (A)-(D). Next, repeat a third time in a location that does not form a line with the other two.

If this process is followed properly, 13 lines should have been accepted for each of the three locations, yielding a total of 39 lines. After all lines have been accepted, click 'Solve for Spatial Calibration'.

The final necessary step is to specify the probe name and depth and accept the calibration. The probe should be given a descriptive name, then pressing the 'Enter' key will accept it. Next the depth should be entered and the 'Enter' key pressed again. It is important that 'Enter' is pressed after each of these text entries or *Stradwin* will not allow the calibration to be accepted. Once this has been completed, the 'Accept Calibration' button should be available, and clicking it will accept this calibration.

At this point, the calibration can be saved by saving the current configuration as a template. This can be done by clicking the 'File' menu and then selecting 'Save as template'. It is good practice to give the template a descriptive name, specifying the probe and depth.

Appendix H

Scan Path Visualization Scripts

The scan path visualization feature described in Section 5.3 relies on MATLAB scripts to map the surface of the manikin and overlay the scan path onto the surface.

H.1 Manikin Surface Generation Script

The surface generation script generates a 3D surface mapping of the manikin based on a captured data set.

H.1.1 Manikin Surface Generation Script MATLAB Code

```
% plot_ts.m
% trakSTAR plotting script
% Christian Banker

    plot3(x/1.5, y/1.5, -z/1.5);    % line plot
    grid on;
    xlabel('x [inches]');
    ylabel('y [inches]');
    zlabel('z [inches]');
    view([58 58])

    figure;
    plot3(x/1.5, y/1.5, -z/1.5, '.');   % point plot
```

```
grid on;
xlabel('x [inches]');
ylabel('y [inches]');
zlabel('z [inches]');
view([58 58])

% surface plot
figure;
[xfit,yfit,zfit] = gridfit(x,y,-z,100,100); % gridfit function
colormap(copper(256));  % copper color mapping - close to skin tone
surf(yfit/1.5, zfit/1.5, xfit/1.5); % surface plot
camlight right; % set up lighting
lighting phong;
shading interp
grid on;
hold on;
xlabel('x [inches]');
ylabel('y [inches]');
zlabel('z [inches]');
view([58 58])

%surface plot with superimposed lines
figure;
[xfit,yfit,zfit] = gridfit(x,y,-z,100,100); % gridfit function
colormap(copper(256));  % copper color mapping - close to skin tone
surf(yfit/1.5, zfit/1.5, xfit/1.5); % surface plot
camlight right; % set up lighting and shading
lighting phong;
shading interp
grid on;
hold on;
plot3(x/1.5,y/1.5,-z/1.5+1,'linewidth',2)   % plot new coordinates on top of surface
xlabel('x [inches]');
ylabel('y [inches]');
zlabel('z [inches]');
view([58 58])
hold off;
```

H.2 Scan Path Overlay Script

The scan path overlay script is called by the training software and generates a display of the recorded scan path drawn onto a surface model created using the script in Section H.1. This permits accurate visualization of the scanning path and motion.

A MATLAB GUI was created using the MATLAB Graphical User Interface Development Environment (GUIDE). Guide provides a simple environment for designing a wide range of graphical interfaces for MATLAB scripts. Further details on the script and GUI can be found in Section 5.3.4.

Four different options are available in the MATLAB GUI for this script, which each perform different action. 'Draw Path' instantly draws the entire path. 'Play Path' causes the points to be drawn one at a time at a time step of 0.02 seconds; roughly corresponding to the training system capture rate. 'Fast Play' only displays half of the points to speed up the playback.

H.2.1 Scan Path Overlay Script MATLAB Code

```
function varargout = plot_gui(varargin)
% PLOT_GUI M-file for plot_gui.fig
%      PLOT_GUI, by itself, creates a new PLOT_GUI or raises the existing
%      singleton*.
%
%      H = PLOT_GUI returns the handle to a new PLOT_GUI or the handle to
%      the existing singleton*.
%
%      PLOT_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PLOT_GUI.M with the given input arguments.
%4
%      PLOT_GUI('Property','Value',...) creates a new PLOT_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before plot_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to plot_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help plot_gui

% Last Modified by GUIDE v2.5 23-Dec-2008 22:59:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @plot_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @plot_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before plot_gui is made visible.
function plot_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to plot_gui (see VARARGIN)

% Choose default command line output for plot_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes plot_gui wait for user response (see UIRESUME)
```

```
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = plot_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

    cla(handles.axes1,'reset');
    load 'C:\Banker\SF_svn\gear\gear_1.0.0\proj\glui_test\matlab\charlie-bu-4.mat';
    output = ezread('C:\Banker\SF_svn\gear\gear_1.0.0\proj\glui_test\matlab\log_data.csv', ',');
    global x y z;
    x = output.x/1.5;
    y = output.y/1.5;
    z = output.z/1.5;
    colormap(copper(256));  % copper color mapping - close to skin tone
    h=surf(yfit/1.5, zfit/1.5, xfit/1.5); % surface plot
    view([58 58])
    camlight right; % set up lighting and shading
    lighting phong;
    shading interp
    grid on;
    hold on;
%     axes(handles.axes1)
%     for k = 1:2:length(z),  % run through at fixed rate
%         plot3(x(k),-y(k),-z(k)+2.3,'.')   % plot new coordinates on top of surface
%         pause(0.01)
%     end
    xlabel('x [inches]');
    ylabel('y [inches]');
    zlabel('z [inches]');
    set(hObject,'toolbar','figure');
    guidata(hObject, handles); %updates the handles

% --- Executes on button press in plotpath.
function plotpath_Callback(hObject, eventdata, handles)
% hObject    handle to plotpath (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global x y z;
    axes(handles.axes1)
    for k = 1:1:length(z),  % run through at fixed rate
        plot3(x(k),-y(k),-z(k)+1.5,'.')   % plot new coordinates on top of surface
    end
    guidata(hObject, handles); %updates the handles

% --- Executes on button press in playpath.
function playpath_Callback(hObject, eventdata, handles)
% hObject    handle to playpath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global x y z;
    axes(handles.axes1)
    for k = 1:1:length(z),  % run through at fixed rate
        plot3(x(k),-y(k),-z(k)+2.3,'.')   % plot new coordinates on top of surface
        pause(0.02)         % 20 ms delay
    end
    guidata(hObject, handles); %updates the handles

% --- Executes on button press in fastplay.
function fastplay_Callback(hObject, eventdata, handles)
% hObject    handle to fastplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global x y z;
    axes(handles.axes1)
    for k = 1:2:length(z),  % run through at fixed rate
        plot3(x(k),-y(k),-z(k)+2.3,'.')   % plot new coordinates on top of surface
        pause(0.01)         % 20 ms delay
    end
    guidata(hObject, handles); %updates the handles

% --- Executes on button press in clearpath.
function clearpath_Callback(hObject, eventdata, handles)
% hObject    handle to clearpath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    cla(handles.axes1,'reset');
    load 'C:\Banker\SF_svn\gear\gear_1.0.0\proj\glui_test\matlab\charlie-bu-4.mat';
```

```
output = ezread('C:\Banker\SF_svn\gear\gear_1.0.0\proj\glui_test\matlab\log_data.csv', ',');
global x y z;
x = output.x/1.5;
y = output.y/1.5;
z = output.z/1.5;
colormap(copper(256));  % copper color mapping - close to skin tone
h=surf(yfit/1.5, zfit/1.5, xfit/1.5); % surface plot
view([58 58])
camlight right; % set up lighting and shading
lighting phong;
shading interp
grid on;
hold on;
xlabel('x [inches]');
ylabel('y [inches]');
zlabel('z [inches]');
guidata(hObject, handles); %updates the handles
```

Appendix I

gen_volume - Volume Generation Tool

This tool is capable of generating 3D volumes in raw, ITK, Analyze or *Stradx* formats based on a series of equations specified within the code. It generates all appropriate header files for each format and is also capable of generating multi-sweep volumes for the *Stradx* format.

I.1 Input, Output and Options

This program takes no input files and outputs appropriate data and header files in the selected format.

The equations that determine the generated data within the volume must be manually specific within the code as a series of if...else...else if statements.

The first option is the filename for the volume. This should be entered without an extension and all generated files will be composed of this filename plus the appropriate extension. Next the user is requested to select between the four available file formats, which are described in sections I.1.1 - I.1.4.

I.1.1 .raw with no header

This generates a simple raw data file with no header information. The file is generated with a .raw extension, however this can be renamed to any extension required. No header information is saved, so it is important to note the settings selected for the dimensions of the volume. At a later time, the makesx utility described in Appendix J can be used to generate *Stradx* headers if the the dimensions of the volume are known. As long as the dimensions and data type for a raw file are known, it can be used with Gear.

The only required options for this format are the x, y, and z dimensions of the volume in voxels.

I.1.2 ITK

The ITK (Insight Segmentation and Registration Toolkit) format is a popular voxel-based format that consists of a raw .vol file and an ASCII header with a .mhd extension. The header includes basic information such as the number of dimensions, data type, spacing between voxels and dimensions of the volume. Various programs are able to view these files, including *VolSuite*. This is a useful file format for working with voxel-based data. *gear* is able to read the raw data files if the dimensions and data type are provided and having the ITK header allows the volumes to be opened in other programs for viewing.

The only required options for this format are the x, y, and z dimensions of the volume in voxels.

I.1.3 *Analyze*

Mayo Clinic's *Analyze* file format is similar to the ITK format, except that the header file is in a binary format and cannot simply be edited with a text editor. The data files are exactly the same as are used in the ITK format, except that these

files must have an extension of .img. The header file has an extension of .hdr and contain similar information to the ITK header, except that it is written in binary format. This file format is accepted by many programs including *Analyze* and AIR [52]. While this file format is useful for permitting volumes to be opened in programs that require this format, the difficulty of generating and viewing binary header files makes it less desirable for general-purpose use than the ITK format. The data files for this format are able to be used with *gear*, provided that the dimensions and data type are known.

The only required options for this format are the x, y, and z dimensions of the volume in voxels.

I.1.4 *Stradx*

The *Stradx* file format is much different from most volume file formats, as it is slice-based rather than voxel-based. *Stradx* data consist of a series of two-dimensional slices along with information on how to position them. Although it is an unusual way of storing data, it is logical for a format developed specifically for freehand ultrasound data. The header file includes position information for each slice. This format also includes a calibration file that includes basic information like slice dimensions, along with more advanced calibration information that can be omitted or replaced with a constant value for simple generated volumes like these.

The first required option is the number of sweeps. This selects whether the data consist of a single sweep or two overlapping sweeps. If two sweeps are selected the x-axis offset between the two sweeps must be entered. This is the difference in position between the left side of the first sweep and the left side of the second sweep.

The dimensions of the volume must also be entered. For a dual-sweep volume, these are the dimensions of each sweep.

Appendix J

makesx - *Stradx* Header Generation Tool

The *makesx* tool can generate the appropriate *Stradx* header and calibration files for a voxelized 3D volume. It is capable of generating headers for a volume containing either 1 or 2 sweeps. This is especially useful for converting volumes from a raw data format into *Stradx* and can also be used to apply different offsets to a *Stradx* volume that has regularly spaced slices.

J.1 Input, Output and Options

This program takes no input files and outputs *Stradx* header (.sx) and calibration (.sxc) files. It is expected that there exists a raw data file with a .sxi extension of the same filename being used for these files.

The program first asks for a filename with no extension, which should match the name of the .sxi data file you have created. It then requests whether this will be a single or dual-sweep volume.

For the single volume case the program requests values for the x, y, and z dimensions of the volume in voxels. These must exactly match the dimensions of the

volume in data file.

For a dual-sweep volume, it requests x, y, and z dimensions for the first volume and a z dimension for the second volume. It is required that the two volumes have the same x and y dimensions, which is typically the case for data captured from an ultrasound scanner. Next, the x, y, and z offsets between sweeps are requested. Note that the x offset is the offset between the left edge of the first sweep and the left edge of the second sweep. If this value is not known, it can be calculated by subtracting the sweep overlap from the x-dimension of the first volume.

The next option for either number of sweeps is the voxel spacing, which is the size of a voxel in each dimension. Currently, this tool only supports isotropic spacing, where voxels are cubic. A typical value for this setting is 0.1 cm/voxel, which equates to 1 mm per voxel. It should be noted that *Stradx* can only account for up to 20 mm of offset in any axis, so the voxel spacing should be set to ensure that no offset is more than 20 mm when the voxel offset is multiplied by the voxel spacing. Once the program prints that the header file and calibration file have both completed, the necessary files have been generated. Ensure that these two files are in the same folder as the .sxi data file and open the volume by selecting the .sx file in *Stradx*.

Appendix K

ts_noisex - *trakSTAR* Measurement Rate Determination Utility

The *ts_noisex* program is designed to capture the necessary data to determine the optimal measurement rate for a given operating environment. It provides a table of data points representing measurement rate and noise pairs, which can be used to select a measurement rate that minimizes noise.

It is modeled after Ascension Technology Corporation's *Noise-X* program for their *Flock of Birds* system. Because no such utility currently exists for the more recent *trakSTAR*, it was necessary to design a custom program to perform this function.

This program works by sweeping through the entire range of measurement rates and capturing noise measurements at each point.

K.1 Inputs, Outputs, and Options

This program takes no inputs and outputs a Comma Separated Value (CSV) file consisting of entries of the form 'meas_rate,quality1, quality2, ... qualityN', where

N is an option controlling the number of data points per measurement rate for use when performing averages. The other user-controlled option is PTS, the number of measurement rate data points to test. This is the number of subdivisions that are made within the entire measurement rate range.

Appendix L

ts_capture - *trakSTAR* Data Capture Utility

This program is designed to capture a stream of data from the *trakSTAR* system and record it to a Comma Separated Value (CSV) file. This data can be used to determine noise on individual channels or to generate an accurate mapping of a surface like was done for the manikin surface mapping.

This program is essentially the same as the *trakSTAR* read functions used in the simulation system, but designed into a simple program that only reads the tracking system.

L.1 Inputs, Outputs, and Options

This program takes no inputs. The output is a CSV table consisting entries of the form 'x,y,z,a,e,r,quality', where x, y, and z are the translations, a, e, and r are the rotations and quality is a quality factor that is indicative of the amount of noise present. The only user-modifiable option is the read rate, in Hz, which can be specified in the header file.

Appendix M

Calibration Fixture Manufacturing Procedure

This section describes the procedure of manufacturing the calibration fixture and lists required materials and tools. It is expected that anyone attempting to manufacture one of these fixtures has a working knowledge of machining practices and familiarity with the tools required.

M.1 Materials

- At least 1" length of 3" diameter solid aluminum rod - McMaster-Carr Part # 9034K51
- At least 6" length of 0.062" x 3/4" brass bar - McMaster-Carr Part # 8954K61
- Sheet of 1/4" aluminum, at least 4"x4.5"
- 2-part epoxy designed to bond metals
- 4x 1/4-20 x 4"bolts, 300-series stainless steel
- 4x 1/4-20 nylon wing nuts
- 4x 1/4" nylon or 300-series stainless steel washers

- 4x 1/4” 300-series stainless steel lock washers
- 1/4” thick, 1” wide foam tape, at least 4” length

M.2 Tools

- Metal cutting lathe
- Band saw with metal cutting blade
- Milling machine
- Fly cutter for milling machine
- 1/2” end mill for milling machine
- 1/16” end mill for milling machine - McMaster-Carr Part # 2802A81
- Drill bit attachment for milling machine
- 1/4” drill bit
- Accurate vernier caliper
- Various files and deburring tools

M.3 Manufacturing Process

The following sections outline the process of machining each part and assembling the entire fixture.

M.3.1 Cutting and Milling Wheels

1. Turn 3” aluminum rod on lathe to clean outside diameter
2. Use lathe to clean front face
3. Use a band saw to cut off approximately a 1/4” thick disc

4. Put remaining bar back into lathe and clean the front face again
5. Cut off a second 1/4" disc with the band saw
6. Attach an appropriate fly cutter to the milling machine with a radius of at least 4"
7. Mount one of the discs in the milling machine's clamp, supported by parallels with the cleaned face down
8. Clean the face and mill down to a thickness of .200"
9. Repeat to mill the second disc
10. Deburr as necessary

M.3.2 Cutting and Machining Brass Bar

1. Cut off a section of the brass bar slightly longer than 6" in length using a band saw
2. Affix the 1/2" end mill to the milling machine
3. Mount the bar in the milling machine with the end just cut facing toward the end mill
4. Clean the face of the bar
5. Turn the bar around and clean the other face, then machine it until the total length of the bar is 6"
6. Mount the bar in the milling machine clamp so that the 3/4" side is vertical, with at least .20" extending above the top of the clamp
7. Machine the bar until its height is .600"
8. Deburr as necessary

M.3.3 Cutting and Machining the Clamp Plates

1. Using the band saw, cut two rectangles out that are slightly larger than 2"x3.15". It is preferable for at least one face to lie on a cleaned edge of the sheet of aluminum

2. Use the milling machine with the 1/2" end mill to clean all four side faces of each of these aluminum rectangles.

3. Machine each to a finished size of 2.00" x 3.15"

4. Deburr as necessary

M.3.4 Cutting 1/16" Grooves for the Brass Bar

1. Mark the center point on each wheel and a point .60" from the center

2. Attach the 1/16" end mill to the milling machine

3. Mount one of the wheels in the milling machine

4. Machine a very thin groove from one mark to the other

5. Repeat in increments of approximately 0.010" - 0.020" to slowly mill through the wheel. This must be done slowly and in small increments to avoid breaking the small 1/16" end mill.

6. Once a groove has been cut all the way through, check the length of the slot and test the fit of the brass bar. Incrementally increase the length of the slot as necessary until the bar just slides into the slot.

7. Repeat for the other wheel.

8. Make marks on the aluminum plates from the center of one short edge to 1.20" up from there, as shown in the drawings.

9. Mount one of the plates in the milling machine
10. Cut a 1/16" groove in the same manner as the wheels
11. Check that the brass bar slides smoothly through the groove, adjust as necessary
12. Repeat for the other clamp plate

M.3.5 Affixing wheels to brass bar

1. Mix 2-part epoxy as specified on packaging
2. Apply epoxy to the inside of the slot on one wheel and the outside of the bar where they will make contact.
3. Slide the wheel onto the bar and allow to dry. If the fit is loose, they may need to be clamped in place.
4. After the epoxy for the first wheel has set, affix the second wheel in the same fashion.

M.3.6 Drilling Holes in Clamp Plates

1. Mark the locations of the 4 holes on each clamp plate as shown in the drawings.
2. Attach a 1/4" drill bit to the milling machine with a drill bit attachment.
3. Line the clamp plates up on top of each other and drill the 4 holes through both plates simultaneously.
4. Clean the edges of the holes with a counter-sink drill

M.3.7 Assembly

1. Ensure that the clamp plates each slide smoothly on the brass bar, file as necessary.

2. Attach a 1.5” strip of foam tape to each clamp plate, centered horizontally and spanning from the top of the plate to just above the top of the groove.

3. Put the clamp plates and hardware onto the 4” bolts in the following order: lock washers, plate 1, plate 2, washers, wing nuts

4. The clamp assembly may now be placed on the bar and wheel assembly.

5. A transducer can be mounted in the clamp assembly by tightening each of the wing nuts one turn at a time until the transducer is firmly affixed in place.

Appendix N

Demonstration Guide

N.1 Introduction

This guide provides step-by-step instructions for setting up and performing a demonstration of the Interactive Training System for Medical Ultrasound. It describes the hardware and software setup and the details of running a demonstration. A troubleshooting guide is also provided in section N.5. The appendices provide some additional information on initial setup and how to access the subversion repository where the code is stored.

This is meant to be a complete guide to cover all situations that may be encountered. However, for the majority of demos the setup will be very straightforward. For a basic demo, only sections N.2 and N.4 are necessary. If the hardware has already been set up appropriately, it may be possible to start directly from section N.4.

N.2 Hardware Setup

The first step to preparing a demo is to set up the hardware and make all necessary electrical and data connections. This section details the required hardware and the methods for setting it up.

The following hardware is required for a demonstration:

- Ascension *trakSTAR* 6 degree of freedom (DoF) Tracking System
- "Choking Charlie" manikin modified with appropriate cutouts
- Manikin mounting board with *trakSTAR* transmitter attached
- 2 sham transducers: linear and convex array
- A properly configured lab computer with MATLAB software installed. The two currently configured systems are the Lenovo ThinkPad T61 laptop or the Dell Precision 490 desktop computer.
- Touch Screen Monitor
- An available internet connection, if a network license for MATLAB is being used.
- Power strip, or at least 2 available outlets.

The proceeding steps outline the hardware setup process:

1. Carefully place the manikin on the mounting board, ensuring that the rods on the board slide into the fittings on the manikin. The manikin should rest flat on the board when mounted correctly.

2. Position the *trakSTAR* control unit at least 18 inches away from the operating area where the manikin will be placed.

3. Plug the transmitter cable coming out of the manikin into the *trakSTAR* control unit. The plug will only fit in one orientation, so twist it until it audibly clicks into place.

4. Connect the two sham transducers to ports 1 and 2 of the *trakSTAR* control unit in the same manner as the transmitter. Which port each sham transducer is connected to does not matter, except that the one connected to port 1 will be the default transducer at startup.

5. Connect the *trakSTAR*'s USB cable to an available USB port on the PC.

6. Verify that the *trakSTAR* is plugged in and turn the power switch, located on the back of the *trakSTAR* system, to the 'I' position to power it up. The LED should initially be solid orange for several seconds and then begin blinking green.

7. Connect the touch screen's monitor cable to the monitor output of the computer. Connect the monitor's USB cable to an available USB port on the computer. Plug the green monitor audio cable into the audio output port of the computer, which is typically colored green. Plug in the power cord of the monitor.

8. Power on the computer system.

After all of these steps have been completed, the hardware is now set up and ready for use.

N.3 Software Setup

If this is the first time this software is being set up on a new system, there are some initial one-time steps that must be first performed, which are described in Appendix N.6.

The software setup process depends on whether or not the code needs to be rebuilt. The following circumstances would require rebuilding:

- The code has been downloaded to a new directory

- Updated code has been downloaded from the SVN
- The code has been modified

If none of these circumstances exist, the steps described in this section are unnecessary and the demonstration may be run as described in sectionN.4.

If a rebuild is necessary, the following steps describe the process of rebuilding the software:

1. Ensure that the code is the desired version. It can be updated to the latest revision by following the instructions in Appendix N.7.1.
2. Open the software project by double-clicking the icon for 'Shortcut to cbanker_dev.sln' on the desktop. Alternatively, it can be accessed by opening Microsoft Visual C++ .NET 2003 and loading the solution 'cbanker_dev.sln' from '[SVN Root]\gear\gear_1.0.0\proj\'
3. Right click on the project 'glui_test' and select 'Build Project'
4. If the message window at the bottom of the screen shows 'Build: 1 succeeded, 0 failed, 0 skipped', the project has been built successfully and is ready to be run.

N.4 Performing the Demonstration

The following steps describe the process of running a demonstration:

1. Ensure that the *trakSTAR* system is connected and powered on as described in section N.2.
2. If necessary, load and rebuild the project 'glui_test' as described in section N.3.

3. To use the Scan Path Playback feature, ensure that a MATLAB license is available. If the license being used is a network license, it will require an active internet connection. For off-campus demonstrations, the WPI VPN must be connected to access the MATLAB license server. Presence of a valid MATLAB license can be verified by running the MATLAB software; if no errors occur, it should be functional.

4. Place the sham transducer that is connected to port 1 on top of the manikin's abdomen; the exact position is not important. This ensures that the sham transducer is in the top hemisphere at the time of initialization. .

5. To run the software, double-click on the 'Ultrasound_Simulation' icon located on the desktop. Initially a black command line window will appear, and then a graphical display will open.

6. Once the graphical display appears, hold the sham transducer connected to port 1 such that it is centered on the naval of the manikin. Press the 'Recenter Probe' button to calibrate the center point for the sham transducer.

7. You may now freely scan the manikin. The transducer should be held with the large groove toward the manikin's right side, which corresponds to the thumb for right-handed scanning. See section N.4.1 for information on how to perform common tasks.

8. When the demonstration is complete, click the 'Exit' button in the upper-right corner of the screen to exit the software. Power down the *trakSTAR* system.

N.4.1 Common Operating Tasks

This section details some common tasks, along with descriptions of how to accomplish them.

Exiting the program: Click the 'Exit' button at the top right of the screen, which is a red button with a white 'X'. Alternatively, this can also be done by pressing the Esc key on the keyboard while the graphical window is the currently selected window.

Re-center the sham transducer: Hold the transducer at the desired location and click 'Recenter Probe' or press the space bar. The naval is typically a good location to center the probe.

Change to the other sham transducer: Select a different probe geometry from the dropdown menu in the GUI. Pick up the other sham transducer and click 'Recenter Probe'.

Perform a screen capture: Click the 'Screen Capture' button on the GUI or press the 's' key on the keyboard. A capture will be saved as 'capture#.jpg', where # is the sequential number of the image within this session. These files will be stored in the program directory, which is currently "C:\Banker\SVN\gear\gear_1.0.0\proj\glui_test". Note that file numbering is reset after the program is closed, so files should be relocated or renamed prior to starting another session.

View scan path: After recording a scan path, click on 'View Scan Path'. This will freeze the screen, stop recording, and close the recording file. The MATLAB engine will then be called and the manikin surface will be displayed. The scan path can then be displayed or played back using the buttons on the plot. To resume scanning, the 'Un-Freeze' button must be clicked, which will start a new scan path recording.

Start new scan path recording: The scan path recording is reset when the display is frozen and then un-frozen or after the scan path is viewed.

Perform region of interest selection: Draw a line using the touch screen to specify two opposite corners of a rectangle. A red selection box should appear. Note that this selection is reference to the current location within the volume and moving the sham transducer may cause the selection to leave the current view. Click 'Check

Selection' to evaluate the selection. This will show either 'Correct' or 'Incorrect' in the 'Selection Result' box in the GUI.

Load a different image volume: To load a different image volume, the startup parameter for the program must be changed. First ensure that the desired image volume and it's corresponding header file are located in the program directory, "C:\Banker\SVN\gear\gear_1.0.0\proj\glui_test". Next, right-click the 'Ultrasound_Simulation' icon on the desktop and select 'properties'. Modify the target field to show: "C:\Banker\SVN\gear\gear_1.0.0\proj\glui_test\Debug\glui_test.exe desired_volume_header.ghd", where "desired_volume_header.ghd" is the header file for the volume to be loaded.

The currently available volume options are 'Patient1_rot.ghd' and 'CIRS-stitched.ghd', which are the prostate and CIRS phantom volumes, respectively.

N.5 Troubleshooting

This section describes some potential problems that may be encountered, along with suggestions for solving them.

The observed image comes out upside down:
Likely Cause: Sensor is in the wrong operational hemisphere at startup.

- Exit and restart the software. Ensure that the sham transducer is resting on the manikin's abdomen at startup.

No graphical window appears and the program either freezes or exits:
Likely Cause: *trakSTAR* is not being properly detected or no data volume is being loaded.

- Ensure that the *trakSTAR* is correctly connected and powered on.
- Check that the specified data volume and header file are available in the program folder.

Visualization is excessively noisy (image changes without moving trans-

ducer):

Likely Cause: Excessive metal nearby is causing interference.

- Ensure that there are not excessive amounts of metal in the area.
- Verify that the *trakSTAR* control unit is at least 18 inches away from the manikin.
- Run *ts_noisex* to determine an optimal measurement rate for the current environment.

An image appears on the screen but cannot be controlled by the sham transducer:

Likely Cause: The *trakSTAR* system is not connected or powered on, or an error has occurred.

- Verify the *trakSTAR* power and communication connections. The LED should be blinking green before starting the program.
- Ensure that upon starting the program, the *trakSTAR* LED starts out solid orange during initialization (about 5-10 seconds) and then turns solid green.
- View the console window during operation to see if any error messages are reported.

The Scan Path Playback feature does not work:

Likely Cause: The MATLAB software is unavailable or is not configured properly.

- Ensure that a valid internet connection is available and that the WPI VPN is running and connected if this demonstration is being performed off-campus.
- Check that the MATLAB software will run and can execute commands.
- Verify that the initial setup described in Appendix N.6 has been performed for this computer.
- Check that the proper project MATLAB directory is included in the MATLAB path by type “path” at the MATLAB command line.

- Re-register the MATLAB COM server as described in Appendix N.6.
- Ensure that MATLAB R2007a is installed and is the default version.

N.5.1 Further Assistance

If none of the suggestions in the troubleshooting guide are helpful in resolving a given problem, Christian Banker can be contacted for additional assistance. He can be reached by email at cbanker@gmail.com or by phone at (860)705-7818. Email is checked frequently and preferred for non-urgent communication.

N.6 Appendix: Initial One-Time Setup Instructions

The following steps must be performed once to set up a new computer system to run the software.

1. Ensure that MATLAB is installed on the system. Only version R2007a has been fully validated, but other versions should work. If version R2007a is not installed, but is available, it may be worthwhile to test with this version.

2. On the windows command line, register the MATLAB server. To do this, click on 'Run' from the Start Menu and type "cmd" to access the command line. Run the following two commands: "cd $MATLAB\bin\win32", then"matlab /regserver". This will ensure that the MATLAB engine is registered as a Windows COM server.

3. Add the project's MATLAB directory to MATLAB's default PATH. To do this, enter the following in the MATLAB command line: "path(path,'path to project MATLAB directory')", then "savepath". Replace "path to project MATLAB directory" with the current path within the project. Right now this is "C:\Banker\SVN\gear\gear_1.0.0\proj\glui_test\matlab", but may change if a different top directory or project name is used.

N.7 Appendix: Subversion (SVN) Repository Instructions

Subversion, or SVN, is a version control system that allows versions of files to be stored to an online server as changes are made. This keeps backups of the code available and allows the ability to revert to a previous version of software at any

time. This appendix describes some of the basic tasks that may occasionally be relevant in the course of performing demonstrations. These instructions assume that TortoiseSVN, available from `http://tortoisesvn.tigris.org/`, is installed. This software is available on both the lab computer and the laptop. It also assumed that the appropriate username and password have been stored on the computer for access to the repository; contact Chris Banker (cbanker@gmail.com) if this is not the case.

Additional instructions and information can also be found on the TortoiseSVN website: `http://tortoisesvn.tigris.org/`.

N.7.1 SVN Update

Performing a SVN update downloads the latest files from the SVN server and uses them to update the files currently on your computer. This ensures that the computer you are using has the latest code that has been committed to the server. This may be necessary when changes have been made on another computer system and you wish to use this new version.

To perform a SVN update, perform the following steps:

1. Locate the folder on your computer where you have the software stored. On the laptop this should be C:\Banker\SVN. If no modifications have been made, the folder should have a green check mark. If it has a red exclamation point, it means that files have been modified.

2. If any files have been modified and a red exclamation point is shown on the folder icon, the files should be reverted before updating to prevent conflicts.

   (a) Right click on the folder and go to the 'TortoiseSVN' submenu.

   (b) Under that menu select 'Revert'.

   (c) Click 'OK' in the next dialog. The process of reverting the files will now commence.

(d) When the process has finished an acknowledgment dialog will come up. Click ‘OK’ in this dialog.

(e) The files have now been reverted to their original state.

3. Right click on the folder and select the ‘SVN Update’ menu item. The update process will commence.

4. Upon completion, an acknowledgment dialog will come up. Click ‘OK’ in this dialog.

5. The files have now been updated to the latest revision available on the server.

N.7.2 Viewing SVN Logs

The SVN repository allows users to enter text into the log to describe the most recent changes. The log also contains a list of the files that were modified in a given revision. Viewing these logs can be especially useful for verifying that the code is actually the desired version. If in doubt about what changes have been made, checking the logs is a good first step.

The SVN log can be viewed by taking the following steps:

1. Locate the folder that you are interested in viewing logs for. The displayed log entries will only be those relevant to the selected folder and any files and folders beneath it.

2. Right click on the folder and select the ‘TortoiseSVN’ submenu.

3. Under that menu select ‘Show Log’.

4. A log window will be displays that shows recent revisions, along with the author, date, log message and files that were modified.

5. To exit this dialog, click ‘OK’.

Bibliography

[1] 3DConnexion. 3DConnexion SpaceNavigator 3D mouse. http://www.3dconnexion.com/3dmouse/spacenavigator.php [Last accessed 3 February 2009].

[2] D. Aiger and D. Cohen-Or. Real-time ultrasound image simulation. *Real-Time Imaging*, 4:263–274, 1998.

[3] AnalyzeDirect. Analyze 8.1 visualization and analysis software. http://www.analyzedirect.com [Last accessed 3 February 2009].

[4] Ascension Technology Corporation. Ascension 3D Guidance trakSTAR. http://ascension-tech.com/medical/trakSTAR.php [Last accessed 3 February 2009].

[5] Ascension Technology Corporation. Ascension flock of birds. http://ascension-tech.com/realtime/RTflockofBIRDS.php [Last accessed 3 February 2009].

[6] Ascension Technology Corporation. Technical description of DC magnetic trackers. http://www.5dt.com/downloads/3rdparty/fobtechnicalspec.pdf [Last accessed 3 February 2009], 2002.

[7] Ascension Technology Corporation. Magnetic fact sheet: DC vs. AC tracking. http://ascension-tech.com/docs/ASCWhitePaperDCvAC.pdf [Last accessed 3 February 2009], 2008.

[8] J. Bryan. VolSuite: A portable scientific application framwork. http://www.osc.edu/archive/VolSuite [Last accessed 3 February 2009].

[9] J. Bryan. personal communication through email, 2008.

[10] J. Dey, J. M. O'Connor, Y. Chen, and S. J. Glick. Temporal change analysis for improved tumor detection in dedicated CT breast imaging using affine and free-form deformation. volume 6913, page 69131D. SPIE, 2008.

[11] DTECTA. SOLID collision detection library. http://www.dtecta.com/ [Last accessed 3 February 2009], 2009.

[12] H.-H. Ehricke. SONOSim3D: a multimedia system for sonography simulation and education with an extensible case database. *European Journal of Ultrasound*, 7(3):225 – 230, 1998.

[13] A. Gee, G. Treece, R. Prager, and L. Cash, C.J.C.and Berman. Rapid registration for wide field of view freehand three-dimensional ultrasound. *Medical Imaging, IEEE Transactions on*, 22(11):1344–1357, 2003.

[14] D. Hill, P. Batchelor, M. Holden, and D. Hawkes. Medical image registration. *Physics in Medicine and Biology*, 46:R1–R45(1), 2001.

[15] HowStuffWorks. HowStuffWorks: How do touch screen monitors know where you're touching. http://computer.howstuffworks.com/question716.htm [Last accessed 3 February 2009].

[16] Inition. Inition - Ascension Flock of Birds. http://www.inition.co.uk/inition/product.php?URL_=product_mocaptrack_ascension_flockofbirds&SubCatID_=18 [Last accessed 3 February 2009].

[17] *International Vocabulary of Metrology - basic and general concepts and associated terms (VIM)*, volume 200. 3rd edition, 2008.

[18] Kitware, Inc. The visualization toolkit. http://www.vtk.org [Last accessed 3 February 2009].

[19] Laerdal Medical, AS. Choking charlie. http://www.laerdal.com/document.asp?subnodeid=7670740 [Last accessed 3 February 2009].

[20] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens. Multimodality image registration by maximization of mutual information. *Medical Imaging, IEEE Transactions on*, 16(2):187–198, Apr 1997.

[21] D. Magee and D. Kessel. A computer based simulator for ultrasound guided needle insertion procedures. *IEE Conference Publications*, 2005(CP509):301–308, 2005.

[22] H. Maul, A. Scharf, P. Baier, M. Wustemann, H. H. Gunter, G. Gebauer, and C. Sohn. Ultrasound simulators: experience with the sonotrainer and comparative review of other training systems. *Ultrasound in Obstetrics and Gynecology*, 24:581–585(5), October 2004.

[23] MITK. MITK: Medical Imaging Interaction Tookit. http://www.mitk.org/ [Last accessed 3 February 2009].

[24] J. J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116. Springer, Berlin, 1977.

[25] NOAA 200th feature stories: The high-precision transcontinental traverse: Accuracy vs precision illustration. http://celebrating200years.noaa.gov/magazine/tct/accuracy_vs_precision.html [Last accessed 3 February 2009].

[26] OpenGL.org. OpenGL: The industry standard for high performance graphics. http://www.opengl.org [Last accessed 3 February 2009].

[27] Planar. Planar: LCD desktop monotir, flat panel display, projection products for medical imaging, kiosk, touch screen, business and home uses. `http://www.planar.com` [Last accessed 3 February 2009].

[28] R. Prager. The Stradwin 3D ultrasound acquisition and visualisation system. `http://mi.eng.cam.ac.uk/~rwp/stradwin/` [Last accessed 3 February 2009].

[29] R. Prager. The Stradx 3D ultrasound acquisition and visualisation system. `http://mi.eng.cam.ac.uk/~rwp/stradx/` [Last accessed 3 February 2009].

[30] R. Prager, A. Gee, and L. Berman. Stradx: real-time acquisition and visualization of freehand three-dimensional ultrasound. *Medical Image Analysis*, 3:129–140(12), June 1999.

[31] R. Prager, A. Gee, G. Treece, C. Cash, and L. Berman. Using image-based regression to acquire freehand 3D ultrasound. *Biomedical Imaging, 2002. Proceedings. 2002 IEEE International Symposium on*, pages 970–973, 2002.

[32] R. W. Prager, R. N. Rohling, A. H. Gee, and L. Berman. Rapid calibration for 3-D freehand ultrasound. *Ultrasound in Medicine and Biology*, 24(6):855 – 869, 1998.

[33] P. Rademacher. GLUI user interface library. `http://glui.sourceforge.net/` [Last accessed 3 February 2009], 2006.

[34] SpaceNavigator mouse joystick keyboard driver by RBC9. `http://rbc.duckinegg.com/` [Last accessed 18 January 2009].

[35] A. Roche, G. Malandain, N. Ayache, and X. Pennec. Multimodal image registration by maximization of the correlation ratio. Technical Report RR-3378, 1998.

[36] A. Roche, G. Malandain, X. Pennec, and N. Ayache. The correlation ratio as a new similarity measure for multimodal image registration. In *MICCAI '98: Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 1115–1124, London, UK, 1998. Springer-Verlag.

[37] D. Rueckert. Image registration toolkit. `http://wwwhomes.doc.ic.ac.uk/~dr/software/` [Last accessed 3 February 2009].

[38] D. Rueckert, L. Sonoda, C. Hayes, D. Hill, M. Leach, and D. Hawkes. Non-rigid registration using free-form deformations: application to breast MR images. *Medical Imaging, IEEE Transactions on*, 18(8):712–721, Aug 1999.

[39] D. Rueckert, L. I. Sonoda, E. R. Denton, S. Rankin, C. Hayes, M. O. Leach, D. L. Hill, and D. J. Hawkes. Comparison and evaluation of rigid and nonrigid registration of breast MR images. In K. M. Hanson, editor, *Proc. SPIE Vol. 3661, p. 78-88, Medical Imaging 1999: Image Processing, Kenneth M. Hanson; Ed.*, volume 3661 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 78–88, May 1999.

[40] J. A. Schnabel, D. Rueckert, M. Quist, J. M. Blackall, A. D. Castellano-Smith, T. Hartkens, G. P. Penney, W. A. Hall, H. Liu, C. L. Truwit, F. A. Gerritsen, D. L. G. Hill, and D. J. Hawkes. A generic framework for non-rigid registration based on non-uniform multi-level free-form deformations. In *MICCAI '01: Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 573–581, London, UK, 2001. Springer-Verlag.

[41] D. Shriner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.1.* Addison-Wesley, sixth edition, 2008.

[42] Sound Technology, Inc. . Sound technology inc., state college, pa, ultrasonic transducers and probes. http://www.sti-ultrasound.com/ [Last accessed 3 February 2009].

[43] Teratech Corporation. Terason 2000 ultrasound system. http://www.terason.com/products/t2000.asp [Last accessed 3 February 2009].

[44] Teratech Corporation. Terason t3000 ultrasound system. http://www.terason.com/products/t3000.asp [Last accessed 3 February 2009].

[45] Teratech Corporation. Terason ultrasound systems. http://www.terason.com [Last accessed 3 February 2009].

[46] C. Terkamp, G. Kirchner, J. Wedemeyer, A. Dettmer, J. Kielstein, H. Reindell, J. Bleck, M. Manns, and M. Gebel. Simulation of abdomen sonography. evaluation of a new ultrasound simulator. *Ultraschall Med*, 2003.

[47] J. Varandas, P. Baptista, J. Santos, R. Mertins, and J. Dias. VOLUS - a visualization system for 3D ultrasound data. *Ultrasonics*, 42, 2004.

[48] P. Viola and I. Wells, W.M. Alignment by maximization of mutual information. *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 16–23, 20-23 Jun 1995.

[49] W. Wells, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis. Multi-modal volume registration by maximization of mutual information. *Medical Image Analysis*, 1(1):35–51, 1996.

[50] Accuracy and precision - Wikipedia the free encylopedia. http://en.wikipedia.org/wiki/File:Accuracy_and_precision.svg [Last accessed 3 February 2009].

[51] R. Woods, S. Cherry, and J. Mazziotta. Rapid automated algorithm for aligning and reslicing PET images. *Journal of Computer Assisted Tomography*, 16:620–633, 1992.

[52] R. P. Woods. Automated image registration. `http://bishopw.loni.ucla.edu/AIR5/index.html` [Last accessed 3 February 2009], 2002.

[53] Y. Zhu, D. Magee, R. Ratmalingam, and D. Kessel. A virtual ultrasound imaging system for the simulation of ultrasound-guided needle insertion procedures. *Proceedings of Medical Understanding and Analysis*, pages 61–65, 2006.