INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

High Speed 3D Ultrasound Reconstruction: A Comparative Study Between Parallel and Sequential Processors

Amol S. Karnick

July 9, 1998

Department of Electrical Engineering
Mcgill University, Montreal,
Canada

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements of the degree of
Masters Of Engineering

Abstract

The utilization of ultrasound in diagnostic medicine has increased since the 1950's. Ultrasound is a non-invasive two dimensional (2D) imaging technique that provides useful information about underlying soft tissue anatomical structures. Developed recently, three dimensional (3D) reconstruction algorithms convert a series of sequential 2D ultrasound images into 3D data sets. 3D reconstruction is one phase in a pipeline for 3D ultrasound volume visualization that typically takes upwards of 2 minutes to complete. The objective of this research focuses on high speed 3D ultrasound reconstruction by comparing reconstruction speeds on parallel and sequential processors. To decrease reconstruction times, optimizations such as; conversion from floating point to fixed point, multithreading, in-line coding, and mathematical analysis were performed. Optimizations achieved a 15%-50% performance increase. Although parallel processors are not a requirement for 3D reconstruction, they will be necessary to achieve the ultimate goal of real-time 3D ultrasound volume visualization.

Abrégé

L'utilisation de l'ultrasons dans la médicine diagnostique a augmenté depuis les années 50. L'ultrasons est une technique d'imagerie à deux dimensions (2D) non-envahissante qui fourni des informations importantes sur les structures anatomiques. Un algorithm qui convertis une série d'images d'ultrasons 2D en une structure 3D a été récemment développé. La reconstruction en 3D est une étape unique dans un pipeline pour la visualisation du volume en 3D et prends typiquement plus que 2 minutes pour compléter. L'objectif de cette recherche est la reconstruction d'images ultrasoniques en 3D et à haute-vitesse. Celà est achevé par la comparaison de la vitesse de reconstruction entre les processeurs parallèles et les processeurs sequentieles. Pour réduire le temps de reconstruction, les optimisations, par exemple, la conversion de pointe flottante à pointe fixe, le "multithreading", la programmation en-ligne, et les analyses mathématiques ont été utilisées. Les optimisations ont atteint une augmentation de performance de 15% à 50%. Bien que les processeurs parallèles ne sont pas requis pour la reconstruction en 3D, ils seront nécessaires pour obtenir une reconstruction rapide et une bonne visualisation du volume en 3D en temps réel.

Acknowledgments

Through the few years that I have been working on my degree, many people have helped me reach my goal of completing my degree. Though the time to completion has been rather lengthy, it has definitely been interesting. My degree has been a tremendous learning experience that I am glad to have endured and finished, but only with the help of my family and friends.

I would first like to thank **Dr. K. L. Watkin**, who is my thesis supervisor, but only in name. In reality, he has been someone to talk to, joke with and seek advise on numerous occasions for both in the office and out of the office problems. No words could ever do real justice to convey my thanks for everything he has done.

Souheil "1 Kill" Hakim, Sunil Vasudevan, Ibrahima Diouf and Jerri Miller are all people that I would like to thank for their time and advice in solving problems and general companionship during the few years that I was here. Specifically, I would like to thank Souheil for his countless, endless, and "useful" discussions.

I would also like to thanks **Dr. G. Gao** and his students for their help on understanding and using the Earth-Manna parallel processing system.

Of course, I have a great deal to thank **my parents and brother,** for everything from just being there to money (sorry mom and dad). Thank for being there and helping me with everything mom. Of course my relatives are also among the people I have a lot to thank for, especially **Sonali Karnick** for her help with the French abstract.

Now, though last, but definitely not least is **Alpana Das.** You have always been there for me and put up with me during the last few months, especially when I was writing the thesis. Thank you for you patience and sweetness.

Table of Contents

1. INTRODUCTION **1-1**

1.1 THE NEED FOR RAPID 3D ULTRASOUND 1-4

2. BACKGROUND **2-1**

2.1 3D ULTRASOUND 2-1
2.1.1 Multidimensional Array *2-1*
2.1.2 Mechanical Scanners *2-2*
2.1.3 3D Freehand and Localized Freehand Acquisition *2-3*
2.2 ULTRASOUND ACQUISITION SYSTEM 2-5
2.3 VOLUME RENDERING 2-8
2.4 ACCURACY OF RECONSTRUCTED 3D DATA 2-11
2.5 PREVIOUS WORK WITH 3D ULTRASOUND AND PARALLEL PROCESSORS 2-14

3. PERFORMANCE ENHANCEMENTS **3-1**

3.1 MULTITHREADING 3-1
3.2 IN-LINE CODING 3-4
3.3 MATHEMATICAL ANALYSIS 3-6
3.4 FIXED POINT CONVERSION 3-7
3.5 ADDITIONAL FEATURES 3-8
3.5.1 Imrecon Considerations *3-9*
3.6 OPERATING SYSTEMS 3-10

4. PARALLEL PROCESSING **4-1**

4.1 SHARED MEMORY MODEL 4-3
4.2 DISTRIBUTED MEMORY MODEL 4-4
4.3 EARTH-MANNA SYSTEM 4-5
4.3.1 MANNA System Specifics *4-6*
4.3.2 EARTH System Specifics *4-6*
4.4 DESIGN CONSIDERATIONS FOR PARALLEL PROCESSING 4-7
4.4.1 Data Passing *4-10*
4.4.1.1 Method 1 - Non-multithreaded Version 4-11
4.4.1.2 Method 2 - Multithreaded Version 4-11

5. RESULTS **5-1**

5.1 PERFORMANCE ENHANCEMENTS 5-2
5.1.1 Additional Features *5-4*
5.1.2 Fixed Point Version *5-5*
5.2 OPERATING SYSTEMS 5-9
5.3 PARALLEL PROCESSOR 5-10

6. DISCUSSION **6-1**

6.1 CODE IMPROVEMENT AND FIXED POINT ALGORITHM 6-2
6.2 PARALLEL PROCESSOR 6-6
6.3 EASE OF USE 6-10
6.4 CURRENT PROCESSORS 6-11

7. FUTURE WORK **7-1**

8. CONCLUSIONS **8-1**

9. REFERENCES **9-1**

List of Tables

TABLE 5-1 - PERCENTAGE INCREASE OF OPTIMIZED CODE 5-3
TABLE 5-2 - MEAN PIXEL INTENSITY 5-4
TABLE 5-3 - PERFORMANCE COMPARISON WITH IMAGE DISPLAY 5-4
TABLE 5-4 - FIXED POINT VS. FLOATING POINT EXECUTION TIMES 5-5
TABLE 5-5 - STATISTICAL ANALYSIS OF FIXED POINT VS. FLOATING POINT 5-7
TABLE 5-6 - SPEED-UP TIMES FOR MANNA EXECUTION 5-11
TABLE 5-7 - COMPLETE EXECUTION TIME AND SPEED-UP (NON-MULTITHREADING) 5-12
TABLE 5-8 - COMPLETE EXECUTION TIME AND SPEED-UP (MULTITHREADING) 5-12
TABLE 6-1 - SPEC BENCHMARK RESULTS FOR VARIOUS PROCESSORS 6-12

List of Figures

FIGURE 2-1 - 3D TRANSDUCER, SWEEPING VOLUME [] 2-3
FIGURE 2-2 - PIPELINE OF 3D PROCESSING SYSTEM 2-7
FIGURE 2-3 - FLOW CHART OF DATA ACQUISITION 2-8
FIGURE 2-4 - SURFACE RENDERING 2-10
FIGURE 2-5 - OPACITY LEVEL RENDERING 2-10
FIGURE 4-1 - SHARED MEMORY ARCHITECTURE 4-4
FIGURE 4-2 - DISTRIBUTED MEMORY ARCHITECTURE 4-5
FIGURE 4-3 - PSEUDOCODE FOR IMRECON ROUTINE 4-8
FIGURE 4-4 - DESCRIPTIVE PICTURE OF PSEUDOCODE 4-9
FIGURE 4-5 - OVERVIEW OF REBUILDING IMAGE - NON-MULTITHREADED VERSION 4-11
FIGURE 4-6 - GRAPHICAL DESCRIPTION OF MULTITHREADED ALGORITHM 4-12
FIGURE 5-1 - EXAMPLE IMAGE FROM ORIGINAL CODE 5-3
FIGURE 5-2 - EXAMPLE IMAGE FROM OPTIMIZED CODE 5-3
FIGURE 5-3 - HISTOGRAM FOR ORIGINAL CODE 5-3
FIGURE 5-4 - HISTOGRAM FOR OPTIMIZED CODE 5-4
FIGURE 5-5 - FLOATING POINT IMAGE 5-6
FIGURE 5-6 - FIXED POINT IMAGE 5-6
FIGURE 5-7 - HISTOGRAM OF FLOATING POINT IMAGE 5-7
FIGURE 5-8 - HISTOGRAM OF FIXED POINT IMAGE 5-7
FIGURE 5-9 - CORRELATION COEFFICIENTS BETWEEN FIXED POINT AND FLOATING POINT IMAGES 5-8
FIGURE 5-10 - RENDERED FLOATING POINT VERSION 5-9
FIGURE 5-11 - RENDERED FIXED POINT VERSION 5-9
FIGURE 5-12 - DIFFERENCE IMAGE BETWEEN FLOATING POINT AND FIXED POINT RENDERED IMAGES 5-9
FIGURE 5-13 - CONTRAST ENHANCED DIFFERENCE IMAGE 5-9
FIGURE 5-14 - SPEED-UP CURVE OF IMRECON ROUTINE 5-11
FIGURE 5-15 - SPEED-UP CURVE OF TOTAL EXECUTION TIME 5-12
FIGURE 6-1 - SPEC PERFORMANCE FOR PROCESSORS 6-13

1. Introduction

Rush, speed, now! These words signify that something is required immediately. Whether that something is important enough to justify the extra work required to meet the deadline depends on one's point of view. For example, in medical imaging during surgery, these words are of utmost importance. A surgeon requires feedback while the patient is in the operating room and not after surgery. Most individuals are reluctant to have invasive exploratory surgery. In fact, the medical profession is heading toward minimally invasive surgery.

Forestalling the vast use of imaging techniques in the operating room are their slow display and reconstruction times. Two dimensional (2D) image modalities, such as Ultrasound, Magnetic Resonance Imaging (MRI), Computed Tomography (CT), X-Rays, and Positron Emission Tomography (PET), produce images that allow accurate, qualitative and quantitative estimation of structure and function [1], but nevertheless are 2D, and therefore limiting. 3D data sets provide a more intuitive picture which has the potential to help surgeons and doctors make a better diagnosis. This leads to the requirement of high speed reconstruction algorithms that essentially convert 2D images to 3D data space. Unfortunately, many imaging modalities create large data sets that are computationally intense and require longer times to process than permitted in medical settings. Long delays are intolerable and offset benefits gained by 3D imaging.

X-rays and thereby CTs are not capable of soft tissue imaging. PETs are not capable of physiological imaging, PET scans are designed to detect chemical reactions. Although the resulting images are excellent for diagnostic purposes, imaging soft tissue with MRI scans are slow. Ultrasound, on the other hand, provides a safe repeatable and fast method for imaging soft tissue.

MRI, CT and PET scans are excellent imaging techniques, but are limited in surgical applications, due to the physical size of the apparatus or long acquisition times. 3D reconstruction for MRI, CT, and PET have an advantage over ultrasound, because they have pre-determined movement of their respective scan heads, that can be easily recorded to the accompanying computer during image acquisition. The acquisition method provides a means to easily interpret image locations in 3D space and reconstruct the images. Ultrasound images are gathered by freely orienting the scan head to obtain "optimal" images, but adds complexity to 3D reconstruction. To simplify 3D ultrasound reconstruction, some researchers have used mechanical means to acquire ultrasound data [2,3,4], but these techniques are limited, sometimes cumbersome and require additional hardware or devices.

The need for real-time 3D ultrasound is a vital next step for ultrasound imaging. 3D ultrasound provides improved visualization of ultrasound data. 3D reconstruction facilitates the study of structures or viewing planes, not attainable through conventional 2D ultrasound techniques. To create real-time 3D

ultrasound images using a scanhead localization technique requires simultaneous image acquisition.

The actual visualization of the 3D ultrasound images requires image rendering. Rendering is the process of shading or coloring objects in the 3D space projected into a 2D viewing space. Rendering is needed since the computer screen is 2D. Two types of rendering are possible, the first is surface rendering where only the surfaces of the objects are given color or opacity. The second is internal rendering or volume rendering, where the contents of the object are visible. The method used depends on the characteristics of the data set, as well as the objective for rendering the image.

There are many researchers that have developed three dimensional ultrasound imaging methods. Among them are Watkin *et. al.* [5], Pini *et. al.* [1], Ohbuchi *et. al.* [2], Shapiro *et. al.* [6] and Capineri *et al.*[27]. Various techniques are employed to create the 3D data, such as 3D transducers [7], 2D transducers controlled by a motor [8], and localized [5] or non-localized 2D[11] freehand. The method used by the reconstruction algorithm discussed in this thesis is the localization method.

The focus of this thesis is to determine whether computational power, such as a parallel processor, is required to achieve real-time reconstruction. Newer sequential processors may be efficient and computationally fast enough to provide

reconstructed data in real-time. Speed is the most important issue for 3D reconstruction. This proposed question does not focus on the use of parallel processors in general, but is aimed more precisely at the problem at hand, 3D ultrasound image reconstruction.

The research conducted for this Master's utilizes an algorithm created by Watkin *et. al.* [5]. The research involved several steps. First, the original code was optimized to provide more efficient execution. Secondly, a conversion from the original floating point to fixed point algorithm was completed. The results were compared for loss of image quality as well as precision. The original algorithm was also modified and compiled to run efficiently on a parallel processor. Comparison of multithreaded code to non-multithreaded code was also completed.

The organization of the thesis is as follows. Section 2, 3, and 4 provide background information on previous work with 3D ultrasound, performance enhancements and parallel processors. Section 5 focuses on the results obtained, and section 6 discusses the results and their implication on the research conducted for this thesis. Section 7 highlights future work and interesting applications of 3D ultrasound. Section 8 summarizes the finds of the research.

1.1 The Need for Rapid 3D Ultrasound

The view of an object from any direction in diagnostic medicine is very important. Certain viewing angles provide information that may not be visible from only one

viewing point, the disadvantage of the conventional 2D ultrasonic image [6]. The ultimate goal for 3D ultrasound research is to provide a real-time, interactive 3D display. The limiting factor is technology and not the imagination. 3D ultrasound has the potential to provide:

- *Improved Visualization*: The present state of 2D ultrasound requires considerable training and experience to provide proper diagnosis. 3D ultrasound can help overcome this limitation by displaying the information more akin to the human visualization system, that is in 3D, and more accurately display anatomical structures.
- *Volume Measurement*: 3D ultrasound will aid in the volume estimations and distance measurements, and surpass those made with 2D ultrasound.
- *Reduced Examination Time*: The ability to wave the transducer over the desired volume in one pass, and not have to build the 3D anatomical structure in his or her mind, the sonographer's acquisition time would be considerably less. Reduced examination time will reduce the patients' stress, as well as the sonographer's, since there would be less chance of errors.
- *Analysis of Volume of Interest*: The volume, once accurately obtained, contains all necessary information for further diagnosis, and can be segmented for desired areas. Surface and Volume rendering will provide means to view the data in a more pleasing manner, as compared to a 2D ultrasound. This is analogous to poor antenna T.V. reception.

2. Background

2.1 3D Ultrasound

Currently, there are four methods employed to create 3D ultrasound (3DUS) images. The four methods for 3D ultrasound are: 1. Multidimensional arrays, 2. mechanical scanners, 3. freehand and 4. localized freehand. The first method uses a 3D ultrasound transducer (or multidimensional array), while the remaining methods use sequential acquisition of 2D ultrasound images. The three methods for sequential image acquisition are motorized mechanical scanhead, localized freehand, and non-localized freehand. Localized freehand is the use of a 3D tracker/localizer that registers the location of the transducer in 3space.

2.1.1 Multidimensional Array

Multidimensional arrays use a M X N transducer array arranged as a matrix, opposed to the conventional 1 X N array (2D ultrasound). Von Ramm *et. al.* [7] have developed a research device which is not available commercially. Many issues remain to be dealt with, such as beam steering and focusing as well as construction of the transducer [7]. These types are devices are very new, and are the only true 3D ultrasound available. The 3D transducer projects and displays a volume rather than a plane as compared to a conventional 2D transducer. General Electric (GE) Medical Systems has recently introduced a quasi 3D transducer with 1024 transducer elements (or crystal elements). They use all the 1024 (arranged in

four rows of 256) for the echo transmit, using only one row to receive the echo. It is not a true 3D probe, but a step in the right direction. The transducer design provides better control for beam steering and focusing in three dimensions.

2.1.2 Mechanical Scanners

Mechanical scanners use a motorized transducer head. Essentially, the 2D transducer is moved in either an angular or planar motion as depicted in Figure 2-1. This provides known fixed locations for the transducer head which simplifies 3D reconstruction. Since, the location is known prior to the image acquisition image data can be easily transformed to voxel space. Acuson and Kretz-Technik[8] are two commercial ultrasound companies that use this method of acquisition.

The advantage to the mechanical scanner system is that the reconstruction is simpler and more accurate. The accuracy claim is based on the inherit advantage that the localization of the transducer head is known at any given moment in time. Reconstruction is accomplished using lookup tables. Thus, reconstruction time is rapid (approximately 5-10 seconds) [9]. This is faster than the 2D freehand acquisitions of current systems.

The main disadvantages of the motorized transducer system are fixed speed of movement of the motor head and interpolation is used to fill the gaps of the an angular movement of the scanhead. A servo or stepper motor controls the

movement of the transducer head, intervals between acquired images may be too large and miss crucial information.

Figure 2-1 - 3D Transducer, Sweeping Volume [10]

2.1.3 3D Freehand and Localized Freehand Acquisition

To create an accurate 3D reconstruction, the relative location of parallel scan planes must be known. Without such knowledge 3D reconstruction would be nearly impossible. However, if the person performing a scan moves the transducer very slowly over a short distance and maintains each scan plane in relatively the same plane, then a sequential "3D data" set has been created without the use of a motor or any localization device. Two companies have included the first method within their products, ATL [11]and GE. Both machines work in the method described above. Although reconstruction of 3D ultrasound using this technique is possible, there are several disadvantages. Slight deviations during scanning distorts the image and thereby introduce artifacts which may alter the medical diagnosis. In addition, the recorded images are uncalibrated in 3space, which does not allow for volume estimation or distance calculations on the 3D images. The non-localized 3D ultrasound images produce approximately 100-200 images per scanned region, which may not necessarily contain enough information to produce

complete images. The drawbacks to this method suggest that the use of localization devices (trackers) in conjunctions with the freehand scanning method would solve both the image distortion problem and create calibrated 3D images.

To freely move in 3space and register each movement, six variables must be recorded. There are three Cartesian or translation movements (x, y, z), and three angular or rotational movements (θ, β, α), resulting in 6 Degrees of Freedom (DOF). The various degrees of freedom make reconstruction difficult. To have accurate reconstruction, all six DOFs must be available, such would be the case in an ideal world. Several methods to capture six DOF are magnetic trackers, optical trackers and acoustic trackers. The first use of capturing six DOF was accomplished using an acoustic spark gap triangulation system [12]. By employing such a system, the researchers could prepare three dimensional wireframe representations of the head and trunk of an abortus as well as measure volume and weight [5].

The most popular tracking method is the magnetic tracker because it is a relatively simple device using two units; a transmitter and a receiver. The transmitter is in a fixed position, whereas the receiver is positioned on the ultrasound probe, such as the system in use by Watkin *et. al.* [5]. The use of magnetic trackers has greatly increased the potential of freehand 3D reconstruction, but the trackers have limitations. The main problem with the magnetic tracker is the lack of precision when movement is a backward motion (relative to captured forward motion).

Measurement error is also increased proportionally to the distance of separation between the transceiver and receiver [21] and nearby magnetic fields produce erroneous readings [20]. These problems actually limit the movement. Only forward motion is permitted, otherwise the tracker data is unreliable.

There are a number of advantages to using localized freehand 3D acquisition. Multiple passes over the strong echogenic regions can fill in gaps within the image, in addition scans over the same region can reduce the noise, since the final image will be an average of several images [13]. An important advantage is that the system can be added to any existing ultrasound machine, by simply purchasing a workstation and tracker (e.g. Polhemus). Rohling [13] has provided a good source for an in-depth look at 3D ultrasound freehand, such as sources of errors, imaging problems.

2.2 Ultrasound Acquisition System

The algorithm tested in this thesis is based on the work conducted by Watkin *et. al.* [5]. The research revolves around a 3D ultrasound reconstruction program (Imrecon). This program produces a 3D ultrasound image from the input of a sequential series of 2D ultrasound images and a file containing the 3-space location of each image. The 3D image reconstruction is one step within a processing pipeline. The 6 stages are illustrated in Figure 2-2 and expanded upon below.

The ultrasound images and localized data are acquired during the image acquisition phase. A sub window is selected from the full NTSC video ultrasound stream. Only this sub-frame is saved and reconstructed further down the pipeline. Proceeding to the next step is optional. This step involves preprocessing the acquired data to enhance the appearance and to highlight selected regions. The images can then be viewed to observe the quality of the data. The next step is to reconstruct the 2D data set to the 3D data set, which is the focus of this research. Subsequent to the reconstruction step is volume rendering, where the image is displayed as a 3D object. To enhance the appearance of the 3D image planes (or slices) further processing can be applied. One example would be to isolate for blood vessels, to show the flow or delivery of blood to an anatomical object.

Localized 3D ultrasound methods captures 300-1000 images creating densely sampled data sets. The large number of images is computationally taxing, slowing down time to output. Fast processing is required to compete and surpass other 3D methods.

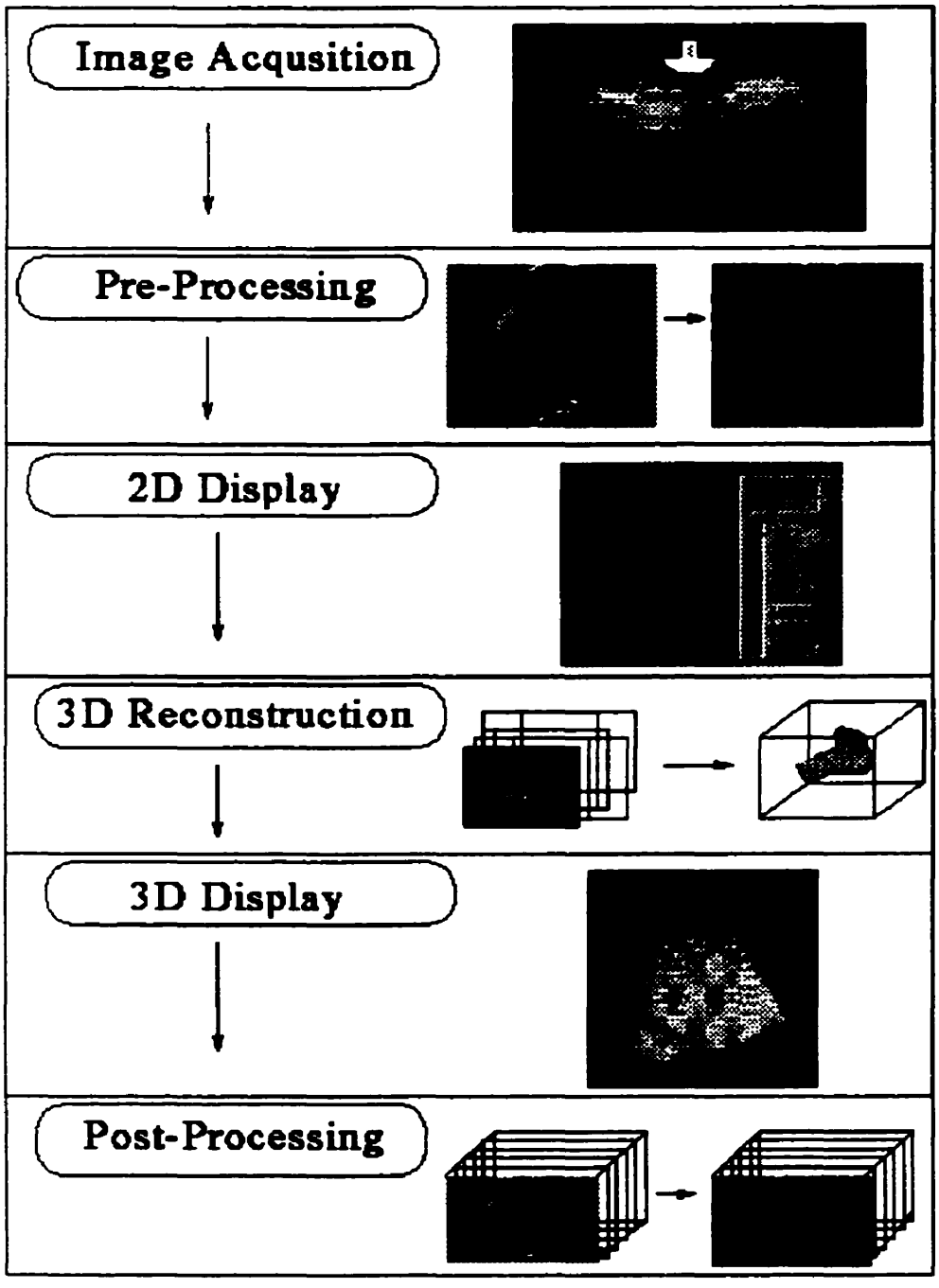

Figure 2-2 - Pipeline of 3D Processing System

Figure 2-3 depicts the flow of data within the capture program used by Watkin *et. al.* [5]. The image file and ASCII file are used as inputs for the 3D reconstruction

algorithm. The image capture software was written in C, and currently runs on a SGI R3000 computer.

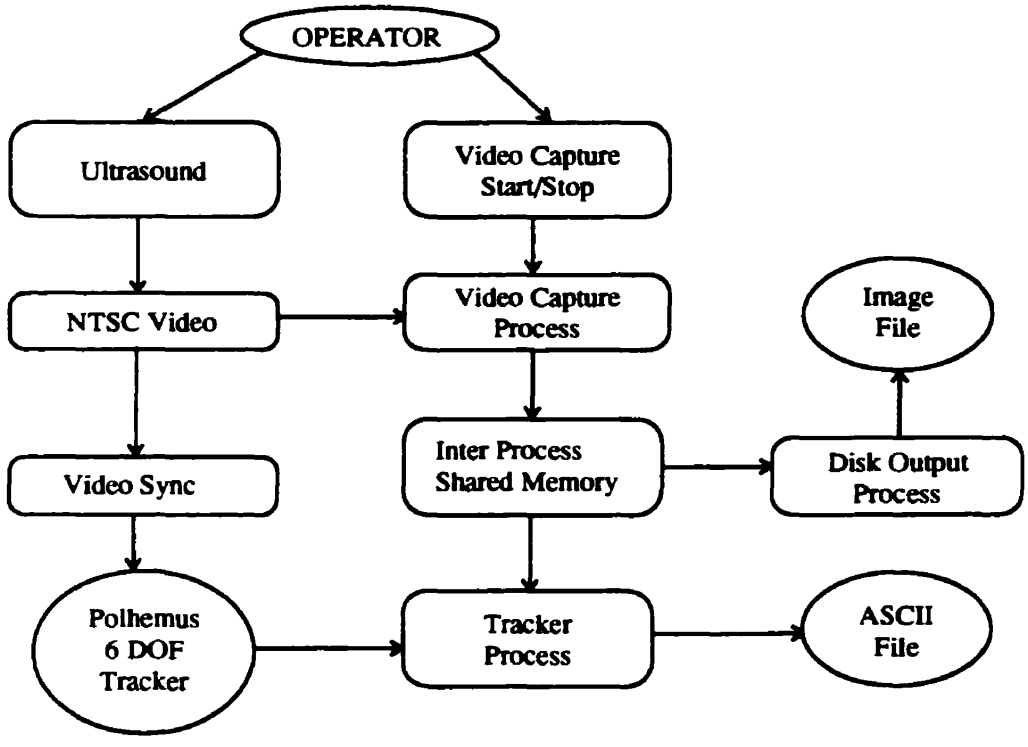

Figure 2-3 - Flow Chart of Data Acquisition

2.3 Volume Rendering

Although, the focus of this research is prior to the rendering in the pipeline, volume rendering is a very important issue, since it is the only way to visualize the final data set in 3D rather than 2D slices. The 2D slices may be displayed along any axis and provide substantial information, but the 3D viewing is only accomplished by volume rendering the 3D data set. "*Toy Story*" (the first animated

movie based solely on compute graphics) is an example of computer graphics and volume rendering. The characters and imagery have all been volume rendered, or more specifically surface rendered, which is discussed in detail below.

Rendering, in the most simplistic view, is the process of giving realism to objects in 3D space. 3D models or data sets, are represented in three dimensions, but only in the memory of the computer and need to be viewed on the computer's monitor, which is 2D. Rendering accomplishes the task of transforming the 3D data set to be viewed on the display system. Many techniques are available to render an image and are generally classified as either image-order and object order. Image order rendering techniques focus on the image as a whole, whereas object order techniques are primarily concerned with the objects or individual models with in an image. For a comprehensive review of rendering, please see Foley *et. al.* [14].

Volume rendering shows the characteristics of an enclosed volume. Each point in the 3D data space has a value associated with it to indicate the intensity of that space or more precisely termed *voxel.* The *voxel* is the 3D equivalent of the pixel (*pi*cture *el*ement) in a 2D image. Medical images, such as CTs, and MRIs extensively use volume rendering to produce a viewable 3D image. Volume rendering methods can be divided into two distinct categories: surface rendering and opacity rendering [14]. Surface rendering only renders the surfaces of the objects within the volume, which hides the details within the object. The other method gives the voxels an opacity level, allowing the inside of the volume to be

viewed. For medical imaging both methods are important depending on the situation, some instances require viewing the internal structures, and other times the surface of the object is of interest. For example, when diagnosing a liver using a CT scan, the doctor would be more interested in viewing the internal structure of the liver rather than the surface. Figure 2-4 and Figure 2-5 (Please note that the image quality of the printed figure, does not match that of the display system) are examples of the two volume rendering methods, respectively. As seen from the figures, the surface rendering (Figure 2-4) is more understandable and more appealing to the eye and casual observer. The problem is that no useful information can be obtained of the internal structures, whereas Figure 2-5 shows details of the internal structure, which is useful for diagnosis. Surface rendering is useful for finer details of surfaces, which may not be visible with the opacity method.

Figure 2-4 - Surface Rendering

Figure 2-5 - Opacity Level Rendering

Time is very important in the visualization process. Volume rendering slows down the total time to display. Fast Volume rendering as applied to medical imaging is the focus of many researchers today [15, 16, 17]. Vast computer systems and special Integrated Circuits (IC) [18] have been developed for the sole purpose of volume rendering. Volume rendering is an extensive topic, and further details are available from the references within this section.

2.4 Accuracy of Reconstructed 3D Data

As the importance of 3D ultrasound imaging grows, the need to ensure that the data is accurate is just as important. Researchers have conducted studies to validate the use of 3D ultrasound either through tracking algorithm reconstruction [21], motorized 3D ultrasound probe [9, 20, 22], or simulating a 2D array ultrasound probe [19].

Work conducted by Pretorius *et al.* [9] is a study on the accuracy of using 3D ultrasound reconstruction versus that of a regular 2D ultrasound scan. The results of the study are positive, indicating that the use of 3D ultrasound is diagnostically helpful. This particular study investigated using 3D ultrasound to diagnose abnormal lips in fetuses. Using 3D images, there were able to confirm 92% of the cases had normal lips compared to 76% using 2D ultrasound images.

3D ultrasound is an effective tool for volume measurement. Verifying the accuracy of volume measurements is required to justify the use of 3D ultrasound. Riccabona *et al.* [20] have conducted tests to measure the accuracy of distance and volume measure measurements using 3D ultrasound. All tests were performed *in-vitro* on balloons, whose volumes were known. The results for the distance measurements are not compared to 2D ultrasound, but are compared for accuracy once the data is reconstructed and rotated versus the data reconstructed and in line with the acquisition plane. The mean measurement error for in-line acquisitions was 1.9 ± 1.8 %, and the mean error for a rotated data set was 6.1 ± 4.7 %, these results were for measurements directly on the phantom surface. For measurements immersed in a water bath, the results were 1.0 ± 0.8 % and 1.0 ± 0.6 % for in-line and rotated data sets respectively. Improved results were a direct consequence of pressures applied to the phantom surface during the acquisition phase.

For volume measurements, conventional 2D ultrasound had a mean error of 12.6 ± 8.7 % and 3D ultrasound had an error of 6.4 ± 4.4 % on ellipse shaped balloons. For irregular shaped balloons, the mean error for 2D ultrasound was 17.3 ± 11.2 % and 3D ultrasound had a mean error of 7.1 ± 4.6 %. The work conducted by Riccabona *et al.* [20] concludes that the use of 3D ultrasound data does improve the volume estimates over 2D ultrasound, which in turn aids physicians in providing more accurate diagnosis.

A study by Hughes *et. al.* [21] addresses volume estimation of 3D reconstructed ultrasound data on a test phantom (balloon), human liver and kidneys. The results are promising though they mention further *in-vivo* tests are required. The balloon, liver and kidneys' volumes were measured using other methods, such as water displacement in a tank for the volumes of the liver and kidney. The mean errors were 0.9 ± 2.4% for balloons, 2.7 ± 2.3 % for kidneys, and 6.6 ± 5.4 % for livers. Errors were attributed to the size and quality of captured images, the scanning technique (e.g., orientation and number of slices), errors in the Fastrak (Polehemus), data (the magnetic tracker), and errors in the volume estimation algorithms.

Tong *et al* [22] have attempted to validate the use of 3D in clinical diagnosis of prostate cancer. As correctly stipulated by Tong, the acquisition of the 3D image is faster than 2D image, though reconstruction times do add to the overall time. It can take up to 5-10 minutes for a technician to build the 3D image mentally by successive scans over the same region. The 3D motorized probe can scan the region of interest in 2 minutes. Prostate imaging requires a rectal exam. The volume or size can help accurately stage the development of prostate tumors. *In-vitro* and *In-vivo* tests on balloons and prostates have an accuracy of ±1% for both volume and distance. The 3D images clearly show the tumor on test subjects that were known to have a tumor. The benefit of the 3D is extensive in aiding diagnosis. In addition, the images can be stored digitally thus can be accurately retrieved at a further date for re-evaluation by a colleague.

3D ultrasound reduces human error by taking the imagination factor away from the technician, as well as expediting the scan time. The overall effect increases income potential for prospectus medical community by allowing more patients to be seen in the same amount of time, since a 3D scan only takes a couple of minutes verses the 10-15 minutes by conventional 2D ultrasound [22]. The studies provide a good basis to validate that 3D reconstruction for ultrasound is accurate and is a useful tool within the medical diagnostic community.

2.5 Previous Work with 3D Ultrasound and Parallel Processors

The application of parallel processing techniques for the display of ultrasound images was first reported in the late 1970's and early 1980's. Several researchers have used parallel processors for linear sequential arrays [23,24,25]. The use of parallel processors for B-mode ultrasound images (the image produced by the ultrasound linear array) increased acquisition or frame rates. Delannoy's system [23] had a frame rate of 1000 frames per second with 70 lines per frame. The higher frame rate allows for images to be produced independently and processed prior to display. Furthermore, the images can be averaged over a range of images and displayed at 30 frames per second, thereby reducing noise and providing a clearer picture. Another advantage was that larger areas could be scanned (with the linear array, meaning that larger arrays could be used) without sacrificing frame rates.

The Fraunhofer Institute for Computer Graphics (IGD), a research consortium in Germany, employs parallel processors in their 3D reconstruction ultrasound machine [28]. They have developed a 4 node parallel processor (expandable to 128 nodes). The reconstruction times have not been published primarily because the system uses a 3D motorized transducer. Essentially the system does not need to do extensive reconstruction on the acquired data set, since the motorized 3D transducer is simple and efficient to reconstruct. The volume rendering is done in 0.2-0.3 seconds on a 10 MB file. This data was presented by the author with no other comparison to other work [28]. Furthermore, they have demonstrated approximately linear speed-up for volume rendering with 3 nodes (2.8 speed-up), though more nodes were mentioned (up to 8 nodes), no quantitative measures were reported. They have termed their technology as "interactive visualization", since it is not real-time, but very efficient.

Another group, Capineri *et. al.* [27], have designed external processing hardware using 3 parallel Digital Signal Processors (DSP), the TMS329C50 at 56 MHz. The medical focus is on echocardiology. Their system uses a mechanically controlled probe mounted on a central rotating axis. The system in synchronized with the cardiac cycle so that scans are taken at the same time during each cardiac cycle. This allows the 3D reconstructed image to be relatively stationary, otherwise the images would not be synchronized with respect to each other, leading to a poor reconstruction. A commercially available 2D ultrasound machine has been modified to control the probes' rotation and to acquire the echographic

sections. Additionally, it has been enhanced by a processing board with the three DSPs. The DSP board is connected via a bus to the host ultrasound. The data for reconstruction is piped to the DSP board. The reconstructed data is written directly to video memory. There is a lag of 60-120 seconds prior to visualization at a rate of 1 frame per second.

The use of parallel processors is even more significant when using a true 3D transducer as developed by von Ramm *et al.* [7], see section 2.1.1. In order to process the large amount of data generated by the probe, a parallel processor was employed. They used the Explososcan [25], a parallel processor initially developed to increase the frame rate of linear arrays. They have expanded the system to handle the multiple arrays of the 2D (20x20 element) array transducer. Since the 3D transducer has multiple arrays that generate data simultaneously, the parallel processor can do varies calculations on the data in parallel for each independent array. The data is presented in nearly real-time rates of 8 frames per second as on-line projection images with depth perspective, stereoscopic pairs, or multiple tomographic images. To date, due to production problems, the 3D transducer is not commercially available.

State *et al.* [26] uses a Pixel-Planes 5 parallel processor for reconstruction and volume render the 3D data set. Their system is slightly different in the sense that they superimpose the reconstructed data set over a video sequence of the observer's view. The overlay is performed off-line, and not in real-time. The

acquisition of the ultrasound and observer video is captured in real-time, but the reconstruction and volume rendering are performed off-line to improve reconstruction times. They use the parallel processor to reconstruct and render the images that are overlaid on the video. The limitation of the system is that the work is performed off-line and cannot be considered real-time. No processing time was reported by State.

Researchers are continuing to use parallel processors in an attempt to achieve real-time 3D ultrasound [7,27,28]. To date, even with parallel processors, none have accomplished this very demanding task. Researchers such as von Ramm have attempted to use 2D arrays, as well as parallel processors, but the task of real-time 3DUS is yet unachieved. The obstacle many researchers have faced is the amount of data that must be processed to obtain and display the 3D images. Both tasks are very demanding on computer resources, and efforts need to be made to allow for faster execution and reconstruction.

Parallel processors have played a pivotal role in advancing ultrasound and decreasing imaging times on various systems. Most of the work conducted on parallel processors is based on older technology. New sequential processors have become prominent, efficient and powerful. These sequential processors are 50 times faster than technology from the mid-80's. One important question not answered in research literature is whether sequential processors can perform the 3D reconstruction phase of the visualization pipeline as rapidly as parallel

processors. The answer to this question has important implications for the development and design of ultrasonic machinery. The purpose of this thesis is to compare the 3D ultrasound reconstruction times on new sequential processors to that of the parallel processors.

Presented in this section was research focused at improving execution speed of 3D reconstruction and visualization. Fast processors alone may not necessary be enough to achieve high speed 3DUS, performance enhancements are needed that will allow faster reconstruction and display. These enhancements include, multithreading, code optimization, and more efficient use of parallel processors, all of which will be discussed in the proceeding sections, followed by the results and discussion.

3. Performance Enhancements

Numerous methods exist to improve computer algorithms' performance to accelerate image processing techniques. The goal of this thesis is to attain near-real-time or real-time 3D reconstruction. To achieve this goal, the reconstruction algorithm needs to be optimized. An optimized algorithm can easily increase the speed of execution by a minimum of 10% [36]. Multithreading, in-line coding, mathematical analysis and fixed point conversions are among the discussed topics to improve the 3D reconstruction algorithm.

3.1 Multithreading

One of the latest advances in programming is the use of multithreading. The concept is not new, in fact experimental systems have been around since the 1950's. More recently, with the advent of superscalar processors, and operating systems that are designed to support multitasking, multithreading has the potential for increased use.

Multithreading provides a means to hide or utilize long latencies. During these latency periods the processor is waiting for the data to either be written or read from an external storage device (i.e. memory, hard-drive). These dead times are useless to the processor, and slow down execution times. Multithreaded architectures utilize the latency by creating multiple threads that can be executed concurrently. These threads are interleaved on a single processor. For example, if one thread is requesting data, another thread can use the CPU for execution or

calculations. Thus, the dead times are constructively used [29]. A multithreaded architecture hides long latency operations by task switching, or, more appropriately, context switching. A multithreaded program will run more efficiently than a single threaded application [29] on a system designed for multithreading.

There are essentially two facets to multithreading, hardware and software. Hardware multithreading has multiple registers to hold multiple contexts, thus reducing context switching overhead. Though the number of contexts is limited by the cost of hardware, it is also limited by 4 other issues. The four factors are: 1. The number of contexts supported by the hardware. 2. The cost of switching between cycles. 3. Run length, the number of cycles typically executed between switches. 4. The characteristic latency of the operations that are to be hidden [29]. Byrd *et al.* [29] suggest that 11 contexts seems to be the right balance, since the processor efficiency remains at 90% beyond 11 contexts.

The other option is software multithreading. Software multithreading is relatively inefficient, since contexts must be saved and accessed using standard memory, which in general, is slower than processor registers. Software multithreading requires an increased number of cycles to perform the context switching, and only achieves beneficial efficiency when switching is infrequent. Frequent context switching will degrade the performance of the program compared to a non-multithreaded version.

Certain limitations must be considered when multithreading programming code on a uniprocessor. Most uniprocessors are not designed for multithreaded code, which places the burden of multithreading on the software. Software multithreading adds overhead to any multithreaded code. Also, one must keep in mind that uniprocessors are exactly as the name implies, a single processor. Is there any dynamic improvement with multithreaded code on a uniprocessor? The improvement depends on the application and the computations being executed. There are essentially two types of computations: CPU-bound and I/O bound [30].

CPU bound computations include those that perform mathematical calculations on data in memory, such as matrix related computations. I/O bound computations include those that require an external source for information, such as a hard drive, network, etc. I/O bound calculations are generally handled asynchronously, either by a dedicated I/O processor or efficient interrupt handles. I/O requests suspend the calling thread until the I/O is completed. An I/O wait can be a very long delay when compared to a local memory fetch. This causes the processor to stall (idle) while waiting for the I/O request to be completed.

Most applications contain a mixture of both I/O and CPU bound calculations. This combination allows for threads to be written to allow for maximum usage of the computer processing system. For example, a processor requests data from the external storage, this requires execution time, in the meantime, the processor can

do memory calculation to set up for the incoming data which makes efficient use of the CPU time. But if the next thread requires the information for the I/O request, there will be no performance increase.

In Asche's article [30], he addressed the use of multithreading on a Windows© 95 and Windows© NT machine. Both versions of Windows© were run on uniprocessor machines, this fact is important, since Windows© NT can be run on multiprocessor platform. Both Windows versions use software multithreading, so any concerns are limited to the systems used. Asche concluded that if calculations are CPU bound, no observable benefit can be realized using multithreads, but if tasks are I/O bound, multithreading shows good improvements.

3.2 In-Line Coding

In-line coding uses direct code within a function rather than making a call to another function. Though, this may not necessarily contribute a vast saving, it can make a difference. The minor disadvantage of using in-line coding is primarily cosmetic. The code tends to be less modular. The use of function calls makes the code more readable, but that should not necessarily be a concern when trying to achieve high-speed image processing.

When a new function is called, the processor must save the previous context of the system, in order to preserve data values used by the calling function. If this was not done, then upon return from the called function, the data values would be

incorrect. This adds overhead (similar to the context switching of multithreading, see Section 3.1), which can reduce system performance. In the grand scheme of the operations performed, the added time is very minute. For example, if we compare a program that takes 10 cycles to save context, with another that takes 10 cycles to restore the context, an additional 20 cycles are added to the program execution. If this was done 5 times within the function, that is an additional 100 cycles. Keep in mind, that a hundred cycles is not very much in computational time on a processor. A 200 MHz Intel Pentium© processor requires only 5 nanoseconds to perform one cycle, amounting to only 500 ns. additional overhead. 500 ns seems infinitesimal, but now if this needs to be done for each pixel in an image, (for example 256(w)x256(h) = 65536), this amounts to about 32 ms. This still is not a lot of time, but when this has to be done for 200 images, the added time is 6.55 seconds. As described above, a small number of cycles for one iteration can increase the total computational time of the program. Thus, unnecessary function calls should be eliminated to achieve better performance.

In Imrecon, the program developed by Watkin *et. al.*[5] to perform ultrasound 3D reconstruction, inline coding was implemented iwhere function calls where used to perform simple tasks. An example of such tasks include multiplication of vectors using function calls, in-line coding replaced the function call and performed the multiplication with the programming code body.

3.3 Mathematical Analysis

Mathematical analysis involves in-depth consideration of the program code. Mathematical calculations performed within the program code are evaluated in further detail to determine whether the calculations are performed efficiently. For example, matrix calculations can be reduced if zeros are perpetually calculated. Matrix multiplication involves multiple loops to achieve a solution. If there are a constant number of zeros within the multiplication, the number of loops may be reduced. Additionally, various variables may not change from one iteration of a loop to the next, thus it is unnecessary to re-compute these values for each iteration. If these variables can be identified, then the values can be computed prior to the start of the loop, thus reducing the number computations during each loop.

In Imrecon, the program developed by Watkin *et. al.*[5] to perform ultrasound 3D reconstruction, there are multiple locations within the code where this type of consideration is useful. There are numerous matrix calculations which have a multiplication by a vector consisting of only a 1 and two zeros. An example is given in Equation 3-1. By working out a simplification for the multiplication, numerous computations can be saved. The $M_3 \cdot p$ multiplication can be reduced to just a vector assignment of the second row of M_3. Thus, one matrix multiplication has been saved. This type of problem occurs numerous times in the Imrecon program code.

$$p = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$Out = M_1 \cdot M_2 \cdot M_3 \cdot p$$

where M_1, M_2, M_3 are 3x3 matrices

Equation 3-1

3.4 Fixed Point Conversion

Image processing has traditionally utilized floating point variables to attain accuracy, but at the cost of reduction in performance. Performance could be increased by converting the code to fixed point from floating point. On a general CPU, fixed point calculations are more efficient than floating point calculations. Fixed point numbers are interpreted as integers and are handled appropriately such that all fixed point calculations use the integer unit. Most processors have a floating point unit and an integer unit (normally part of CPU). Integer operations are faster, since the number of required bits are known before the calculations are done, although precision is reduced.

Imrecon routines use mainly floating point calculations. To improve performance, the values will be converted to fixed point, prior to multiplication or divisions. However, precision may be reduced when using fixed point calculations. This may not be a major issue considering the data available to the computer from the tracker has only 2 significant digits after the decimal. The image pixel values (the true image data) are stored as integer values, therefore there should be no loss of

image quality by converting to fixed point, due to fact that the original data has only 2 significant digits after the decimal point.

Statistical methods will be employed to evaluate the correctness of the generated image after the implementation of the fixed point algorithm. A standard histogram analysis will be used to compare the images, one generated from the floating point version to that of the integer version. Identical histograms for any given image, will show that the rounding off by the floating point conversion does not affect the distribution of the pixel intensities therefore, the reconstructed images are indistinguishable. Also, a cross correlation between the fixed point algorithm and floating point algorithm will show changes in pixel intensity as a function of location.

3.5 Additional Features

In addition to the performance enhancements, a supplementary feature will be added that displays the reconstructed images as they are formed. The additional feature will enable the physician performing the scan to view the images as they are reconstructed. Once the reconstruction is complete, the images will remain in memory for further viewing. The image display will require more resources thus will most likely slow down the reconstruction time. As long as the extra time is not excessive, the benefit will outweigh the cost. Currently, the reconstruction program saves the images to the hard drive, which must be re-loaded prior to viewing the image slices. This adds unnecessary steps and time to the viewing

process. By combining the process, time is saved in the overall reconstruction and viewing.

3.5.1 Imrecon Considerations

One consideration for this feature to work correctly is that the images must be viewable as they are being reconstructed. The question here is: Can the images be viewed as they are being reconstructed with the current algorithm? The answer to this question lies in the kernel of the reconstruction algorithm.

Profiling the algorithm determines the amount of time spent per routine or function. For Imrecon, the function that is computationally intense is the function called Imrecon. This function performs the actually reconstruction by matrix calculations to determine the correct location for each *voxel*. The information provided here is just an overview of the algorithm, since a more detailed description is given in section 4.4.

Close inspection of the Imrecon code reveals that the outer loop is independent of each successive iteration, thus the loops can be viewed independently. Though the reconstructed images are independently reconstructed, the *voxel* space may not be fully viewable. The algorithm does reduce the number of images after reconstruction. Approximately, 4-5 image slices are used to construct 1 image plane in the *voxel* space. Thus, if the reconstructed image is shown immediately after processing a single image, the computer will encounter a referencing problem

when the displayed plane is greater than the number of image planes for the *voxel* space. A solution would be to display every 5th reconstructed image from the original image slices.

3.6 Operating Systems

The original Imrecon was executed on a SGI (R3000 33Mhz), using the UNIX operating system (O.S.). For this research the code was altered and ported to the Windows operating system. Windows has extra overhead in their system calls and is generally a slower O.S. than UNIX systems. To test the effects of different operating systems the Imrecon routine will be run using Windows and UNIX, and compared for relative performance. One system has both a UNIX system (Linux) and Windows 95, thus any overhead added by the O.S. can be easily compared.

4. Parallel Processing

Parallel processing is the ability to simultaneously compute or process data. Generally, a parallel processor contains multiple similar CPUs. Various interconnection routes, and memory schemes have been implemented to allow communication between the CPUs. There are two main memory architectures for parallel processors; they are shared memory system and distributed memory system. These two systems are summarized in sections 4.1 and 4.2.

Scientific and engineering programs are well adaptable to being either parallelized or multithreaded. The reason is that generally the same computation is done on different data sets. This is especially true for image processing. For example, when the mathematical calculations are performed on different images and each image is independent of each other, the calculations could either be performed on a parallel processor or multithreaded machine, resulting is an efficient program. The inherent parallelism within programs (such as Imrecon), can fully utilize parallel processors, and achieve good linear speed-ups.

A good performance increase would be a linear speed-up . Linear speed-up means that the execution time will be inversely proportional to the number of nodes. An example of linear speed-up is; if the run time for one node is 20 seconds, then for two nodes it should be 10 seconds. Essentially the execution time decreases linearly as the number of processors increases. The theoretical speed-up is limited

by Amdahl's law. That is if there is an uneven distribution of operations among the processors [31], performance will suffer. For example, in a 2 node system, if a program has 5 unequal parallel units, and one processor gets the 3 largest units, and the 2 others are given to the other node, this would result in poor speed-up curve. Thus, a good speed-up depends on the parallel units being distributed evenly over the n-nodes, as well as the execution times for the units be similar to maximize the performance. To achieve maximum performance, an even work load distribution is critical.

The idea of using multiple processors both to increase performance and to improve reliability dates back to the earliest electronic computers. About 30 years ago, Flynn [32] proposed a simple model of categorizing all computers that is still useful today. Flynn looked at the parallelism in the instruction and data stream calls, and according to the constraints of the machine, all computers were placed in one of the four categories:

1. *Single instruction stream, single data stream* (SISD), -- Standard uniprocessors i.e. Pentium
2. *Single instruction stream, multiple data stream* (SIMD), -- The same instruction is executed by multiple processors using different data streams. Each processor has its own data memory (hence multiple data), but there is a single instruction memory and control processor, which fetches and dispatches instructions. The processors are typically special purpose, since full generality is not required.
3. *Multiple instruction stream, single data stream* (MISD), -- No commercial machine of this type has ever been built, to date.

4. *Multiple instruction stream, multiple data stream* (MIMD), -- Each processor fetches its own instructions and operates on its own data. The processors are often off-the-shelf microprocessors.

There are many hybrids of the above categories, but the most common types are SISDs. From a historical perspective, most of the earlier multiprocessors were SIMD, and a resurgence of this type was seen in the early 1980's. Recently, MIMD has emerged as the choice for general multiprocessor systems, mainly due to the following two reasons [32]:

1. MIMDs offer flexibly. With the correct hardware and software support, MIMDs can function as single-user machines focusing on high performance for one application, as multiprogrammed machines running many tasks in parallel, or a combination of these functions.
2. MIMDs can be constructed from off-the-shelf microprocessors, providing a cost/performance benefit. Many of the MIMDs are built from uniprocessors found in common workstations or personal computers [33,34].

4.1 Shared Memory Model

The shared memory model consists of 1 single memory for the system. All nodes of the parallel processor utilize this one memory setup. The shared memory model advantages are: 1. Any one processor can easily access global memory, or variables, 2. no memory passing needs to be included and 3 only one memory to control. Disadvantages to the share memory model are: 1. Significant bottle necks occur when all processors try to access memory at the same time, 2. only one processor may access memory at same time, 3. difficult to maintain memory

consistency and 4. numerous memory semaphores must be maintained to prevent data corruption [32].

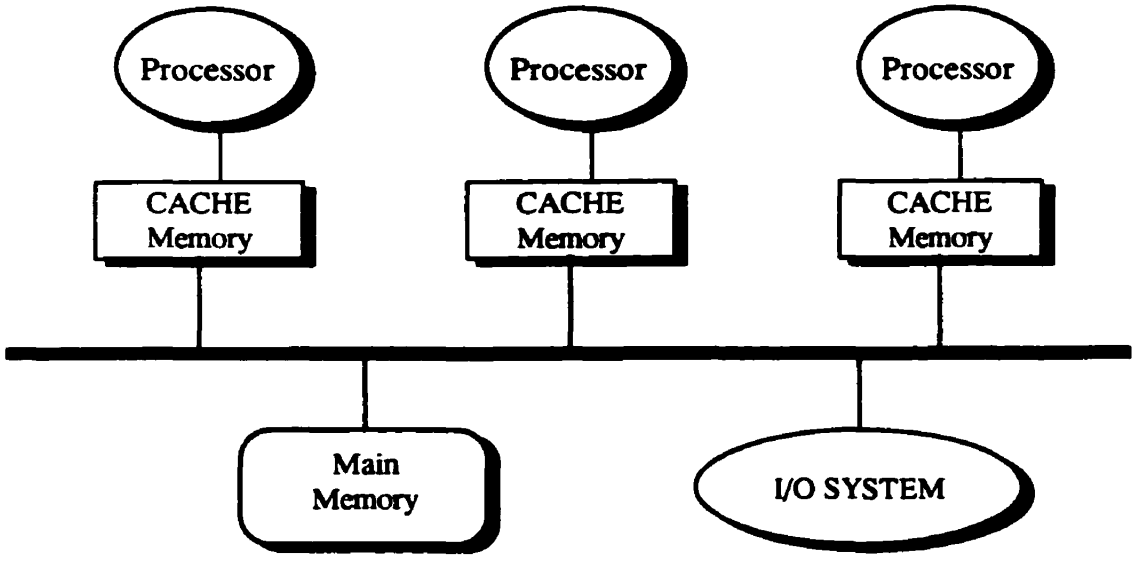

Figure 4-1 - Shared Memory Architecture

4.2 Distributed Memory Model

Distributed memory implies that all nodes have their own memory system. Thus all memory accesses and data are local to each processor. Advantages are: 1. Data is local, no bottle necks, 2. faster memory access and 3. no need for complicated memory organization. Disadvantages to distributed memory system are: 1. If processors need data from other processors, slows down the bus and 2. requires data passing.

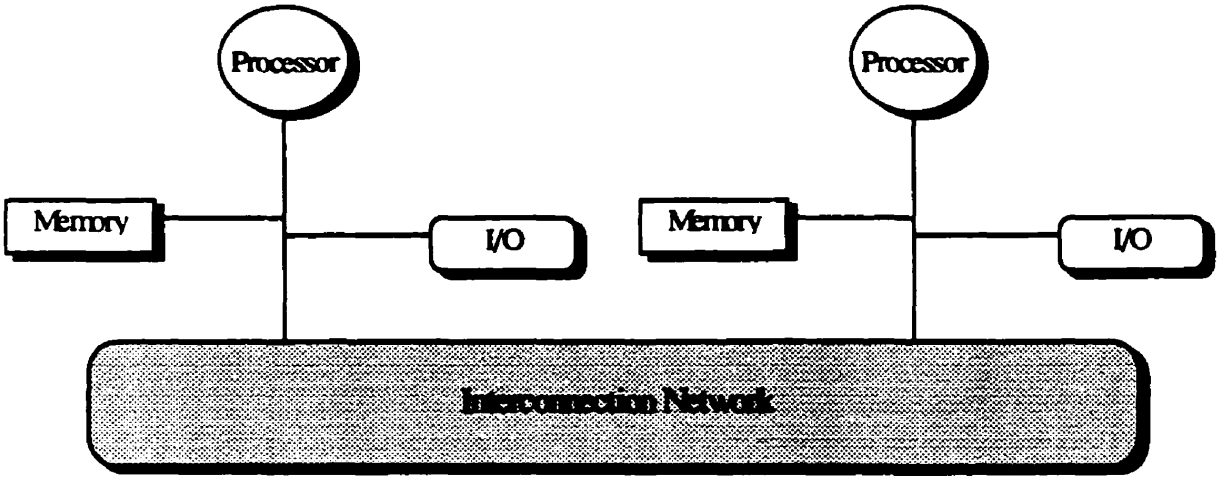

Figure 4-2 - Distributed Memory Architecture

4.3 EARTH-MANNA System

The EARTH (*E*fficient *A*rchitecture for *R*unning *Th*reads)-MANNA, is a multithreaded parallel processor with 20 nodes. This system was developed by the ACAPS group at McGill University, under the supervision of Prof. G. Gao. The multithreaded execution has been simulated with software on the EARTH machine. The true MANNA parallel processor does not support multithreaded execution, but lent itself well to support multithreading. Each node on the parallel processor contains two Intel 50 MHz i860XP CPUs. One CPU is primarily for execution of code (EU), and the second in used for synchronization and communications tasks, and named the synchronization unit (SU). Message queues are used to pass information between the EU and SU. Threads on the EARTH system may only execute when all its data is ready and available.[50]

4.3.1 MANNA System Specifics

The MANNA (*M*assively parallel *A*rchitecture for *N*umerical and *N*on-numerical *A*pplications) was developed by GMD FIRST of Germany. The MANNA system was developed as an European effort to be competitive in the global market place. The system uses a pioneering 2 CPUs per node. W. K. Giloi is one of the principal design engineers to start the project. Currently, they have a 40 node machine and are developing a machine based on the PowerPC 620 by Motorola [35].

4.3.2 EARTH System Specifics

The MANNA system, as mentioned before, is not a multithreaded architecture, but has been emulated by the software created by the EARTH project. This introduces a software overhead, that can affect the performance of the overall system. Various benchmarks have been tested on EARTH, and proved that the overhead is minimal in well-coded programs. A new measurement has been introduced to capture the true nature of the EARTH system, the USE (Uni-Node Support Efficiency) factor. The measurement is the sequential execution time versus the threaded execution time on one node as shown in Equation 4-1.

$$USE = \frac{T_{SEQ}}{T^{1}_{THREADED}}$$

Equation 4-1

A good USE factor would indicate that there is: Sufficient parallel threads to hide the latency of multithreading operations, (i.e. context switching); the overhead of

performing multithreaded operations is minimal; multithreaded code is not intrusive on sequential code [50]. Please see reference [50] for further detail.

4.4 Design Considerations for Parallel Processing

When designing programs for parallel processors, one must first consider if the program can be parallelized to take advantage of the parallel processing power. The essence of such analysis concentrates on the loops within the program and the general structure of the program. The first step in analyzing the program code is to determine where most of the computation time is spent. A Rule of Thumb of programs is that 90% of the execution is done by 10% of the code. This holds true in the general case, but not necessarily for the code on hand.

A parallel processor needs to be programmed such that each node of the parallel processor is efficiently used. A program that uses only half of the available nodes is not effectively using all available resources. Regarding the original Imrecon code, the program that is currently used to perform the 3D ultrasound reconstruction, needs to be analyzed to determine where it can be executed on a parallel processor.

Upon inspection of the running code, most of the execution time is spent within a routine called Imrecon, this routine is basically the kernel of the program where all the necessary data is combined to produce the resulting 3D image. On the SGI (R3000 33Mhz), the execution time for a 10M file is approximately 110 seconds,

of this 110 seconds, 105 seconds are within the Imrecon routine. Thus, the main focus of parallelizing the code will be within the Imrecon routine, since the routine monopolizes 96.5% (105/110) of the total execution time.

The main structure of the Imrecon routine is as shown in Figure 4-3 and depicted in Figure 4-4.

```
imrecon (...)
{
    ACC[i] = 0;
    SHADOW[i] = 0;

    for ( i = 0; i < num_images ; i ++)
    {
        pre-computations
        for ( i = 0; i <MAX_X ; i ++)
        {
            pre-computations
            for ( i = 0; i < MAX_Y ; i ++)
            {

                Calculate target location for voxel
                Find intensity of each pixel value and store value in
                      target location
                ACC[target] = ACC[target] + intensity;
                SHADOW[target] = SHADOW[target] + 1;

            }
            set-up for next iteration
        }
    }
    VOXEL_SPACE = ACC/SHADOW;
}
```

Figure 4-3 - Pseudocode for Imrecon routine

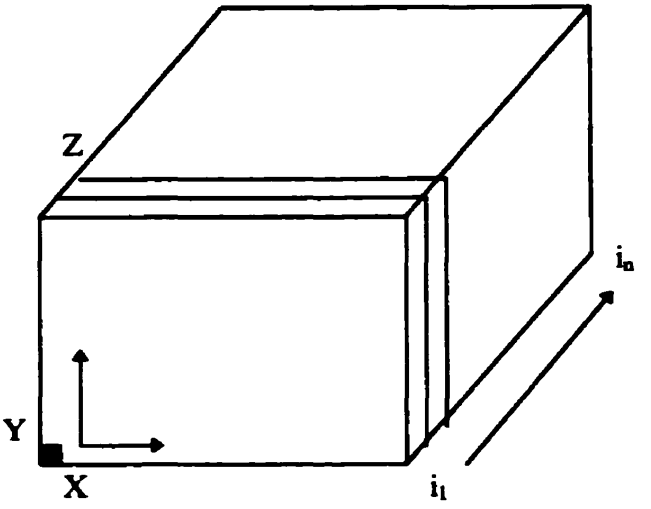

Figure 4-4 - Descriptive Picture of Pseudocode

There are three main loops within the Imrecon routine. The first loop is on a per image basis, the second loop traverses the x-axis and the third loop the y-axis. With these three loops, each pixel within each captured image is used to calculate the voxel data. The variables ACC and SHADOW are subsequently divided such that the integer version answer is used as the *voxel* data. Equation 4-2 gives the division used to calculate each *voxel* intensity.

$$Voxel\_Intensity[i] = \frac{ACC[i]}{SHADOW[i]}$$

Equation 4-2

The coordinates for the *voxel* intensity are calculated from the *a.dat* file; an input file accumulated during the scanning process which contains the tracker data. Each pixel must go through a set of transformation, as shown in Equation 4-3.

$$Voxel\_location[i] = RC \cdot TR \cdot image\_to\_locator \cdot [x, y]$$

where RC = a corrective factor used to minimize distortion, also a translation to the origin, essentially transforms from *3-space* to *voxel space.*

TR = Rotation and translation in *3-space*

image_to_locator = Takes the image and translates it to *3-space* coordinate system

$[x,y]$ = pixel location on the image

Equation 4-3 - Voxel Space Transform Equation

On close inspection of the code for the Imrecon routine, the outer loop was found to have no loop carried dependencies. There were the two variables ACC and SHADOW, but they are accumulative within the inner most loop. However, the two inner loops are loop dependent. They cannot be executed in parallel. Such a parallelization will result in erroneous *voxel* data and *voxel* locations. The variables ACC and SHADOW are cumulative from the previous loops, such that the calculated value from each loop can be added at the termination of the outer loop. Simply put, the outer loop can be executed in parallel.

4.4.1 Data Passing

The last step in the reconstruction required that the data from each node be added together to form the final reconstructed 3D image. As mentioned in the previous section, the two variables ACC and SHADOW need to be summed together prior to the last divide. The image will be reconstructed in various steps to take

advantage of the parallel processing power. Two methods are presented to rebuild the final image, a multithreaded version and non-multithreaded version.

4.4.1.1 Method 1 - Non-multithreaded Version

The structure of this method has the data sequential built from one step to the next. The first step has the data passed to a neighboring node, and the neighboring node adds the data. This is continued, until a final image is constructed at node 0. Figure 4-5 depicts the method.

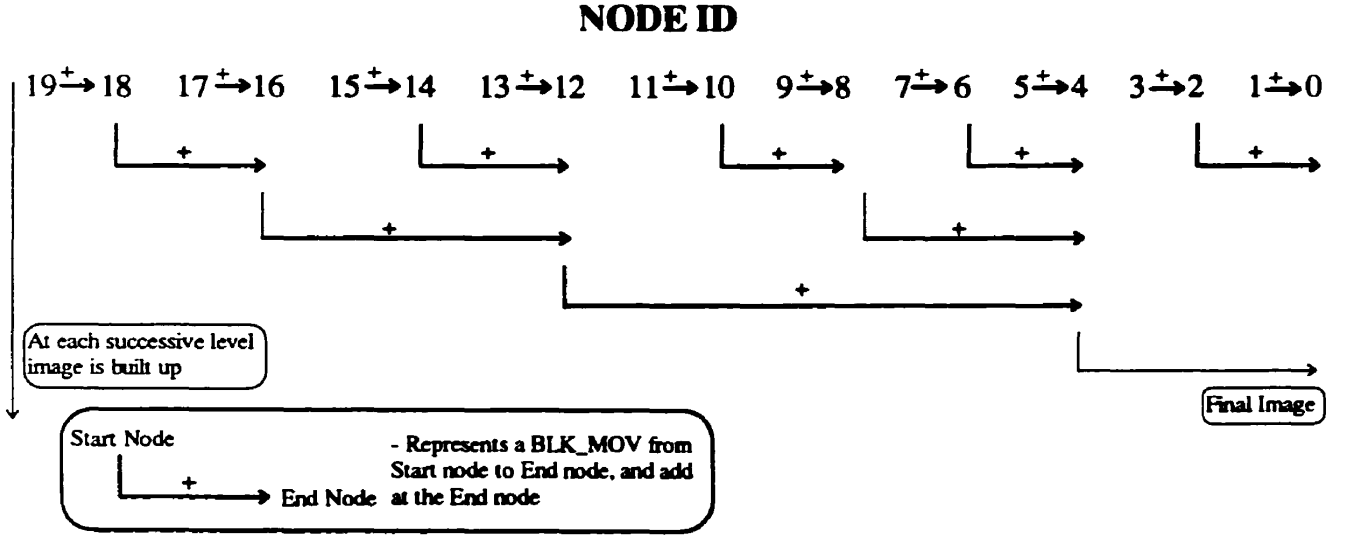

Figure 4-5 - Overview of Rebuilding Image - Non-Multithreaded Version

4.4.1.2 Method 2 - Multithreaded Version

The multithreaded version divides the data such that each node is responsible for a smaller portion. Each node then requests the same block of data from all other nodes and then adds each variable. Once each node has completed the task, the data block is passed data back to node 0, where the final image is assembled. See Figure 4-6.

It should be noted that even though the multithreaded version should be faster, the performance gain will still be limited to the bottle neck of the available bandwidth on the inter-connection network between each node.

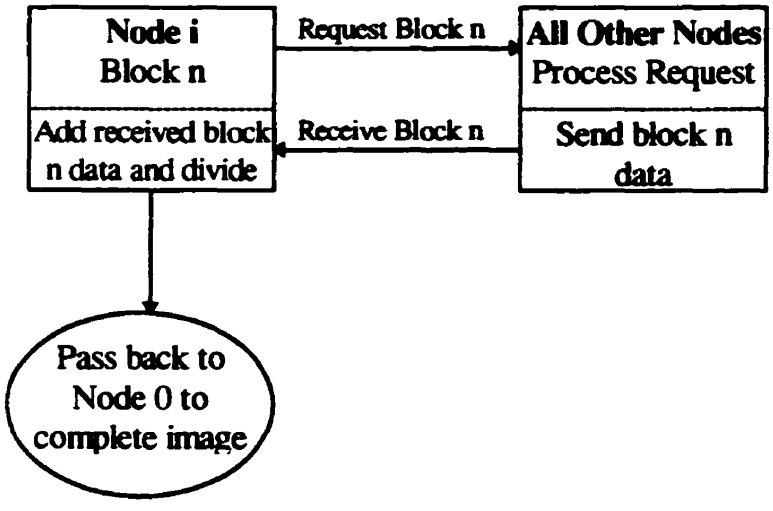

Figure 4-6 - Graphical Description of Multithreaded Algorithm

5. Results

This section has been divided in the following manner. Section 5.1 presents the results from the performance enhancements, without the fixed point conversion. A histogram analysis is presented to verify if the optimizations effected the image quality. Section 5.1.2 presents the fixed point results. A histogram analysis is also presented to verify the accuracy of the fixed point version versus the floating point version. The fixed point version will not be compared for speed, since the program was run under the Windows© 95 operating system. Section 5.2 presents the results of the comparison between the Windows 95 operating system and the Linux operating system. Section 5.3 presents the execution times of the parallel processor and achieved speed-up.

Although many different biological organ data sets were not used to verify the findings reported above, all ultrasound images are similar in a sense of image quality. In this research each data set consisted of a large number of images used to analyze the experimental results (approximately 200 images per data set). The results and conclusions were based on the evaluation of approximately 350 images. Also, it should be noted that a different data set was used for the fixed point volume rendered images, but was not included for brevity of the thesis and the results were similar to that of the egg. The egg used for the rendering depicted a better image representation for people unaccustomed to ultrasound images.

5.1 Performance Enhancements

All enhancements were performed, except for the fixed point algorithm, on the UNIX based systems, the SGI (R3000) and Linux (Pentium 200). Because of difficulties associated with implementing assembler routines on the UNIX systems (compiler would not recognize assembler commands), the fixed point algorithm was only run under the Windows© 95 and Windows© NT systems. For the fixed point comparison, speed-up results will be presented between various processors. A statistical analysis of the floating point version versus the fixed point version will be performed for accuracy of the reconstruction.

The optimized code achieved an average speed-up of 16.8 % across the platforms. The speed-up times are presented in Table 5-1. The resulting images are accurate and have no loss of information. Figure 5-1 and Figure 5-2 are examples taken from the reconstructed images of the original code and optimized code, respectively. Visual inspection of the images confirms that the optimization has not reduced the image quality and accuracy. Furthermore, this has been further verified by the histogram analysis presented in Figure 5-3 and Figure 5-4. The mean and standard deviation for both images are identical (see Table 5-2). The above findings confirm that the speed-up achieved by the optimizations are accurate and have no adverse effects on the quality of the images.

Table 5-1 - Percentage Increase of Optimized Code

Platform	Original (sec)	Optimized (sec)	% Increase
SGI R3000	120	98	**18.33**
Pentium 200	14	10	**28.57**

Figure 5-1 - Example Image from Original Code

Figure 5-2 - Example Image from Optimized Code

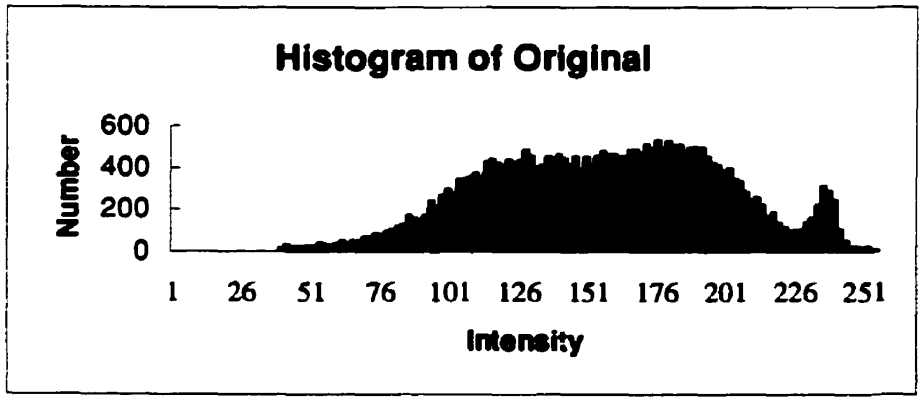

Figure 5-3 - Histogram for Original Code

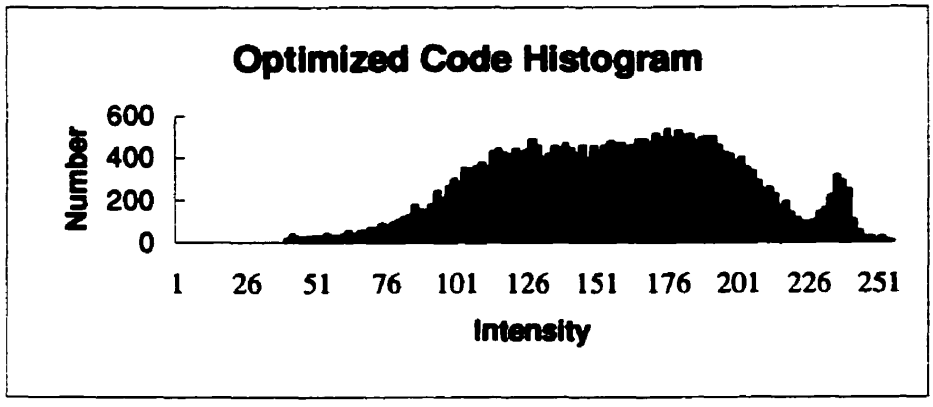

Figure 5-4 - Histogram for Optimized Code

Table 5-2 - Mean Pixel Intensity

	Original	Optimized	%error
Mean	181.50	181.50	0.00
Std	57.25	57.25	0.00

5.1.1 Additional Features

This section deals with reconstruction speed when reconstructed slices are seen on the display as they are completed. The tests were performed only on the Pentium 200. From Table 5-3, the "with image display" execution adds about 40% extra time when compared to "without image display" execution. This is a considerable performance hindrance on the execution of the code. Although, when compared to the three step process of the "without image display" execution that must wait for the program to complete, save the data, and then start a new program to view the images, the loss is acceptable.

Table 5-3 - Performance Comparison with Image Display

	With Image Display	Without Image Display
Reconstruction Time	14 s	10 s
Time to Display	14 s	25 s

5.1.2 Fixed Point Version

The fixed point version was executed on a Windows NT and Windows 95 platform. Various CPUs were used, a Cyrix PR233, Intel 166MMX and Intel 200MMX, the computer systems have 64 M on board, though the Cyrix uses SDRAM. SDRAM has faster access time (10ns vs. 60ns) than standard memory, which may affect the results. It should also be noted that the Floating Point Unit (FPU) of the Intel chip is far superior than that of the Cyrix chip [36]. The size of the data files executed on both platforms where 6.4M and 10.4M. The results are summarized in Table 5-4

Table 5-4 - Fixed Point vs. Floating Point execution Times

		6.4 M File			10.4 M File		
CPU	O.S.	Fix. Pt	Fl. Pt.	Ratio	Fix. Pt	Fl. Pt.	Ratio
Pentium 166MMX	Win'95	6	17	2.83	12	26	2.17
Pentium 200MMX	Win'95	5	13	2.60	10	22	2.20
Cyrix PR233	Win NT	6	19	3.17	12	33	2.75

The ratio column in Table 5-4 is the division of the floating point execution time versus the fixed point execution time. The value is indicative of the performance increase between the fixed point version and floating point version.

The next set of figures are the samples from the fixed point and floating point reconstructed images. They have been presented here for visual comparison purposes. The histogram and correlation function alone are not sufficient to judge the image, the images themselves need to be inspected to determine whether the fixed point reconstruction algorithm loses quality and detail.

Figure 5-5 - Floating Point Image

Figure 5-6 - Fixed Point Image

As seen in Figure 5-5 and Figure 5-6, there seems to be more detail in the floating point version, but this is not the case for all images. Though not presented here, several of the images seem to have more detail in the fixed point version rather than the floating point version. Also, in general, the fixed point version seems to smooth out the images, and thus is less noisy.

The histograms of the images in Figure 5-5 and Figure 5-6 are presented below in Figure 5-7 and Figure 5-8. The two histograms are very close in shape, but do differ in some regions of the histogram. The difference seems to be in amplitude at certain locations. The original image has sharper peaks, whereas the fixed point version has flatter peaks. The fixed point version seems to be smoother in the transitions between intensity values, which correlates to the smoother image. The statistical values for the images are almost identical, as shown in Table 5-5.

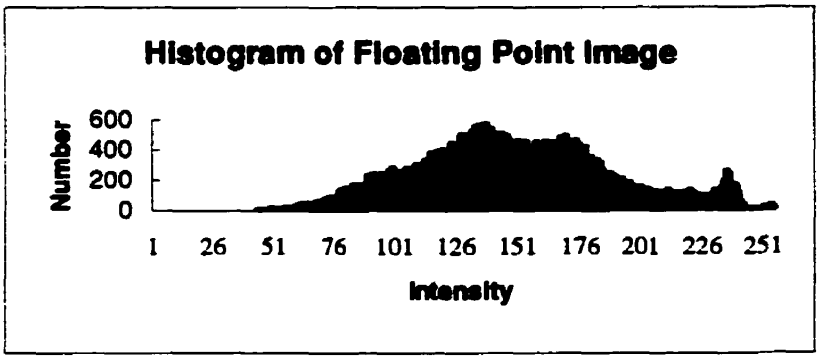

Figure 5-7 - Histogram of Floating Point Image

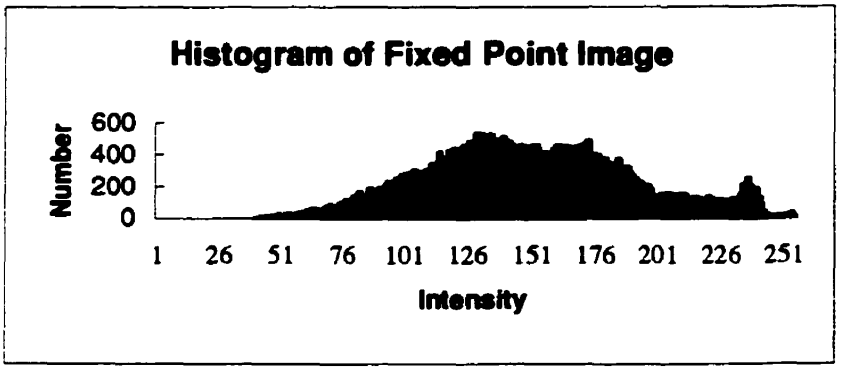

Figure 5-8 - Histogram of Fixed Point Image

Table 5-5 - Statistical Analysis of Fixed Point vs. Floating Point

	Non-Fixed Point	Fixed Point	% error
Mean	174.31	174.42	-0.06
Std	57.54	57.37	0.30

An additional histogram comparison was performed to verify the precision of the fixed point algorithm. An A-line scan (one column of data on the ultrasound image) was analyzed using the same histogram analysis method. No difference was found between the resulting histograms, and a subtraction of the two histograms yielded no difference (statically and visually) between the two A-line scans.

Figure 5-9 is a graph of the correlation coefficients between respective images of the fixed point version and floating point version. Except for the first few images and last few images, the correlation coefficient is on average 96 %, which signifies that images are closely related. Including all images, the average coefficient is 93 %. The images at the beginning and end differ significantly, since these images are not complete due to the freehand scanning technique. The images at the beginning and end are incomplete because there is insufficient information at these locations to provide full images. It takes approximately 5 full 2D images to produce one 3D image, without enough 2D images the 3D image cannot be fully reconstructed.

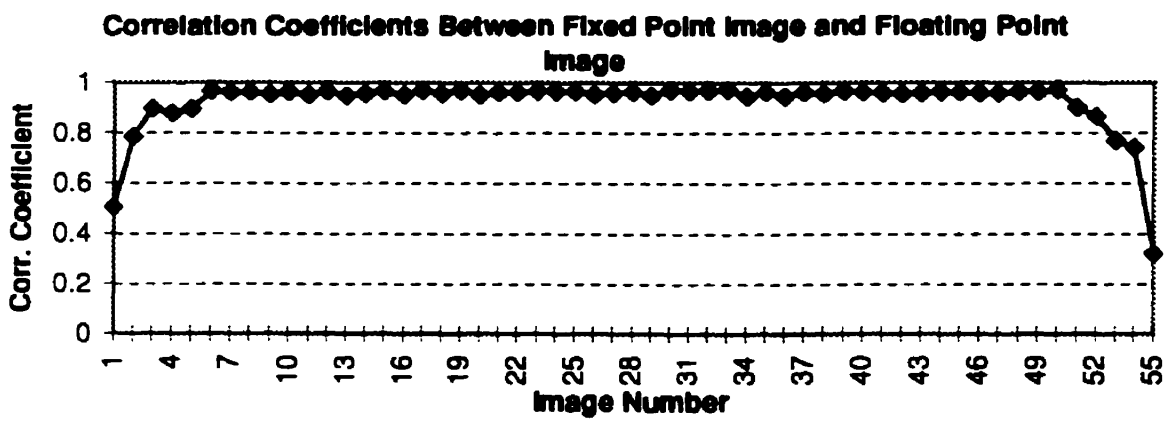

Figure 5-9 - Correlation Coefficients Between Fixed Point and Floating Point Images

As a last comparison between the fixed point and floating point algorithm, the data sets have been rendered to show differences in the images that may be more apparent during volume rendering. A hard boiled egg *in-vitro* was used for the comparison, since it mimics soft tissue properties. The fixed point images seem softer, in the sense that the edges are not as clear nor as bright, as can be

compared in Figure 5-10 and Figure 5-11. Figure 5-12 shows the difference (subtracting floating point version from fixed point version) image, and Figure 5-12 is a contrast enhanced version. The contrast enhanced version clearly shows that edges are a main difference within the images.

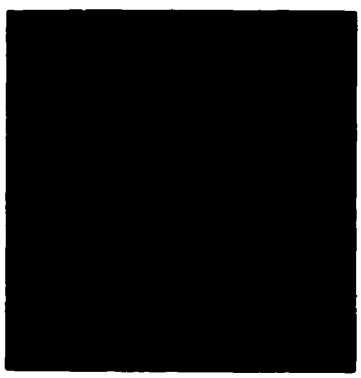

Figure 5-10 - Rendered Floating Point Version

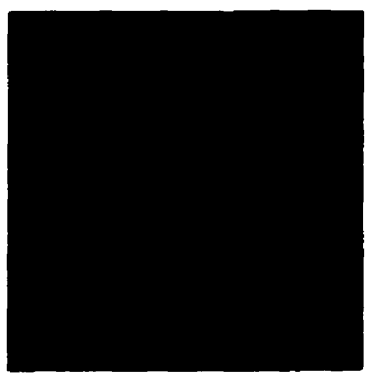

Figure 5-11 - Rendered Fixed Point Version

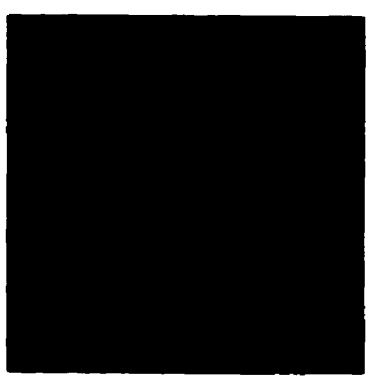

Figure 5-12 - Difference Image Between Floating Point And Fixed Point Rendered Images

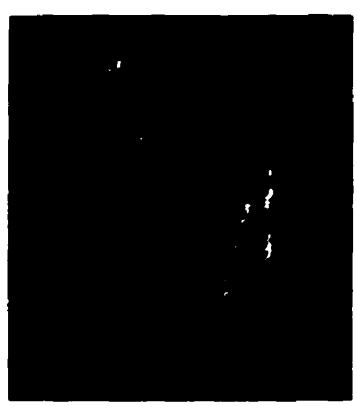

Figure 5-13 - Contrast Enhanced Difference Image

5.2 Operating Systems

The overhead associated with the operating system is much greater than those under the UNIX operating system. The time of execution for the Windows 95 version was 13 seconds for the fixed point version, which should be faster than the

floating point version. The program was also executed on the same computer (Pentium 200) but under the Linux operating system; the same file (6.4 M) was reconstructed in 10 seconds. Under Windows© 95, a file size of 10.4 M was reconstructed, but the execution time was 55 seconds, though the computer only had 32 M of RAM, which is not very linear in terms of comparing the 6.4 M file to the 10.4 M file (note, the computer only had 32 M of RAM, which is not optimal for the larger file size). Unfortunately, due to the limited memory space, the same file could not be executed under Linux. Thus, from the results, Windows 95 adds up to 30% overhead, and does not lend itself well for fast execution. No direct comparison was made between Windows NT and Linux, since the available computer could not support both operating systems.

5.3 Parallel Processor

This section deals with the results of the execution on the parallel processor. The speed-up of the actual Imrecon routine was linear. This is shown in Table 5-6 and Figure 5-14. From Figure 5-14, it is easy to see that the speed-up is linear. No multithreading was performed on this routine, since the results indicate a linear speedup was achieved. Multithreading aims to hide latency and improve linear speed-up, but in this case, the algorithm was already linear, thus required no multithreading. Furthermore, the tradeoff between performance gain and added development time could not be justified due to the achieved linear speedup. It should be noted that the I/O times have not been included in the data, since the I/O on the MANNA system is very poor and not indicative of a true parallel system.

The parallel system communicates with a UNIX host for I/O functions such as reading or writing of data files. Data transferred over the network from the host computer to the parallel processor increases read/write operation times.

Table 5-6 - Speed-up Times for Manna Execution

Number of Nodes	Execution Times (s)	Speed Up
1	85.9	1.00
2	41.6	2.06
4	21.5	4.00
8	10.4	8.26
16	5.6	15.34
20	4.5	19.09

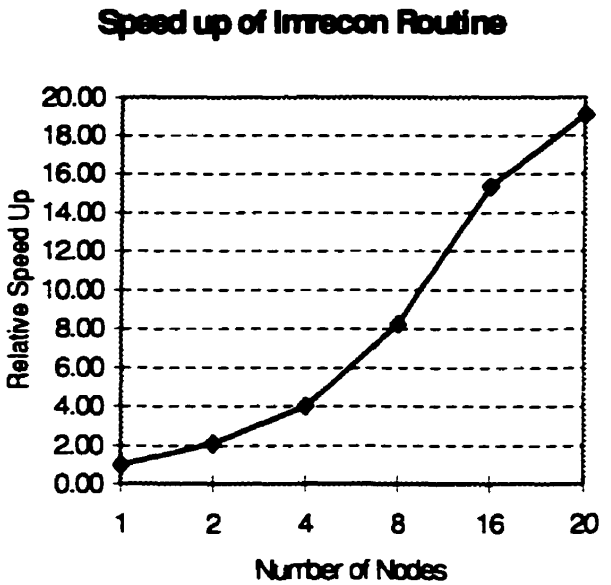

Figure 5-14 - Speed-up Curve of Imrecon Routine

Table 5-7 presents the results for the complete execution, including reassembling the image across all the nodes. The speed-up has been reduced, when including the reassembling of the image across all nodes. To improve performance, a multithreaded version of the reassembling was implemented. This modification resulted in moderately improved execution time and speed-up curve. Performance

gains were consistent for all nodes using the multithreaded version. This can be observed in the times reported by each node for the given data sets. Comparing the 20 node versions, there is a 6 second gain with the multithreaded version compared to the non-multithreaded version.

Table 5-7 - Complete Execution Time and Speed-Up (Non-Multithreading)

Number of Nodes	Total Execution Time (s)	Speed-Up
1	86	1.00
2	52	1.65
4	34	2.53
8	27	3.19
16	22	3.91
20	21	4.10

Table 5-8 - Complete Execution Time and Speed-up (Multithreading)

Number of Nodes	Total Execution Time	Speed Up
1	85.2	1.00
2	51	1.67
4	27.6	3.09
8	17.8	4.79
16	22.6	3.77
20	14.78	5.76

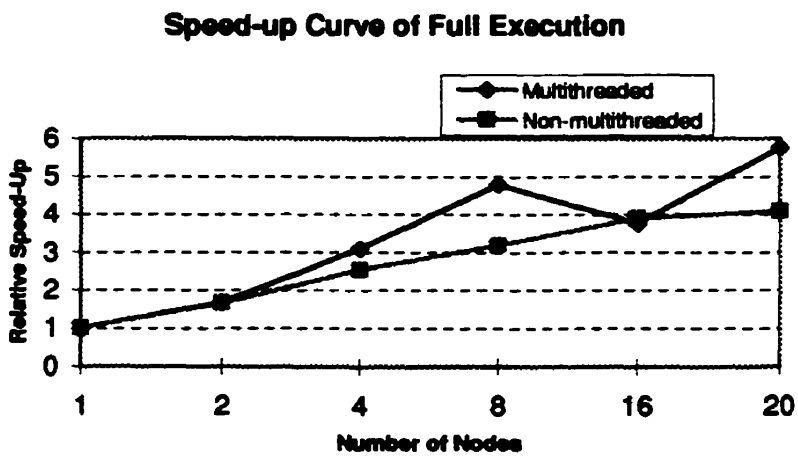

Figure 5-15 - Speed-Up Curve Of Total Execution Time

6. Discussion

To results of this study suggest that a sequential processor is efficient and powerful enough to do real-time 3D reconstruction. A distinction, however, must be made between 3D reconstruction and 3D interactive volume visualization system. The 3D reconstruction is a part of the full system. Volume rendering is another step within this pipeline, which is also computationally intense. Parallel processors are still required to achieve 3D interactive volume visualization. The remainder of the discussion focuses on the results obtained in this study.

Programming simplicity, learning curves, and relative speeds will help determine whether the use of parallel processor is efficient and effective with the advent of the new generation super scalar processors, such as the Intel Pentium, DEC Alpha, and the R10000. A direct comparison between the EARTH-MANNA and sequential processors, in terms of speed, would be an unfair comparison, considering that the MANNA system uses an outdated i860 Intel processor.

The results for the Pentium and DEC Alpha processors are encouraging. Achieving speeds of sub 10 seconds, and sub 5 seconds for reconstruction surpasses that of a motorized 3D transducer, with the added benefit that the transducer is easier to manipulate and can be added to any existing system. The speed improvements are based primarily on the processor used and the code optimizations, including the fixed point version.

Section 6.1 below discusses the code improvements and fixed point algorithm. The subsequent section (section 6.2) focuses on the parallel processor. The ease of use factor and current processors trends and performance are discussed in sections 6.3 and 6.4, respectively.

6.1 Code Improvement and Fixed Point Algorithm

From the results section, the optimized code runs faster than the original code and is accurate as well. There was no loss of information by optimizing the code execution. This extends the common believe that program code has room for improvement [36]. The optimization performed on Imrecon was fairly high level, and logical. The math improvements were simple to identify and improve. Further optimizations could improve or decrease execution times, by replacing current time consuming C routines with assembler routines. The advantage of the assembler routine is that the compiler will not perform the C to assembler conversion, and the programmer can use more advanced instructions or take advantage of the hardware architecture available for a particular CPU, such as a Pentium MMX that can act like a SIMD machine. This would obviously limit the portability of the code, but would increase performance. Taking the Pentium as an example, the floating point unit can execute instructions in parallel to the CPU once the instruction is issued. Thus, integer operations can be executed simultaneously as floating point instruction [36].

Table 5-4 summarizes the execution times between the floating point version and fixed point version. For the fixed point Windows NT system, the execution times are very impressive, though not linear. The reconstruction was twice as fast for the 6.4 M file, but was not half the 10.4 M file. The main reason for this discrepancy is due to memory access. The 6.4 M file is a small file, which can fit into one sequential memory block, whereas the 10.4 M file requires multiple memory blocks. Non-sequential access is slower than sequential access and slows overall performance. In both cases no hard drive access was detected during the reconstruction phase, which would have contributed to further increases in reconstruction time.

The floating point version was slower than the fixed point version on the Windows NT system, which was anticipated by the poor FPU of the Cyrix chip. The floating point version seems to be more linear in terms of speed up versus file size. There is a 62.5% difference between the file sizes and 73.7% difference between the computation times. This difference is not as vast as the 100% difference with the fixed point version. The slower execution time of the floating point version could contribute to this difference. The slower execution allows for the CPU to have more time to pre-fetch or cache the data. The data then will be ready when required.

The fixed point algorithm when executed on the Pentium 200MMX, with 64 M of RAM and Windows 95 operating system, is the fastest reconstruction achieved (5

seconds), even including the Linux system. The DEC Alpha is not compared since no fixed point version was executed on it. The Linux system (Pentium 200) reconstruction was performed using only the floating point algorithm, and computation time was 10 sec. vs. the 10 sec. of the Windows 95 system. When the system was equipped with only 32M of RAM, the Linux system performance dominated the Windows 95 system with a reconstruction time of 10 seconds versus 55 seconds. With only 32 M of RAM, the Windows 95 system must constantly access the hard drive for virtual memory, resulting in poor performance. Though the times are equal when the system is equipped with 64 M of RAM, the Pentium 200MMX is a faster processor because of the internal design advancements, such as larger internal cache (16 kb vs. 32 kb), SIMD commands, and newer instruction set [37]. The results are a good indication that performance is not solely based on the CPU, but also on the operating system. These results show the superiority of the UNIX system versus that of the Windows systems in terms of efficiency.

The accuracy of the fixed point algorithm is comparable to that of the floating point version. The floating point version is accurate based on the work conducted by Watkin *et. al.* [5]. The histogram of the fixed point version (Figure 5-8) differs slightly from that of the floating point version (Figure 5-7), though the general shape is consistent. These finding were consistent with all reconstructed images slices. The standard deviation and mean are virtually identical for the two presented histograms, further indicating that they are similarly distributed. Visual

inspection of all images in the fixed point reconstruction attests to the comparable image quality, though in some cases the fixed point images seem sharper, but in other instances the floating point images seem sharper. This effect is attributed to the rounding off and conversion between floating point and fixed point variables. Some variables need to be converted to fixed point from either integer or floating point, and then converted back, which causes rounding errors to occur twice.

The fixed point versus floating point images are well correlated (96%). The computation speed gains seem to outweigh the minor differences between the reconstruction algorithms. There is no significant loss of information, nor loss in detail to prevent the use of the fixed point algorithm.

The volume rendered images show that the fixed point algorithm is comparable to that of the floating point version. As mentioned above, the fixed point version seems softer, which shows up well within the rendered images. The contrast at the edges in the floating point version seems brighter, and the two difference images (Figure 5-12 and Figure 5-13) support this claim. The fixed point version is approximately 3 times as fast as the floating version, and the accuracy is relatively close, considering the trade-off the use of the fixed point version seems justified. If the fixed point version is re-implemented as a fixed point version from scratch rather than converting between floating point, integer and fixed point, the accuracy would improve and should be at par as the floating point version.

6.2 Parallel Processor

All discussion below is specific to the 3D ultrasound reconstruction program and problems revolving around it, and in no way do the ideas and conclusion presented here reflect on MANNA as a system specifically.

Based on the execution times on the core function of Imrecon, the speed up was linear as shown in Table 5-6. This linearity was achieved without the use of multithreading within the function. The extra time necessary to re-code the algorithm to include multithreading does not seem justified considering the performance of the algorithm. For the 2 node and 4 node configurations the execution times were actually faster than the single node basis. This is accounted for by the fact that the times are reported by the node 0, which after load distribution has the least amount of work to do. If there are 101 images to be reconstructed, node 0 would be responsible for the first 50 image, whereas node 1 would be responsible for 51 images, thereby having more work to do, but the time reporting would be done by node 0. For the tests, a 10.4M file was used which was comprised of 201 images. The distribution for the 4 node case would, be 50,50,50,51, respectively. Again node 0 would finish ahead of node 3. The times seem faster than a linear speed-up, but taking into account that node 0 displays a time faster than available, the speed-up is linear. One image would take an extra 0.46 seconds based on the results presented for the 1 node case in Table 5-6, which affects the speed-up curve.

As mentioned above, a linear speed was achieved. To understand the achieved speed-up, the program code must be examined. The section of code which was performed in parallel was provided in Figure 4-3. This section is fairly loop independent (a criteria for parallelizing), but more than that the computations are straight forward and each processor has an equal amount of work to do. The work required to compute the volume image was divided on a per image basis (the outer most loop), thus further simplifying the work done by each processor. Since, the computations were the same for each processor, and no inter-processor communication was necessary during the loops, a linear speed-up was achieved.

When the full reconstruction process is taken into consideration, the performance deteriorates. The speed-up curve is disappointing, with poor reconstruction times. The primary problem is the shear volume of data that is required to be passed among the various nodes and finally to node 0, where communication is established with the host computer. The file size that was used is 10 Mb, which obviously is a large file that requires a large amount of bandwidth. The passing of data among nodes is the bottle neck for this application program on MANNA. The code to rebuild the image incorporates multithreading, such that each node requests data from the other nodes, and then proceeds to do the required calculations. Initially the code did not use any multithreading and the results were even worse as shown in Table 5-7. This one problem shows the positive effect of multithreading on a platform that supports it. With 2 nodes and 4 nodes, the effect of multithreading is not as affective since the bandwidth on the interconnect

network is not saturated. As the number of nodes increases, the interconnect network becomes saturated and adversely affects performance. Also, it should be noted that the speed-up curves in both cases seem to approach an asymptote. The large amount of data and required data passing presents an upper limit in the reconstruction speed-up. A true linear speed-up will not be possible with such large volumes of data, methods need to be devised to correct the problem, such as increasing allowable bandwidth, or faster inter-connection networks. Wittenbrink *et. al.* [38] have focused their work on various methods to reduce communication costs on parallel processors. They use a MasPas MP1 as a test platform and image warping as the specific application. Their results indicate that exclusive- read-exclusive-write (EREW) performs better than concurrent- read-exclusive-write (CREW) by almost a 100% [38]. This is similar to the finding conducted for this research on parallel processors, where the limitation was the communication costs over the interconnect network.

The degraded performance is directly related to the distributed memory system [39]. The distributed memory system must communicate a large set of data between all nodes to create the final image, since the algorithm sums the data from each node to produce the final 3D data set. The result is a saturated inter-connection network thereby increasing execution time. If a shared memory system were used, the performance would improve, plus all communications would be transparent to the user [40]. The performance has not been verified with a distributed system, but the data would not be passed between nodes, resulting in

less traffic on the interconnect network. The speed-up would not be linear, since there would be a bottle neck related to memory access. Semaphores, critical sections, or shared variable programming techniques [41] would be required to ensure the correctness of the data.

The USE factor was not presented here, since the main function Imrecon does not use any multithreading. The USE factors primary indication is on how well the code hides the threads. The USE factor requires a multithreaded version for a single node execution as well as a multi-node execution, since the 3D data rebuilding stage is not required for single node execution, the USE factor cannot be evaluated.

Parallel processor's execution architecture is well suited for volume rendering algorithms, where the images can be sub-divided and individually volume rendered. Volume rendering on parallel processors has been accomplished with good success by many researchers [42,43,44]. Regarding volume rendering to the problem at hand, and considering that the reconstruction can be done independently for each image, the parallel processor would allow for real-time 3D ultrasound reconstruction with volume rendering. This would provide a means to allow for 3D image to be rendered as each new frame is processed. A similar technique has been used by Ohbuchi [2], where images are incrementally reconstructed and volume rendered. To achieve the goal of real-time 3D ultrasound an incremental method in conjunction with a parallel processor would be required.

The reconstruction times on the DEC Alpha suggest that the current generation of processors are well suited for 3D Ultrasound reconstruction. The DEC reconstructed the data set in 2 seconds for the 6.4 M file and 4 seconds for the 10.4 M file. These reconstruction times are truly impressive and are appropriate for clinical use. In terms of frame rate the 6.4 M file contains 155 images, which is approximately 75 frames per seconds, and the 10.4 M file is 201 images longs, approximately 50 frames per seconds. The difference in frame rate is attributed to the larger x and y resolution of the larger file. Both rates are above real-time rates of 30 frames per second. Except for volume rendering, the DEC Alpha's performance, even including incremental viewing, allows for real-time reconstruction.

6.3 Ease of Use

The learning curve for the parallel processor is greater than that of the sequential CPU. More detail needs to be understood about multithreading and inter-processor communication when programming on a parallel processor. Programming for a sequential processor is much easier considering the programmer is not generally concerned with the underlying hardware. A gain is eminent with multithreaded architectures as shown with the two versions of image rebuilding used on the MANNA system.

6.4 Current Processors

Taking everything into perspective, one must realize at the time that the parallel processors were being designed and used for 3D ultrasound reconstruction, the processing power of the Workstations and personal computers were limited. For example, Exploroscan was developed in the early 80's, and the computational power of the PC was very minimal. Parallel processing was a definite requirement to achieve real-time image reconstruction. The older computer systems were not as powerful as the ones available today. A single Pentium II 333 processor is more powerful than 20 i860, and there is no comparison in the ease of use, and design time. But it is easy to disregard parallel processing now that we have Pentium class processors, or DEC Alphas. Looking at Table 6-1 and Figure 6-1, we can see the increase in processor performance in just the last few years

Parallel processors are slower to market when compared to sequential processors. For a company or research group to use a DEC Alpha 600 processor in a parallel processor, developing the actual machine is a lengthy process, since the memory architecture, cross network, and inter-node communications systems must all be designed. By the time the system is complete, the sequential processor market has released even more powerful CPUs. The parallel processor market is at a large disadvantage. Intel has announced that it will release a new processor capable of 1 GigaHertz [37]. There are applications that do require the processing power of parallel processors, such as weather predicting algorithms. Benefits of parallel processing are seen in dual/quad Pentiums that are invisible to the programs. The

programmer designs the algorithm for a sequential unit, but the operating system will handle the extra processors. Such systems are advantageous for simplicity and power gained.

If system cost is an issue, the use of a pipeline (Figure 2-2) could be used on a fast sequential computer. The data once gathered can be reconstructed in about 2.5 seconds on a Pentium II 333, and volume rendered in a short time thereafter. Based on the work conducted for this thesis, the reconstruction phase is not a limitation for high speed 3D ultrasound.

Table 6-1 - SPEC Benchmark results for Various Processors

Processor	SPECfp95	SPECint95
Pentium 133	3.50	4.01
Pentium 166	3.92	4.58
Pentium 200	4.32	5.17
Pentium 166 MMX	4.34	5.60
Pentium 200 MMX	4.87	6.44
Pentium 233 MMX	5.21	7.12
Pentium II 233	7.04	9.47
Pentium II 266	7.68	10.80
Pentium II 300	8.15	11.70
Pentium II 333	9.14	12.80
DEC 500	15.7	19.5
DEC 600	18.4	21.6

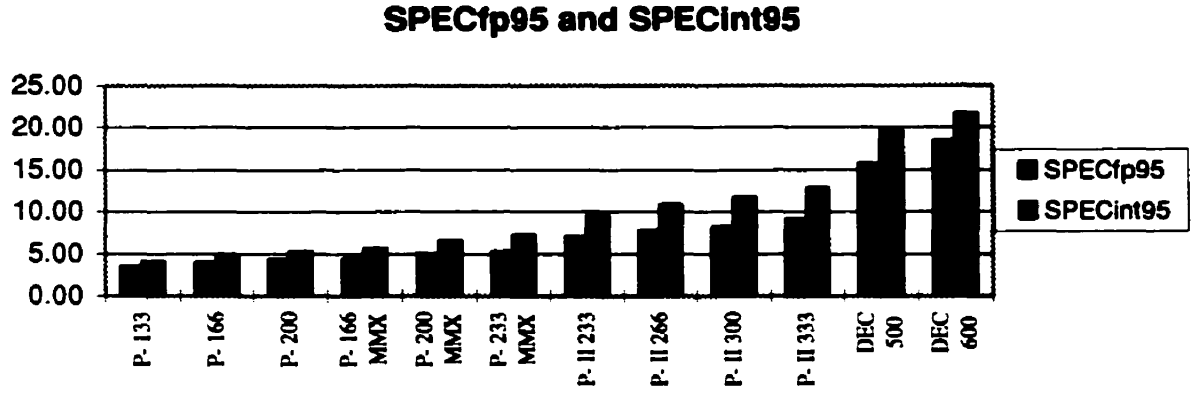

Figure 6-1 - SPEC Performance For Processors

A secondary goal of this thesis is to establish if the algorithm can be used for real-time displaying. This question has been answered by the implementation of the parallel processor. Though, the actual need for a parallel processor is not required for this type of application, the need to segment the data and reconstruct the image, proved that the image can also be displayed in real-time, provided the hardware is sufficiently efficient to process the data at real-time rates of 24 - 30 frames per second. With the current processor we are running at 15 frames per second on a Pentium 200 MHz processor. With the next generation of processors already on the market, reaching 24-30 frames/sec would require processor speeds of 1.6 - 2 times the Pentium 200 MHz processor. The next generation is already at these levels with the Pentium II 333 MHz processor (2.5 times as shown in Figure 6-1 and Table 6-1). The results for the Dec Alpha surpassed expectations attaining rates in the range of 50 - 75 frames/second.

7. Future Work

With the results attained with the DEC Alpha, research should be focused on fast or real-time volume rendering. An incremental reconstruction program could be designed to allow 3D real-time ultrasound viewing with volume rendering. Such a system would be well adapted for clinical use, expanding the potential and use of 3D ultrasound. A fixed point algorithm also should be developed for use on a UNIX system, which would further improve the execution time of the reconstruction on the DEC Alpha. The availability of real-time 3D ultrasound reconstruction is now not necessarily limited by technology and with more work the goal can be achieved.

Using parallel processors or possibly two DEC Alphas should provide the necessary computation power to attain real-time 3D interactive volume visualization. The computational power available today is more than 50 times than that was available only 6 years ago when the original 3D reconstruction algorithm was developed. Within a relatively short period of time, the use of real-time 3D interactive volume visualization can be realized with parallel processors, or maybe with the next generation of sequential processors.

A possible application of 3D ultrasound or more precisely a viewing method involves stereopsis viewing. Stereopsis is the true 3D viewing. Biologically, all humans have stereo viewing, since we each have two eyes. Each eye contributes

to our 3D view of the world. Patents [45] exist that allow an inherently 2D image be displayed as a 3D image. Such technology uses liquid crystal display shutter glasses. The operation of these glasses is fairly straight forward. One eye is covered, while the other eye can see-through. On the next cycle, the roles of each eye are reversed. At the same time, the image of the screen is shifted to the left or right depending on which eye is being shown. The shift within the image, acts as if our eyes were seeing a stereoscopic display [46]. It must be understood that the viewed image is a perceived 3D image, not a true 3D image. In the entertainment industry, the pseudo 3D image is a pleasure to view, but on a medical side, the perceived 3D is inadequate, since the surgeon needs accuracy when performing an operation.

An interesting application of high speed 3D reconstruction has been conducted at University of North Carolina [26, 47] and in Japan by Matani [48]. UNC's work focuses on obstetrics and gynecology, whereas the work from Japan concentrates on cardiac imaging. Their work focuses on superimposing 2D or 3D ultrasound images on a standard NTSC camera image. The video image is the view of the ultrasound operator. The video imaging camera records the view of the person using the ultrasound transducer. The overlaid ultrasound image is in the correct spatial position, projected into the body. The net result is the observer sees into the body with the ultrasound transducer. The main problem of the system used by UNC is that it only operates at 10 frames per second. A minimum of at least 15 frames is required to perceive smooth motion. Also the operator's visual

perception is limited to the video signal, since the virtual reality headset is enclosed for the operator. Thus, during a routine scan, this could be problematic for an ultrasound technician or doctor, since they would not be able to see the controls of the ultrasound machine. To date, the UNC group have not extended their mechanism to handle 3D images. The work in Japan is in 3D, and they use a liquid crystal shutter display to view the 3D images in 3D. The shutter display allows the operator to see images using stereopsis.

8. Conclusions

The objective of the thesis was to compare sequential processors to parallel processors in regard to 3D ultrasound reconstruction. 3D reconstruction is just one phase in a pipeline to 3D volume visualization. The computational power required to achieve real-time 3D volume visualization is immense and does require parallel processors. The reconstruction algorithm is well suited to operate on parallel machines, with the limitation of memory problems as demonstrated with the EARTH-Manna system.

The optimizations and conversion to fixed point proved to be beneficial in improving execution speed. The fixed point algorithm was nearly twice as fast as the floating point version, even with optimizations. Fixed point also was shown to be accurate and thus should and can easily replace the floating point version.

The execution speed of the reconstruction is at par of motorized 3D transducers with numerous advantages such as: portable, cheaper, and is easily adapted to any ultrasound system. Until true 3D probes are available, the use of 3D reconstruction is a useful tool to the medical diagnostic community. Even if 3D probes are available, they will be bulky and have the same weight and size issue of 3D motorized probes. Obstetrics and gynecology and rectal exams require smaller probes, so 3D transducer may not be suitable for these fields. 3D reconstruction will have a future regardless of the status of true 3D probes.

9. References

[1] R. Pini, M. Costi, L. Masotti, K.L. Novins, D.P. Greenberg, B. Greppi, M. Carofolini, and R.B. Devereux, Three-Dimensional (3D) Acquisition And Display Of Beating Heart Echo Images, *Acoutical Imaging*, Volume 20, pp.425-431, 1993.

[2] R. Ohbuchi and H. Fuchs. Incremental 3D Ultrasound Imaging From A 2D Scanner, *First Conference on Visualization in Biomedical Computing*, IEEE, pp. 360-367, 1990.

[3] D. Rotten, J.M. Levailaant, E. Constancis, A.C. Billion, Y. LeGuerinel, And P. Rua. Three-Dimensional Imaging Of Solid Breast Tumors With Ultrasound: Preliminary Data And Analysis Of Its Possible Contributions To The Understanding Of The Standard Two-Dimensional Sonographic Images, *Ultrasound in Obstet . Gynecol.*, 1:384-390, 1991.

[4] K.R. Stickels and S. Wann, An Analysis Of Three-Dimensional Reconstructive Echocardiography, *Ultrasound in Med. Biol.*, 10:575-580, 1984.

[5] K.L. Watkin, L.H. Baer, S. Mathur, R. Jones, S. Hakim, I. Diouf, B. Nuwayhid, S. Khalife, Three-Dimensional Reconstruction and Enhancement of Arbitrarily Oreinted and Positioned 2D Medical Ultrasonic Images, *CCECE/CCGEI '93*, pp. 1188-1195, 1993 .

[6] I. Shapiro, E. Drapkin, N. Halmann, O. Shochet, M. Graif, "Free-Hand 3D Ultrasound System", *http://reality.sgi.com/ofer/rsna_paper/rsna_paper.html.*, 1997

[7] O.T. von Ramm, S.W. Smith and H.G. Pavy, High Speed ultrasound volumetric imaging system, Parts I and II. *IEEE Trans. Ultrasonic, Ferroelectric and Frequency Control*, 38:100-115, 1991.

[8] M. Grimm, In ViVo-ScanNT Freehand 3D-Ultrasound Acquisition, Processing, and Visualization, *Computer Graphic Topics*, Issue 4, p11-13, 1997.

[9] D. H. Pretorius, M. House, T. R. Nelson, and K. A. Hollenbach, Evaluation of Normal and Abnormal Lips in Fetuses: Comparison between Three- and Two-Dimensional Sonography, *American Journal of Roentgenology (AJR)* , 165:1233-1237, 1995.

[10] *http://www.3dsono.org/3dsono/f_what3d.htm.*

[11] HDI 5000 Ultrasound System Reference Manual, ATL Ultrasound, Bothell, WA, USA, pp.4-39, 1997.

[12] James F. Brinkley, William E. Moritz, and Donald W. Baker. Ultrasonic Three-Dimensional Imaging And Volume From A Series Of Arbitrary Sector Scans. *Ultrasound in Med. and Biol.*, 4:317-327, 1978.

[13] R.N. Rohlings and A. H. Gee, Issues In 3-D Free-Hand Medical Ultrasound Imaging, *Cambridge University Engineering Department*, F-INFEND, Technical Report # 246, January 1996.

[14] Computer Graphics, J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, Addison-Wesley Publishing Company, New York, USA, 1996.

[15] G. Sakas, L. Schreyer, and M. Grimm, Preprocessing and Volume Rendering of 3D Ultrasonic Data, *IEEE Computer Graphics and Applications*, pp. 47-54, July 1995.

[16] R. Ohbuchi, D. Chen, H. Fuchs, Incremental Volume Reconstruction and Rendering for 3D Ultrasound Imaging, *Proceeding of the Visualization Biomedical Computing 1992*, October 13-16, 1992, Chapel Hill, NC., SPIE Proceedings Vol. 1808, pp. 312-323, 1992.

[17] M. Grimm, InViVo-ScanNT Freehand 3D-Ultrasound Acquisition, Processing and Visualization, *Computer Graphics Topics*, p.11-13, 1997(4).

[18] T. Ikedo and J. Ma, The Truga001: A Scaleable Rendering Processor, *IEEE Computer Graphics and Applications*, Vol. 18, No. 2, pp. 59-79, March/April 1998.

[19] N. Nitta, K. Hagihara and T. Shiina, Experimental Investigation of 3-D Blood Flow Velocity Measurements, *Jpn. J. Appl. Phys.*, Vol. 35, Part 1, No. 5B, pp. 3126-3130, 1996.

[20] M. Riccabona, T.R. Nelson, and D.H. Pretorius, Three-Dimensional Ultrasound: Accuracy Of Distance And Volume Measurements, *Ultrasound in Obstetrics and Gynecology*, Vol. 7, No. 6, pp. 429-434, June 1996.

[21] S.W. Hughes, T.J.D'Archy, D.J. Maxwell, W. Chiu, A. Milner, J.E. Saunders and R.J. Sheppard, Volume Estimation From Multiplanar 2D Ultrasound Images Using A Remote Electromagnetic Position And Orientation Sensor, *Ultrasound in Medicine and Biology*, Vol. 22, No. 5, pp. 561-572, 1996.

[22] S. Tong, D.B. Downey, H.N. Cardinal and A. Fenster, A Three-Dimensional Ultrasounnd Prostate Imaging System, *Ultrasound in Medicine and Biology*, Vol. 22, No. 6, pp. 735-746, 1996

[23] B. Delannoy, T. Torguet, C. Bruneel, E. Bridoux, J.M. Rouvaen, and H. LaSota, Acoustical Image Reconstruction In Parallel-Processing Analog Electronic Systems, *J. Appl. Phys.*, Vol. 50, 3153-3159, 1979.

[24] A. Koyano, Y. Yoshikawa, T. Konishi, Y. Kobayashi, Y. Kimita, T. Hidai, N. Okasda, C. Hayashi, and K. Fujie, A High Quality Ultrasound Imaging System Using Linear Array Transducer, *Ultrasound in Med. Biol.*, Vol. 8, Suppl. 1, pp. 100, 1982.

[25] D.P. Shattuck, M.D. Weinshenker, S.W. Smith, and O.T. von Ramm, Explososcan: A Parallel Processing Technique For High Speed Ultrasound Imaging With Linear Phased Arrays, *J. Acoust. Soc. Am.*, 75(4), pp. 1273 - 1282, April 1984.

[26] A. State, D.T. Chen, C. Tector, A. Brandt, H. Chen, R. Ohbuchi, M. Bajura, H. Fuchs, Case Study: Observing A Volume Rendered Fetus Within A Pregnant Patient, *Proceedings Visualization 1994*. IEEE, Los Alamitos, CA, USA, 94CH35707, pp. 364-368, 1994.

[27] L. Capineri, L. Masotti, S. Rocchi, F. Andreuccetti, M. Cerofolini, and A. Tondini, Nearly Real-Time Visualization Of Arbitrary Two-Dimensional Sections From Three-Dimensional Acquisition, *Ultrasound in Med. and Biol.*, Vol. 22, No. 3, pp. 319-328, 1996.

[28] S. Walter, 3Dultra - High-Performance Computing for Clinical Applications of 3D-Ultrasound, *Computer Graphics Topics*, 1997(4), p.8-10, 1997.

[29] G.T. Byrd, and M.A. Holliday, Multithreaded Processor Architectures, *IEEE Spectrum*, Aug. 1995, pp 38-46, 1995.

[30] R. R. Arche, "Win32 Multithreading Performance", *http://www.microsoft.com/win32dev/base/threadi.htm*, 1996.

[31] J.L. Gustafson, Reevaluating Amdahl's Law, *Communications of the ACM*, Volume 31, Number 5, May 1988, pp. 532-533, 1988.

[32] Computer Architecture, A Quantitative Approach, J. Hennessy and D.A.Patterson, Morgan Kaufmann Publishers, INC., San Francisco, California, USA, 1996

[33] O.C. Maquelin, H. Hum, G.R. Gao, Costs and Benefits of Multithreading with Off-the-Shelf RISC processors, *Proceeding of the ERO-PAR'95 - Parallel Processing*, Lecture Notes in Computer Science, Stockholm, Sweden, August 1995, pp. 117-128, 1995.

[34] H. Hum, K.B. Theobald, and G. R. Gao, Building Multithreaded Architectures with Off-the-Shelf Microprocessors, *Proceedings of the 8th IEEE International Parallel Processing Symposium (IPPS'94)*, Cancun, Mexico, April 23-26, 1994, pp. 288-294, 1994.

[35] *http://www.first.gmd.de*

[36] P. Hsieh, "Programming Optimizations", *http://www.geocities.com/SiliconValley/9498/optimize.html*.

[37] Intel Website, "Pentium Processor with MMX technology", *http://www.intel.com/mmx/index.htm*.

[38] C.M. Wittenbrink and A.K. Somani, 2D and 3D Optimal Parallel Image Warping, Journal of Parallel and Distributed Computing, Vol. 25, pp. 197-208, 1995.

[39] Contemporary Perspectives in Three-Dimensional Biomedical Imaging, *Eds.* C. Roux and L.L. Coatrieux, IOS Press, Netherlands, pp.79-105, 1997.

[40] A. Law and R. Yagel, *The Active Ray Approach to Rendering on Distributed Memory Multiprocessors*, Proceedings of Symposium on Parallel and Distributed Memory Achitectures, SPDP '96, New Orleans, Lousiana, October 1996.

[41] Real-Time Systems and Programming Languages, A. Burns and A. Wellings, Addison Wesley Longman, Harlow, England, 1997.

[42] D. A. Ellsworth, A New Algorithm for Interactive Graphics on Multicomputers, *IEEE Computer Graphics and Applications*, July 1994, pp. 33-40, 1994.

[43] S. Whitman, Dynamic Load Balancing for Parallel Polygon Rendering, *IEEE Computer Graphics and Applications*, July 1994, pp.41-48, 1994.

[44] U. Neumann, Communication Cost for Parallel Volume-Rendering Algorithms, *IEEE Computer Graphics and Applications*, July 1994, pp. 49-58, 1994.

[45] B.J. Garcia, Synthesized Stereoscopic Imaging System and Method, United States Patent Office, Patent # 5510832, April 23, 1996.

[46] M.T. Bolas, S. S. Fisher, and J.O. Merritt, Stereoscopic Displays and Virtual Reality Systems III, *Proceeding of the SPIE Volume 2653*, pp. 85-95, Feb. 2, 1996.

[47] M. Bajura, H. Fuchs and R. Ohbuchi, Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient, *Computer Graphics (Proceedings of the SIGGRAPH'92)*, Vol 26. No. 2, pp. 203-210, 1992.

[48] A. Matani, Y. Ban, O. Oshiro and K. Chihara, A system for Superimposing a 3-Dimensional Stereoscopic Motion Echo Image of the Heart onto the Chest, *Jpn. J. Appl. Phys.*, Vol. 35, Pt.1, No. 5B, pp. 3121-3125, 1996.

[49] Desai, R., Buckey, J.C, Pearce, J.A., Timing Specifications and Accuracy of the Real-Time 3D Echocardiographic Reconstruction System, *0-8186-6950-0/94, IEEE*, pp. 895-899, 1994.

[50] Hum, H.H.J., Maquelin, O., *et al.* A Design Study of the EARTH Multiprocessor, *PACT'95*, June 27-29, pp.59-68, 1995.

[51] Theobald, K, Gao, G.R., and Hendren, L.J., On the Limits of Program Parallelism and it Smoothability, *MICRO'25*, December 1-4, pp. 10-19, 1992.